

Andreas Meier

Traduit par Dac Hoa Nguyen

Collection IRIS

dirigée par Nicolas Puech



Introduction pratique aux bases de données relationnelles

Deuxième édition

 Springer

Introduction pratique aux bases de données relationnelles

Deuxième édition

Springer

Paris

Berlin

Heidelberg

New York

Hong Kong

Londres

Milan

Tokyo

Andreas Meier

Introduction pratique aux bases de données relationnelles

Deuxième édition

Traduit de l'allemand par : Dac Hoa Nguyen

 Springer

Andreas Meier
Département d'Informatique
Université de Fribourg
Boulevard de Pérolles 90
CH-1700 Fribourg
Suisse

Traduction de l'ouvrage allemand :

Relationale Datenbanken par Andreas Meier
Copyright © Springer-Verlag Berlin Heidelberg 1992,1995,1998
Tous droits réservés

ISBN-10 : 2-287-25205-3 Springer Paris Berlin Heidelberg New York
ISBN-13 : 978-2-287-25205-1 Springer Paris Berlin Heidelberg New York

© Springer-Verlag France 2002, 2006
Imprimé en France

Springer-Verlag France est membre du groupe Springer Science + Business Media

Cet ouvrage est soumis au copyright. Tous droits réservés, notamment la reproduction et la représentation, la traduction, la réimpression, l'exposé, la reproduction des illustrations et des tableaux, la transmission par voie d'enregistrement sonore ou visuel, la reproduction par microfilm ou tout autre moyen ainsi que la conservation des banques données. La loi française sur le copyright du 9 septembre 1965 dans la version en vigueur n'autorise une reproduction intégrale ou partielle que dans certains cas, et en principe moyennant les paiements des droits. Toute représentation, reproduction, contrefaçon ou conservation dans une banque de données par quelque procédé que ce soit est sanctionnée par la loi pénale sur le copyright.

L'utilisation dans cet ouvrage de désignations, dénominations commerciales, marques de fabrique, etc., même sans spécification ne signifie pas que ces termes soient libres de la législation sur les marques de fabrique et la protection des marques et qu'ils puissent être utilisés par chacun.

La maison d'édition décline toute responsabilité quant à l'exactitude des indications de dosage et des modes d'emploi. Dans chaque cas il incombe à l'utilisateur de vérifier les informations données par comparaison à la littérature existante.

SPIN: 11403517

À Heiri, Ramani et Tina

Avant-propos

Nous observons tous une constante mutation structurelle de l'emploi, marquée au cours du temps par le transfert de la main-d'oeuvre du secteur agricole vers le secteur des industries manufacturières, puis vers celui des services et du traitement de l'information. De nos jours, les métiers de l'information prédominent indéniablement sur les autres branches professionnelles. La main-d'oeuvre occupée à produire, traiter et diffuser l'information gagne toujours plus en importance. Dans plusieurs pays européens, le besoin en informaticiens de gestion qualifiés reflète cette évolution vers la société de l'information et une économie numérique.

Avec les infrastructures de télécommunication, les systèmes de gestion de bases de données, qui mettent à notre portée leurs langages et leurs fonctionnalités d'analyse de données, constituent les technologies clés de la société de l'information. Dans le commerce électronique, nous consultons les catalogues de produits orientés Web pour récolter des informations dans la phase préparatoire de vente. Nous accédons aux bases de données clientèle pour préparer des offres, élaborer et exécuter des contrats. Grâce aux bases de données en ligne nous supervisons la distribution de nos produits, quelle que soit leur nature, numérique ou matérielle. Un marketing personnalisé selon la clientèle est inconcevable sans base de données ou entrepôt de données.

Le présent ouvrage a pour but d'exposer un panorama de la technologie des bases de données avec une approche orientée vers la pratique, depuis la conception des bases de données relationnelles jusqu'au développement des systèmes de gestion de bases de données post-relationnelles. Après cette *introduction avancée aux systèmes de bases de données relationnelles*, l'informaticien qui désire oeuvrer dans ce domaine aura acquis les aptitudes suivantes :

- appliquer la technologie relationnelle à la modélisation des données et la conception des bases de données ;
- mettre en pratique les connaissances acquises sur les langages relationnels de requête et de manipulation de données ;
- comprendre les opérations internes et les mécanismes implantés dans un système de bases de données relationnelles ;
- identifier et intégrer dans ses réflexions les atouts et faiblesses de la technologie relationnelle, évaluer avec justesse les futurs développements.

Ce manuel s'adresse à la fois aux *praticiens*, aux *responsables de la formation* en entreprises, aux *enseignants* et *étudiants* des universités et des Grandes Écoles, et à tous ceux qui désirent *s'initier* à la technologie des bases de données relationnelles *par une approche pratique*. Une place centrale sera faite aux concepts essentiels qui sont souvent mal compris et incorrectement appliqués. Le livre est richement illustré par des figures simples et éloquentes. À la fin de chaque chapitre, une bibliographie sélective proposera au lecteur intéressé des ouvrages avancés pour approfondir les différents sujets abordés.

Ce livre d'introduction traitera de la technologie des bases de données relationnelles sous différents angles, englobant les méthodes de conception, les langages et les concepts fondamentaux de l'architecture des systèmes de bases de données relationnelles. Les thèmes seront développés indépendamment des produits offerts par les éditeurs de logiciels de bases de données pour *accorder une large place aux méthodes et techniques fondamentales* et mettre l'accent sur la compréhension liée à l'usage des bases de données relationnelles. Cette publication comble ainsi une lacune dans la littérature axée sur la pratique dans ce domaine.

La cinquième édition, revue et augmentée, présente un nouveau chapitre sur l'intégration des bases de données sur le Web qui comprend aussi la transformation des schémas de bases de données et la migration de données. Le chapitre 6, consacré aux bases de données postrelationnelles, contient une nouvelle section sur l'application de

la logique floue (bases de données floues). Une étude de cas dans le domaine du tourisme sera proposée : il s'agit d'une application de bases de données dont les étapes de conception et de mise en oeuvre avec Access seront expliquées en détail. Le chapitre de révision a été mis à jour et augmenté de nouvelles questions de compréhension accompagnées de solutions modèles.

Ce livre résulte d'un programme de formation en entreprise dans le domaine bancaire, enrichi par les discussions dans le cadre des cours académiques : «Praxis relationaler Datenbanken» (pour informaticiens et ingénieurs) à l'École Polytechnique Fédérale de Zürich, «Informationssysteme und Datenbanken» et «Informatique de gestion I» (pour étudiants en Informatique de gestion) à l'Université de Fribourg en Suisse. De nombreux experts d'entreprises et collègues universitaires ont contribué à la clarté du texte et la pertinence des figures. Je tiens à remercier Urs Bebler, Eirik Danielsen, Bernardin Denzel, Samuel Charles Fasel, Emmerich Fuchs, Peter Gasche, Caroline Grässle-Mutter, Michael Hofmann, Stefan Hüsemann, Günther Jakobitsch, Hans-Peter Joos, Klaus Küspert, Gitta Marchand, Michael Matousek, Thomas Myrach, Mikael Norlindh, Michel Patcas, Fabio Patocchi, Ernst-Rudolf Patzke, Thomas Rätz, Werner Schaad, August Scherrer, Walter Schnider, Max Vetter et Gerhard Weikum. Hartmut Wedekind a étudié l'ouvrage en détail et contribué, par ses précieuses suggestions, à la structure et au contenu de la cinquième édition. Mes compliments vont à Andreea Ionas dont le travail patient et minutieux a permis d'adapter le texte et les symboles à la nouvelle mise en page du livre. Anke Hees a assuré la relecture du livre dans un court délai. Mes remerciements s'adressent tout particulièrement à mon collègue, Dac Hoa Nguyen, qui, par ses compétences, son esprit critique et son inlassable travail, a réalisé de manière professionnelle la traduction de cet ouvrage. Je suis également reconnaissant à Springer-Verlag, en particulier à Nathalie Huilleret et Nicolas Puech pour leur collaboration très appréciée et notamment pour leur relecture attentive et leur correction du manuscrit.

Fribourg, juillet 2005

Andreas Meier

Table des matières

1	Vers un système de gestion de données	1
1.1	Principes fondamentaux du modèle relationnel	1
1.2	SQL, langage normalisé au niveau international.....	4
1.3	Les composants d'un système de bases de données relationnelles	8
1.4	Organisation de la mise en œuvre des bases de données.....	11
1.5	Notes bibliographiques	14
2	Les phases de la construction d'un modèle de données	17
2.1	De l'analyse à la base de données	17
2.2	Le modèle entité-association	20
2.2.1	Entités et associations.....	20
2.2.2	Les types d'associations	22
2.2.3	Généralisation et agrégation.....	25
2.3	Le schéma d'une base de données relationnelle	30
2.3.1	Le passage du modèle entité-association au schéma de base de données relationnelle	30
2.3.2	Règles de passage pour les ensembles de liens	33
2.3.3	Règles de passage pour la généralisation et l'agrégation	37
2.4	Les dépendances entre données et les formes normales.....	41
2.4.1	La signification et le but des formes normales.....	41
2.4.2	Les dépendances fonctionnelles	44
2.4.3	Les dépendances transitives	47
2.4.4	Les dépendances multivaluées	50
2.5	Les contraintes d'intégrité structurelles.....	53
2.6	L'architecture de données d'entreprise est vitale	57
2.7	Guide de la construction d'une base de données	61
2.8	Notes bibliographiques	64
3	Langages de requête et de manipulation des données	67
3.1	Exploitation d'une base de données	67
3.2	Les bases de l'algèbre relationnelle	69
3.2.1	Vue d'ensemble des opérateurs	69

3.2.2	Les opérateurs ensemblistes	71
3.2.3	Les opérateurs relationnels	75
3.3	Les langages relationnels complets	81
3.4	Aperçu des langages relationnels	83
3.4.1	SQL.....	83
3.4.2	QUEL.....	87
3.4.3	QBE	89
3.5	Les langages immergés.....	92
3.6	Traitement des valeurs nulles	94
3.7	La protection des données	96
3.8	La formulation des contraintes d'intégrité	100
3.9	Notes bibliographiques	103
4	Les composants de l'architecture d'un système de bases de données	105
4.1	Vue d'ensemble de l'architecture du système.....	105
4.2	Traduction et optimisation des requêtes	108
4.2.1	Construction d'un arbre d'interrogation.....	108
4.2.2	Optimisation des requêtes par transformation algébrique.....	111
4.2.3	Évaluation de l'opérateur de jointure	114
4.3	Fonctionnement d'un système de bases de données multi-utilisateur.....	118
4.3.1	Le concept de transaction	118
4.3.2	La sérialisabilité	120
4.3.3	Approches pessimistes	124
4.3.4	Approches optimistes	129
4.4	Structures de stockage et d'accès	131
4.4.1	Structures arborescentes	131
4.4.2	Méthodes de hachage	135
4.4.3	Structures de données multidimensionnelles	138
4.5	Traitement des erreurs	142
4.6	Architecture détaillée du système.....	145
4.7	Notes bibliographiques	148
5	Intégration et migration des bases de données	149
5.1	Exploitation d'ensembles de données hétérogènes	149
5.2	Les bases de données sur le Web	151
5.2.1	Création d'un système d'information orienté web	151
5.2.2	Documents et schémas XML.....	153
5.2.3	Le langage de requête XQuery.....	156

5.3	Règles de conversion pour l'intégration et la migration des données	158
5.3.1	Conversion des ensembles d'entités simples et des groupes répétitifs	159
5.3.2	Conversion des ensembles d'entités dépendants.....	161
5.3.3	Les conversions indirectes pour l'intégration et la migration des données.....	164
5.4	Variantes de migration des bases de données hétérogènes	167
5.4.1	Caractérisation des variantes de migration.....	168
5.4.2	Duplication des bases de données sous contrôle du système	171
5.5	Principes de la planification de l'intégration et de la migration.....	174
5.6	Notes bibliographiques.....	178
6	Les systèmes de bases de données post-relationnelles	181
6.1	Évolution future : pourquoi et dans quelle direction	181
6.2	Les bases de données réparties	182
6.3	Les bases de données temporelles	188
6.4	Les bases de données relationnelles-objet	192
6.5	Les bases de données multidimensionnelles	197
6.6	Les bases de données floues	203
6.7	Les bases de connaissances	210
6.8	Notes bibliographiques.....	214
	Révision.....	219
	La mise en œuvre d'une base de données avec Access :	
	l'agence de voyage travelblitz.....	231
	Glossaire	259
	Lexique anglais-français	267
	Bibliographie.....	273
	Index.....	285

Préface

Les Systèmes de Gestion de Bases de Données (SGBD) ont supplanté les Systèmes de Gestion de Fichiers au début des années 70. Ils apportaient des concepts nouveaux et essentiels : intégration des données, concurrence, reprise et confidentialité. Les premiers systèmes, définis par la norme CODASYL du DBTG étaient basés sur le modèle dit réseau. Défini au début des années 70, le modèle relationnel a vu ses premières versions industrielles dix années plus tard, et son véritable succès commercial vingt années plus tard. De nombreuses tentatives de le remplacer par un modèle plus élaboré ont eu lieu par la suite, mais aucun essai, soit de créer un nouveau vendeur de SGBD, soit d'introduire une technologie vraiment différente n'a réussi. Ni les modèles dit sémantiques, ni les bases de données objet, ne sont parvenus à s'imposer commercialement, soit parce que les vendeurs n'ont pu réussir commercialement, soit parce que le poids des systèmes patrimoine était trop fort. Aujourd'hui, une trentaine d'années après l'apparition des premiers systèmes, la technologie des systèmes de gestion de bases de données est mature et établie et représente une industrie de plusieurs milliards d'euros de chiffre d'affaire annuel. En étudiant cette industrie, on peut dégager plusieurs tendances :

Le modèle relationnel s'est imposé comme naturel et omniprésent. Le modèle simple introduit par Codd en 1970 est maintenant uniformément adopté comme standard de stockage de données secondaires. Il a le grand mérite de la simplicité, et de sa correspondance directe avec le modèle des tableurs connu et adopté par tous. Il a aussi l'avantage de l'existence d'une technologie de langage de requête largement basée sur le langage SQL. Il bénéficie aussi du long travail de standardisation du comité ANSI qui a produit les versions successives de la norme du langage. Donc, même si ce modèle n'est pas le plus naturel pour représenter des données de type

nouveau (XML, objets de langages tels que Java ou C++), des extensions ont été faites pour permettre de les accommoder.

Le marché s'est consolidé autour de quatre acteurs principaux. Après la disparition des acteurs initiaux tels Unify ou Interbase, après l'absorption d'acteurs clé comme Informix ou Ingres, il ne reste plus sur le marché que quatre vendeurs : Oracle, Microsoft, IBM et Sybase. Chacun de ces acteurs a une taille critique importante, investit de façon significative en recherche et développement, fournit une offre complète de produits et services associés, et a dans son portefeuille d'outils logiciels de nombreux autres produits complémentaires.

Les SGBD sont devenus des commodités. Les systèmes de gestion de bases de données font maintenant partie du paysage traditionnel du logiciel. Ils se sont imposés irrévocablement tant pour les applications spécialisées que pour les progiciels horizontaux ou verticaux. Pratiquement aucune application n'est aujourd'hui développée au-dessus d'un gestionnaire de données autre qu'un SGBD. Dès qu'il y a besoin d'une grande quantité de données, dès qu'il y a un partage de données entre plusieurs utilisateurs différents, dès qu'il y a un besoin de fiabilité des données, dès qu'il y a besoin de sécurisation ou de confidentialité des données, l'application est développée avec un SGBD. Le même phénomène s'est produit pour les progiciels commerciaux, qu'ils soient verticaux, comme des systèmes de CAO, des systèmes de gestion de documents, ou horizontaux, comme des systèmes de gestion du personnel, des systèmes de paye ou des systèmes de gestion de type ERP (enterprise resource planning) ou SCM (supply chain management) : chacun de ces progiciels, soit contient une version incluse de SGBD, soit doit pour fonctionner être lié à l'exécutif d'un SGBD.

La technologie continue à évoluer. Paradoxalement pour une technologie mature et parvenue au stade de la commodité, l'évolution technologique reste importante. Elle est due à des facteurs multiples : pression de la concurrence qui reste vive entre les quatre acteurs du marché, investissements technologiques importants consentis par les vendeurs, et demandes nouvelles du marché. Des extensions sont proposées régulièrement et progressivement intégrées dans les

systemes, que se soit pour prendre en compte des nouveaux phénomènes comme l'apparition de XML, de nouveaux langages de programmation comme Java dans les années 90 ou C++ dans les années 2000. Il convient donc de comprendre comment cette technologie évolue et ce que sont les nouveaux produits.

Il est donc essentiel pour tous les professionnels de l'informatique de comprendre et de maîtriser cette technologie. Ce livre présente l'ensemble des éléments nécessaires à cette maîtrise et cette compréhension. Il présente un panorama moderne et mis à jour du modèle relationnel, des notions de modélisation de données, des langages de requêtes essentiels. Il couvre de façon complète et détaillée l'ensemble des concepts essentiels des systèmes : traitement de requêtes, structures de stockage et méthodes d'accès, concurrence et reprise. Il constitue donc un outil essentiel pour tout enseignant, chercheur, ou praticien qui s'intéresse à cette technologie.

Andreas Meier allie une profonde connaissance de la théorie des bases de données (il est l'auteur de nombreux articles sur le sujet), à une expérience détaillée de l'utilisation pratique des SGBD (il a participé dans l'industrie à la mise en place de grands projets d'application). Ce double point de vue lui permet de mettre en valeur dans cet ouvrage tant les fondements théoriques essentiels à la maîtrise de cette technologie que l'aspect concret indispensable à l'utilisateur de ces systèmes.

François Bancilhon

Préface à l'édition allemande

Pour un ouvrage spécialisé en informatique, la parution de la cinquième édition marque un événement rare qui mérite que nous nous interroguions sur les raisons de cette remarquable pérennité.

La réponse suivante s'avère appropriée, mais encore trop simplifiée : Andreas Meier propose un livre écrit de manière compréhensible aux praticiens et aux apprenants grâce à son orientation vers la pratique. Explorons la question corollaire suivante : Pourquoi le livre de Meier est-il compréhensible ? J'apporterai une réponse plus élaborée à cette interrogation :

Pour les apprenants, la bonne compréhensibilité du présent ouvrage repose sur trois caractéristiques. Premièrement, l'auteur s'adresse aux lecteurs de niveau débutant dans le domaine des bases de données. Le langage et les figures s'harmonisent tout au long du livre. L'apprenant débutant ne se perd pas dans des abstractions de haut niveau avec toute la précision requise. Deuxièmement, l'auteur présente la matière aux apprenants dans un contexte où elle devrait se placer, tant dans sa profondeur que dans son ampleur. À cet égard, la recherche académique et l'application industrielle sont deux contextes fondamentalement différents. Meier met en exergue l'application industrielle. Troisièmement, le souci de "justesse" est un point sensible pour tout auteur : elle signifie non pas simplement la vérité logique, mais la conformité aux approches de modélisation reconnues qui sont en constante évolution. Le monde des bases de données, issues des traditionnels systèmes de fichiers implantés dans les systèmes d'exploitation, a connu une évolution continue des modèles depuis plus de trois décennies. Nous étions partis du modèle CODASYL, nous avons ensuite découvert l'approche relationnelle, et l'Internet met aujourd'hui à notre portée le plus grand système de bases de données que nous puissions imaginer. Prédire les conséquences dans le futur n'est pas une tâche facile. La seule

certitude est que nous ne nous dirigeons pas vers des eaux calmes. Dans cette perspective, l'auteur a augmenté la cinquième édition d'un important chapitre traitant de l'intégration des bases de données hétérogènes sur le Web.

Je souhaite que la cinquième édition touche un large lectorat. Devant l'évolution constatée ci-haut, il n'y a pas à douter qu'une sixième édition se profilera à l'horizon.

Erlangen, juillet 2003

Hartmut Wedekind

1 Vers un système de gestion de données

1.1 Principes fondamentaux du modèle relationnel

La table est une forme simple et parlante pour rassembler des données ou représenter des informations. La forme tabulaire nous étant familière, il est aisé d'interpréter sa structure au premier coup d'œil.

Par exemple, la table de la figure 1-1 est conçue pour recueillir des informations sur les employés d'une entreprise. Elle porte un nom écrit en majuscule, EMPLOYÉ. Les noms de colonnes désignent les attributs des employés. Dans notre table, ce sont le numéro d'employé «E#», le nom de l'employé «Nom» et son domicile «Ville».

Représentation de l'information sous forme tabulaire

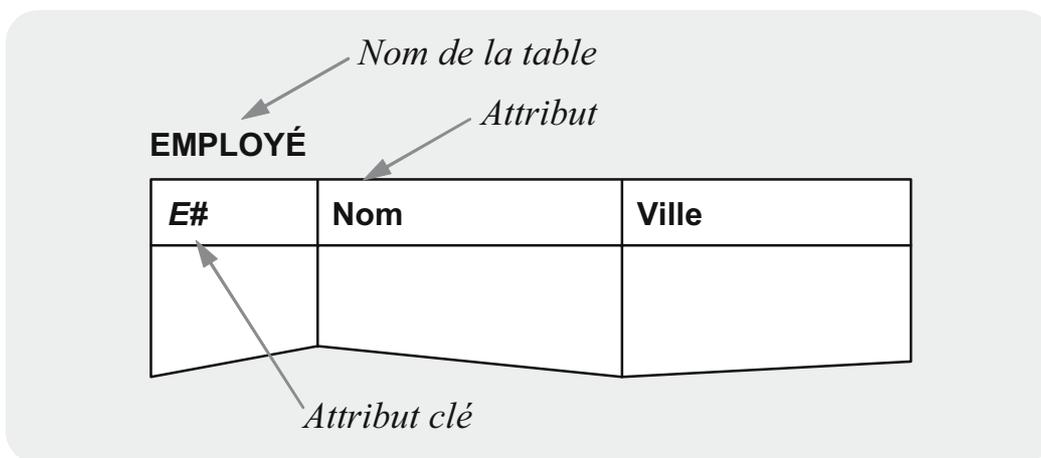


Figure 1-1
Structure de la table EMPLOYÉ

Chaque *attribut* (*attribute*, en anglais) représente une propriété dont les valeurs entrées dans la table appartiennent à un *domaine* prédéfini (*domain*, en anglais). Dans la table EMPLOYÉ, l'attribut E# permet d'identifier chaque employé de manière unique. En vertu de cette propriété, nous déclarons que le numéro de l'employé est une clé. Pour mettre en évidence un attribut clé, son nom apparaîtra

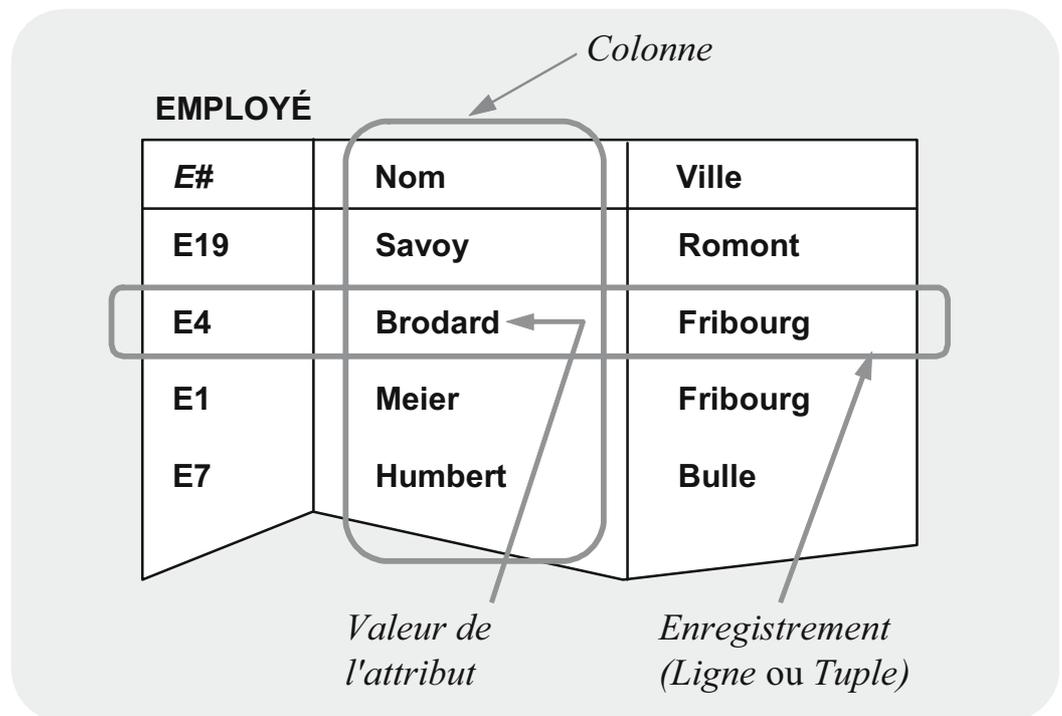
Les attributs définissent les propriétés

désormais en italique dans l'en-tête de la table¹. L'attribut Ville prendra comme valeurs les noms des localités, et l'attribut Nom, les noms des employés.

Une table consiste en un ensemble d'enregistrements

Introduisons maintenant les données du personnel dans la table EMPLOYÉ ligne par ligne, comme indiqué dans la figure 1-2. Nous observons que certaines valeurs apparaissent plusieurs fois dans la table. Ainsi, le lieu de domicile Fribourg est présent deux fois dans la table EMPLOYÉ. Cette double apparition traduit un fait important : les employés Brodard et Meier habitent tous les deux à Fribourg. Dans la table EMPLOYÉ, non seulement les noms des localités mais aussi ceux des employés peuvent apparaître plusieurs fois. Pour cette raison, l'attribut clé E# est indispensable pour identifier chaque employé dans la table de manière unique.

Figure 1-2
La table EMPLOYÉ et les valeurs des attributs



Que signifie une clé d'identification ?

Une *clé d'identification* ou *clé* (*identification key*, en anglais) dans une table est un attribut ou une combinaison minimale d'attributs dont les valeurs clés permettent de désigner chaque enregistrement (appelé aussi *ligne* ou *tuple*) à l'intérieur de la table de manière

¹ Dans les ouvrages de référence sur les bases de données, une autre convention consiste à souligner les attributs clés pour les mettre en évidence.

unique. Deux *propriétés de clé* importantes découlent de cette définition succincte :

- Chaque valeur clé identifie de manière unique un enregistrement dans la table ; en d'autres termes, des enregistrements différents ne doivent pas prendre une valeur clé identique (*unicité*).
- Si la clé est constituée de plusieurs attributs, cette combinaison doit être minimale ; en d'autres termes, aucun attribut de la combinaison ne peut être retiré sans violer l'unicité de l'identification (*minimalité*).

Contraintes d'unicité et de minimalité

Une clé se caractérise entièrement par ces deux contraintes d'unicité et de minimalité.

Au lieu d'un attribut naturel ou d'une combinaison d'attributs naturels, nous pouvons définir une clé par un attribut artificiel. Dans notre exemple, le numéro d'employé E# est artificiel parce qu'il ne désigne aucune propriété naturelle de l'employé. Pour diverses raisons, nous sommes souvent réticents au choix de *clés artificielles* ou «numéros» comme attributs d'identification, surtout s'il s'agit d'informations sur des personnes. Mais les clés ou les combinaisons de clés naturelles ont aussi leur inconvénient, notamment en matière de protection des données. Considérons par exemple le numéro d'assurance-vieillesse et survivants², bien connu en Suisse (numéro AVS) : il contient, entre autres, la date de naissance de l'assuré. Dès lors, sur le plan de la protection des données nous nous demandons s'il est judicieux de choisir comme attribut d'identification le numéro AVS qui inclut des informations sur des personnes et qui figure dans de nombreux documents et pièces d'identité.

La prudence s'impose dans la définition d'une clé

Sur la base de ces réflexions, une clé artificielle doit être *indépendante des applications et dépourvue de toute signification* (sémantique). Si les valeurs d'un attribut permettent d'en déduire des informations précises, cela ouvre une brèche qui met en péril la sécurité d'une base de données. En outre, la signification initiale d'une

Les clés artificielles ont des avantages indéniables

² L'assurance-vieillesse et survivants (AVS) est obligatoire et forme avec l'assurance-invalidité le principal pilier du système de sécurité sociale suisse.

valeur clé peut évoluer voire disparaître au cours du temps. C'est ainsi qu'une clé dont le choix initial paraît évident pourrait violer un jour la contrainte d'unicité.

Définition d'une table

Les propriétés d'une relation

Une *table* ou *relation* (*table*, *relation*, en anglais) est un ensemble de tuples représenté sous la forme tabulaire et ayant les propriétés suivantes :

1. Chaque table porte un nom unique.
2. À l'intérieur de la table, le nom de chaque attribut est unique et désigne une colonne avec des propriétés spécifiques.
3. Une table peut contenir un nombre quelconque d'attributs, l'ordre des colonnes dans la table est indifférent.
4. L'un des attributs ou une combinaison d'attributs identifie de façon unique chaque tuple dans la table et sera la clé primaire.
5. Une table peut contenir un nombre quelconque de tuples, l'ordre des tuples dans la table est indifférent.

Selon cette définition, dans un *modèle relationnel* (*relational model*, en anglais) chaque table est vue comme un *ensemble non ordonné de tuples*. En vertu de cette notion ensembliste, un tuple ne peut donc apparaître qu'une seule fois dans une table.

1.2 SQL, langage normalisé au niveau international

Langage de base de données ensembliste

Comme nous l'avons déjà vu, un modèle relationnel représente les informations sous la forme tabulaire. À chaque table est associé un ensemble de tuples ou d'enregistrements de même type. Ce concept d'ensemble permet d'implémenter des *opérations ensemblistes pour interroger et manipuler les bases de données*. Ainsi, le résultat d'une opération de sélection est un ensemble ; en d'autres termes, chaque interrogation de la base de données génère un résultat sous la forme d'une table. Si la table consultée ne contient aucun tuple répondant

aux propriétés requises, le résultat obtenu par l'utilisateur est une table vide. Les opérations de mise à jour portent également sur une table ou un groupe de tables spécifiques.

Le plus important langage de requête et de manipulation des tables s'appelle *Structured Query Language* ou SQL en abrégé (voir figure 1-3). Les normes du langage SQL sont élaborées par deux organismes, l'ANSI (American National Standards Institute) et l'ISO (International Organization for Standardization)³.

SQL est un langage normalisé

Le langage SQL définit une structure syntaxique générale, illustrée par la requête développée dans la figure 1-3 :

«Sélectionner (SELECT) l'attribut Nom
de (FROM) la table EMPLOYÉ
suivant le critère (WHERE) du domicile à Fribourg !»

L'instruction SELECT-FROM-WHERE traite une ou plusieurs tables et génère toujours une table comme résultat. Dans notre exemple, le résultat de la requête précédente consiste en une table contenant les noms recherchés, Brodard et Meier.

Chaque requête produit une table résultat

C'est par son approche ensembliste que SQL se distingue essentiellement des langages non relationnels. En outre, il présente un avantage important pour l'utilisateur : chaque requête SQL déclenche toute une suite d'opérations dans un système de bases de données. Ainsi, l'utilisateur n'a pas besoin de «programmer» lui-même la procédure de sélection. Le système de bases de données relationnelles exécute ce travail à sa place.

SQL est un langage ensembliste

³ L'ANSI, organisme national de standardisation américain, est l'homologue du DIN (Deutsches Institut für Normung) en Allemagne et de l'AFNOR (Association Française de Normalisation) en France. Les organismes de standardisation nationaux sont membres de l'ISO.

Figure 1-3
Formulation d'une
requête en SQL

EMPLOYÉ

E#	Nom	Ville
E19	Savoy	Romont
E4	Brodard	Fribourg
E1	Meier	Fribourg
E7	Humbert	Bulle

Un exemple d'interrogation
«Sélectionner les noms des employés qui habitent à Fribourg»

Formulation de la requête en SQL

```
SELECT Nom
FROM EMPLOYÉ
WHERE Ville = 'Fribourg'
```

Table résultat

Nom
Brodard
Meier

C'est le **QUOI** qu'il faut définir sans se préoccuper du **COMMENT**

Les langages relationnels de requête et de manipulation de données sont *descriptifs*. Pour obtenir les résultats désirés, il suffit à l'utilisateur de définir les critères qu'ils doivent satisfaire. Il n'est pas nécessaire de spécifier les modalités d'extraction des enregistrements qui respectent ces critères. Le système de bases de données se charge des opérations pour traiter une requête ou une manipulation de données par des méthodes de recherche et d'accès adéquates en vue de produire les tables résultats souhaitées.

Contrairement au cas des langages de requête et de manipulation descriptifs, les utilisateurs des langages de bases de données procéduraux doivent programmer le déroulement des opérations pour obtenir les informations recherchées. Le résultat de chaque requête consiste en un seul enregistrement au lieu d'un ensemble de tuples.

La navigation dans un ensemble de données est assurée par le système

Dans la formulation descriptive d'une requête en SQL, il suffit de spécifier les critères de sélection désirés dans la clause WHERE ; en revanche, avec les langages procéduraux l'utilisateur doit programmer un algorithme de recherche des enregistrements. En nous référant par exemple aux langages connus des systèmes de bases de données hiérarchiques, la figure 1-4 montre que nous devons d'abord coder

GET_FIRST pour obtenir un premier enregistrement qui répond au critère de recherche. Ensuite, la commande GET_NEXT est programmée pour lire tous les enregistrements jusqu'à ce que nous arrivions à la fin du fichier ou au niveau hiérarchique suivant dans la base de données.

Langage naturel :

```
«Sélectionner les noms des employés  
qui habitent à Fribourg»
```

Langage descriptif :

```
SELECT Nom  
FROM EMPLOYÉ  
WHERE Ville='Fribourg'
```

Langage procédural :

```
get first EMPLOYÉ  
  search argument (Ville = 'Fribourg')  
while status = 0 do  
begin  
  print(Nom)  
  get next EMPLOYÉ  
  search argument (Ville = 'Fribourg')  
end
```

Figure 1-4
Différence entre
langage descriptif
et langage
procédural

En résumé, les langages de bases de données procéduraux impliquent l'usage de commandes orientées enregistrement ou basées sur la navigation pour traiter les données. Cela requiert de la part des développeurs d'applications l'expérience et la connaissance de la structure interne de la base de données. En outre, un utilisateur occasionnel ne serait pas en mesure d'interroger une base de données de manière autonome. Au contraire des langages procéduraux, les langages de requête et de manipulation de données relationnels ne nécessitent aucune spécification des chemins d'accès, des procédures de traitement ou de navigation. Par conséquent, le coût de développement des applications de bases de données est sensiblement réduit.

La structure de
données physique
est cachée

Une chance pour l'utilisateur occasionnel

L'approche descriptive est d'une importance capitale pour déléguer aux départements de l'entreprise et aux utilisateurs finaux la tâche de formuler eux-mêmes les requêtes pour interroger les bases de données. Des études menées sur les interfaces descriptives des bases de données ont montré que c'est grâce aux éléments descriptifs d'un langage qu'un *utilisateur occasionnel est tout à fait capable* de formuler lui-même les requêtes adaptées à ses besoins. Par ailleurs, la figure 1-4 révèle que le langage SQL est proche du langage naturel. Actuellement, il existe aussi des systèmes de bases de données relationnelles qui disposent d'une interface en langage naturel.

1.3 Les composants d'un système de bases de données relationnelles

Ted Codd est le père du modèle relationnel

Le modèle relationnel a été conçu par Codd au début des années 1970. Les premiers systèmes de bases de données ont été bâtis dans des centres de recherche et prennent en charge le langage SQL ou d'autres langages de requête similaires. Avec le temps les produits issus de la technologie relationnelle ont atteint un haut degré de maturité et conquis le monde de la pratique.

Qu'est-ce qu'un SGBD relationnel ?

Un *système de gestion de bases de données relationnelle (relational database management system, en anglais)*, SGBDR en abrégé, souvent appelé simplement système de bases de données relationnelles, est un système intégré pour la gestion unifiée des bases de données relationnelles, comme le montre la figure 1-5. Un SGBDR dispose de fonctions utilitaires d'une part, et d'un langage descriptif pour la définition et la manipulation de données d'autre part.

Les fonctions des composants de stockage et de gestion

Un système de bases de données relationnelles est constitué d'un composant de stockage et d'un composant de gestion de données (voir figure 1-5) : le *composant de stockage* a pour but de réunir dans des tables l'ensemble des données et tous les liens qui les unissent. On distingue d'une part les tables qui contiennent des données appartenant aux applications des utilisateurs, d'autre part les tables systèmes indispensables au fonctionnement d'une base de données. Les tables systèmes contiennent les définitions de données que les utilisateurs peuvent consulter à tout moment sans être autorisés à les

modifier. Le *composant de gestion* comporte essentiellement un langage relationnel de définition et de manipulation des données. Ce composant englobe aussi des fonctions utilitaires telles que la restauration de la base de données en cas de panne, la protection et la sécurité des données.

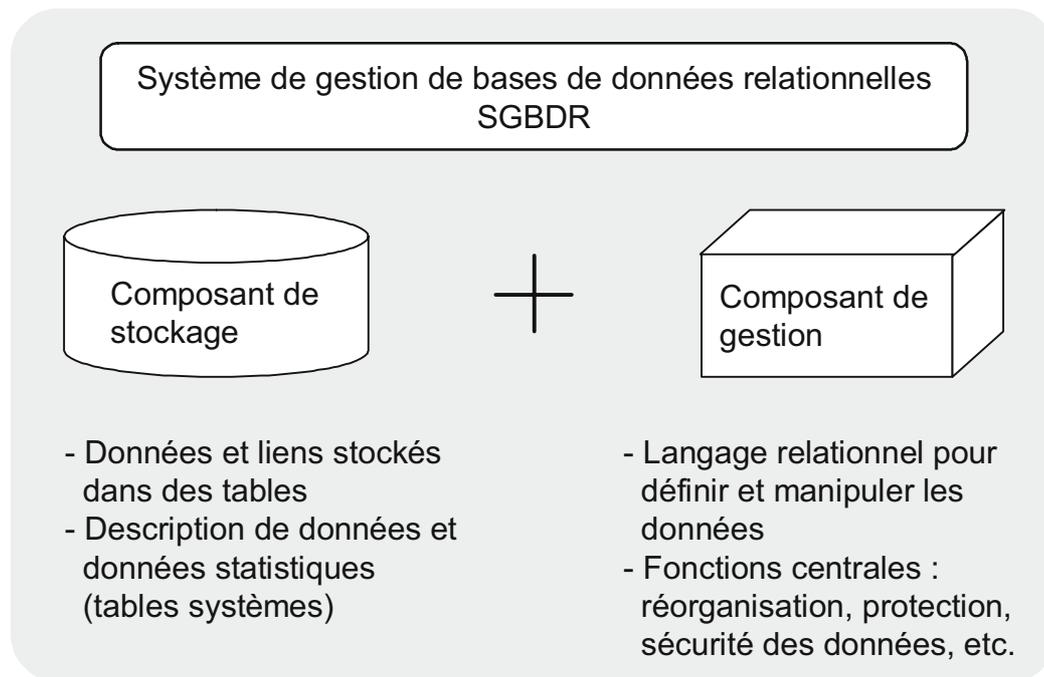


Figure 1-5
Les deux composants d'un système de bases de données relationnelles

Un système de bases de données relationnelles se caractérise par les propriétés suivantes :

- Un SGBDR permet une organisation structurée des données, fondée sur une *base formelle claire*. Toutes les informations sont stockées dans des tables. Des dépendances entre les valeurs d'attributs dans une table, ou la présence d'informations redondantes sont identifiées. Ces instruments formels permettent une conception cohérente des bases de données et garantissent des structures de données propres.
- Un SGBDR dispose d'un langage ensembliste pour la définition et la manipulation de données. C'est un langage descriptif qui vise à faciliter la formulation des requêtes par l'utilisateur en le déchargeant des tâches de programmation. L'utilisateur définit dans ce langage un critère de sélection en fonction de ses besoins d'information. Il appartient ensuite au système de bases de

Les SGBDR reposent sur une base formelle

Les langages de requête relationnels sont de nature ensembliste

données d'effectuer la recherche dans la base et de produire une table résultat.

*Indépendance
entre l'organisation
des données et les
programmes
d'application*

- Un SGBDR garantit une grande indépendance des données, c'est-à-dire la nette séparation entre celles-ci et les programmes d'application. Cette propriété résulte du fait que le composant de gestion dissocie les applications du composant de stockage dans un SGBDR. Une totale indépendance des données permet idéalement d'apporter des modifications physiques dans une base de données relationnelle sans entraîner une adaptation conséquente des programmes d'applications.

*Plusieurs
utilisateurs peuvent
travailler
simultanément*

- Un SGBDR fonctionne dans un environnement multi-utilisateur en permettant à plusieurs personnes d'interroger ou de traiter simultanément une même base de données. Par conséquent, un système de bases de données relationnelles doit être capable de gérer des opérations concurrentes pour éviter qu'elles soient en conflit entre elles ou compromettent la justesse des données.

*La cohérence et
l'intégrité des
données sont
assurées*

- Un SGBDR dispose de mécanismes garantissant l'intégrité des données. Ce terme englobe le stockage de données sans erreur, la protection contre les destructions, les pertes, les abus et les accès non autorisés.

*Les SGBDR
dominent le
marché*

Les systèmes de bases de données non-relationnelles ne satisfont que partiellement aux propriétés que nous venons de présenter. De ce fait, les systèmes de bases de données relationnelles ont réussi une percée sur le marché ces dernières années et continuent à gagner du terrain. On assiste à une amélioration continue de leurs *performances* (*performance*, en anglais) malgré le prix à payer qui découle de la conception ensembliste du traitement. L'expérience pratique des systèmes de bases de données relationnelles a donné une impulsion aux nouveaux développements dans le domaine des bases de données réparties et des bases de connaissances et de méthodes. En outre, la théorie des bases de données relationnelles est à l'origine de nombreux travaux de recherche et de développement dans ce domaine (voir chapitre 6).

1.4 Organisation de la mise en œuvre des bases de données

Pour de nombreuses entreprises et institutions, la masse de données qu'elles détiennent constitue une *ressource vitale*. Le besoin d'information nécessite non seulement la maintenance des données qui leur appartiennent, mais aussi l'usage croissant de données publiquement accessibles. Des fournisseurs d'informations proposent leurs services sans interruption. Leur croissance persistante et sur un plan mondial illustre bien l'intérêt des entreprises pour cette catégorie de données.

*L'information
comme facteur de
production*

L'importance des informations tenues à jour et reflétant la réalité exerce une influence directe sur l'organisation du domaine informatique. Nous assistons ainsi à la création de postes en gestion de données, qui traduit la prise de conscience des responsabilités et des tâches relatives aux données. La gestion moderne des données se préoccupe d'une part de la création et de l'exploitation des informations au niveau stratégique, et d'autre part, de la mise à disposition et du traitement efficace des données actuelles et cohérentes au niveau opérationnel.

*Le rôle primordial
d'une bonne
gestion des
données*

La gestion de données entraîne des coûts élevés de mise en œuvre et d'exploitation, et engendre des bénéfices difficiles à mesurer au début. Plus précisément, il n'est pas évident, dans nos réflexions économiques, d'évaluer avec exactitude les avantages d'un modèle de données bien structuré, des définitions de données non conflictuelles et compréhensibles à tous, des données propres et cohérentes, des concepts de sécurité pertinents, de la disponibilité des informations tenues à jour, et d'une multitude d'autres facteurs. Seule une prise de conscience de l'importance et de la pérennité des données amène l'entreprise à consentir des investissements nécessaires.

*La pérennité des
données doit être
assurée*

Pour mieux saisir le concept de *gestion des données* (*data management*, en anglais), il convient tout d'abord de la décomposer en quatre catégories de fonctions : l'architecture de données, l'administration de données, la technologie et l'exploitation des

*Fonctions et
obligations en
matière de gestion
des données*

données. La figure 1-6 présente les objectifs et les outils attribués à ces quatre domaines.

Figure 1-6
Les quatre piliers
de la gestion des
données

	Objectifs	Outils
Architecture de données	Élaborer et entretenir le modèle de données d'entreprise. Contribuer à la modélisation des données dans le développement des applications	Analyse de données et méthodologie de conception. Outils de modélisation de données assistée par ordinateur
Administration de données	Gérer les données et les fonctions conformément aux directives de standardisation et aux normes internationales. Conseiller les développeurs et les utilisateurs	Dictionnaires de données. Outils de contrôle de l'utilisation des données
Technologie de données	Installer, réorganiser et sauvegarder les bases de données. Restaurer les bases de données après panne	Systèmes de gestion de bases de données. Outils de restauration des bases de données et d'optimisation de la performance
Exploitation de données	Mettre à disposition les outils d'analyse et de reporting en veillant à la protection et à la propriété des données	Langages de requête et de manipulation de bases de données. Générateurs d'états

Le rôle fondamental et stratégique de l'architecture de données

Dans le domaine de *l'architecture de données*, les tâches consistent à analyser, classifier et structurer les données de l'entreprise par des méthodes confirmées. Outre l'analyse des données et des besoins d'informations, les principales classes de données et leurs liens réciproques doivent être définis de manière détaillée dans des modèles de données. Nés de l'abstraction du monde réel, ces modèles qui se complètent constituent la base de l'architecture de données.

Il faut gérer les définitions et les formats de données

L'administration de données vise à coordonner et superviser les définitions et les formats de données, à en déterminer clairement les responsabilités afin de garantir que les données pérennes de l'entreprise soient disponibles à l'ensemble des applications. Face à la tendance actuelle vers le stockage décentralisé des données soit sur des stations de travail intelligentes soit sur des systèmes

départementaux, l'administrateur de données assume une responsabilité de plus en plus grande dans la maintenance des données et la gestion des autorisations d'accès.

L'expert en *technologie de données* a pour tâches d'installer, de superviser et de réorganiser les bases de données, d'assurer leur protection par une stratégie de sécurité à plusieurs niveaux. Ce domaine, appelé aussi technologie de bases de données ou administration de bases de données, inclut la gestion des technologies dans une perspective d'évolution au rythme des progrès continuels en matière de bases de données, et de perfectionnement des méthodes et outils existants.

Les défis technologiques dans la gestion des données

Le quatrième pilier, *l'exploitation des données*, comprend les fonctions qui permettent l'usage proprement dit des données de l'entreprise. Un service de soutien aux utilisateurs, assuré éventuellement par une unité appelée infocentre, accompagne spécialement les départements fonctionnels de l'entreprise pour qu'ils puissent entretenir et exploiter leurs propres données de manière autonome.

L'infocentre gère l'exploitation des données

Cet examen des fonctions et des responsabilités liées à la gestion de données nous amène à la définition suivante :

La gestion de données désigne l'ensemble des fonctions opérationnelles et techniques que sont l'architecture, l'administration et la technologie de données pour assurer le stockage, la maintenance et l'usage de l'ensemble des données de l'entreprise.

Définition de la gestion de données

Le concept ainsi défini englobe à la fois les fonctions techniques et opérationnelles. Ceci ne signifie pas nécessairement que l'architecture, l'administration et la technologie de données, ainsi que le service aux utilisateurs, doivent être centralisées dans une seule unité organisationnelle sur l'organigramme d'une entreprise.

Pas de gestion de l'information sans la gestion des données

1.5 Notes bibliographiques

Ouvrages de référence, en anglais et en français, sur les systèmes de bases de données

L'abondance des ouvrages de référence publiés sur les bases de données traduit l'importance accordée à ce domaine de l'informatique. Certains manuels abordent non seulement les systèmes de gestion de bases de données relationnelles, mais aussi les modèles de données hiérarchiques et en réseau qui sont encore utilisés de nos jours. Parmi les grands classiques, nous pouvons citer Connolly et Begg (2005), Date (2005), Hoffer et al. (2004). L'ouvrage d'Ullman (1982) met l'accent sur l'aspect théorique et celui de Silberschatz et al. (2005) est très instructif. Les travaux publiés par Elmasri et Navathe (2004), Ramakrishnan et Gehrke (2003) constituent des références complètes. Gardarin et Valduriez (1989) présentent une introduction à la technologie des bases de données relationnelles et au domaine des bases de connaissances. Banos et Mouyssinat (1991) expliquent les systèmes de bases de données hiérarchiques, en réseau et relationnelles. Delobel et al. (1991) définissent les concepts des systèmes de bases de données relationnelles et orientées objet. Gardarin (2000), Gardarin (2003), Miranda (2002) traitent des fondements des systèmes de bases de données hiérarchiques, en réseau, relationnelles, déductives et orientées objet.

Livres de texte en allemand sur les systèmes de bases de données

Parmi les livres de texte publiés en allemand sur les bases de données, mentionnons Schlageter et Stucky (1983), Vossen (2000), Wedekind (1991) et Zehnder (2002). Les récentes publications de Heuer et Saake (2000) et de Kemper et Eickler (2001), ainsi que le livre de Lang et Lockemann (1995), traitent des nouveaux développements dans les systèmes de bases de données déductives et orientées objet.

Ouvrages et articles de recherche sur les aspects opérationnels de la gestion de données

Dippold et al. (2001) ont publié un véritable livre spécialisé dans les aspects opérationnels de la gestion de données. Dans leur ouvrage sur la gestion du développement, Biethahn et al. (2000) consacrent plusieurs chapitres à l'architecture et à l'administration de données. L'ouvrage d'introduction de Martin (1986) se concentre sur la technologie des bases de données et contient également des réflexions du point de vue du management. Le manuel de l'informatique de gestion de Kurbel et Strunz (1990) dédie un chapitre à la gestion de

données. Les livres de Scheer (1991) et de Vetter (1998) traitent de l'architecture de données et décrivent les méthodes de conception d'un modèle de données d'entreprise. Heinrich (1999), Martiny et Klotz (1989), Österle et al. (1991) survolent dans leurs ouvrages le domaine de la gestion de l'information et les thèmes liés à la gestion de données. Aux livres susmentionnés il convient d'ajouter certains articles dont celui de Gemünden et Schmitt (1991) qui rapporte les résultats d'une recherche empirique sur les grandes entreprises allemandes. L'article de Meier (1994) définit les objectifs et les fonctions de la gestion de données du point de vue du praticien. Les questions relatives à l'administration de données sont analysées dans les articles de Meier et Johner (1991) et d'Ortner et al. (1990).

2 Les phases de la construction d'un modèle de données

2.1 De l'analyse à la base de données

Un *modèle de données* (*data model*, en anglais) est une description formelle et structurée des données et de leurs relations dans un système d'information. La figure 2-1 montre l'exemple d'une entreprise qui, pour gérer ses projets informatiques, doit rassembler des informations sur ses employés, sur ses projets et ses départements. L'entreprise devra construire un modèle de données où seront définies les classes de données requises (catégories de données) et leurs dépendances. À ce stade, les classes de données, ou ensembles d'entités dans notre terminologie spécialisée, et les ensembles de liaisons sont définis indépendamment des ordinateurs et des systèmes de bases de données qui serviront plus tard à la saisie, au stockage et à la mise à jour des informations. *Du point de vue des utilisateurs*, cette indépendance a pour but d'assurer la *stabilité* des données et de leurs relations face au développement des systèmes informatiques et à l'évolution des logiciels.

De la description d'une portion du monde réel jusqu'à l'élaboration d'une base de données proprement dite, la méthode de travail comporte trois phases majeures : l'analyse des données, la construction d'un modèle entité-association et sa conversion en un schéma de base de données relationnelle.

La première phase, l'analyse de données, vise à déterminer, en collaboration avec les utilisateurs, les données nécessaires à un système d'information, leurs relations ainsi que la structure des ensembles qui en résultent. C'est ainsi qu'on parvient à délimiter dès le début les frontières d'un système. À travers une démarche itérative, les interviews, l'analyse des besoins, les questionnaires, les formulaires, etc., doivent permettre de produire une documentation

Le modèle de données est une abstraction du monde réel

Les trois phases pour construire un modèle de données

Première phase : analyse des données et de leurs relations

complète. Elle contient obligatoirement une description narrative du mandat avec des objectifs clairement définis, ainsi qu'une *liste des informations factuelles pertinentes* (voir l'exemple dans la figure 2-1). Dans la phase d'analyse, la description narrative des relations entre les données est complétée par leur représentation graphique ou illustrée d'un exemple récapitulatif. Durant cette phase, il est crucial de formuler, dans le langage des utilisateurs, les faits nécessaires au développement ultérieur d'une base de données.

*Deuxième phase :
définition des
ensembles
d'entités et de liens*

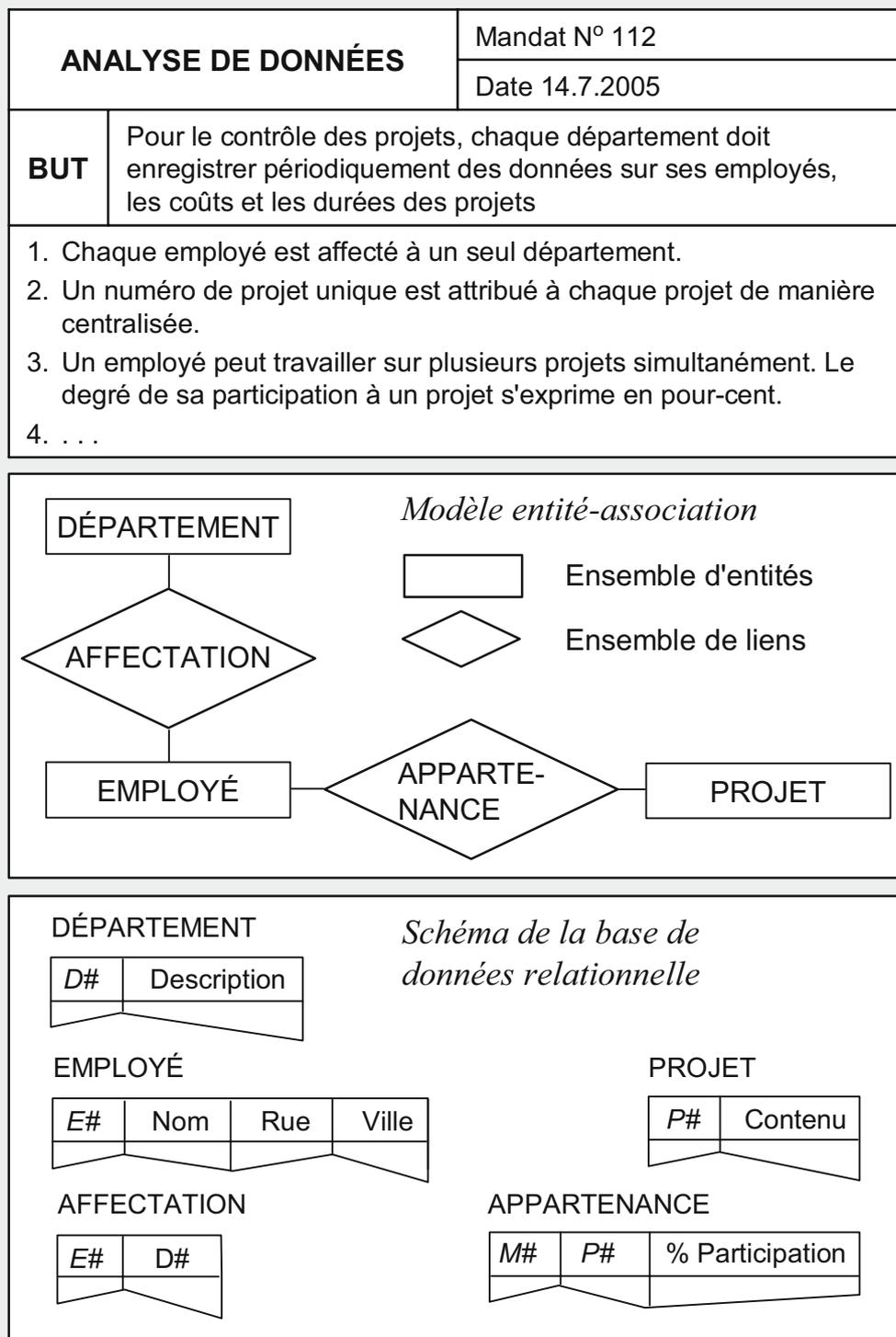
La deuxième phase d'abstraction vise à concevoir un *modèle entité-association* (*entity relationship model*, en anglais) où l'on définit les ensembles d'entités et les ensembles de liens entre ces entités. Dans ce modèle, les ensembles d'entités sont représentés graphiquement par des rectangles, et les ensembles de liens par des losanges. Partant de l'analyse des données présentée dans la figure 2 1 on découvre trois principaux ensembles d'entités : DÉPARTEMENT, EMPLOYÉ et PROJET¹. Pour savoir dans quel département et sur quel projet travaille chaque employé, on définit deux ensembles de liens : AFFECTATION et APPARTENANCE, que l'on connecte aux trois ensembles d'entités dans le diagramme. Ainsi, un modèle entité-association permet de structurer et de visualiser avec clarté les faits recueillis dans la phase d'analyse des données. Il convient de souligner ici qu'il n'est pas toujours facile d'identifier des ensembles d'entités et de liens ainsi que leurs attributs de manière unique. Bien au contraire, la phase de conception exige de la compétence et de l'expérience pratique de la part de l'architecte de données.

*Troisième phase :
conception du
schéma de la base
de données
relationnelle*

La troisième phase a pour but de convertir le modèle entité-association en un *schéma de base de données relationnelle* (*relational database schema*, en anglais). Définir un schéma de base de données, c'est fournir une description formelle des objets dans la base de données considérée. Sachant qu'une base de données relationnelle n'admet que des tables comme objets, on doit donc exprimer tous *les ensembles d'entités et de liens sous forme de tables*.

¹ Comme pour les noms de tables, nous écrivons en majuscules les noms des ensembles d'entités et des ensembles de liens.

Figure 2-1
Les principales
phases de la
modélisation des
données



Dans la figure 2-1, une table est créée pour chacun des trois ensembles d'entités, à savoir DÉPARTEMENT, EMPLOYÉ et PROJET. Pour représenter sous forme tabulaire les liens entre les ensembles d'entités, on définit pour chaque ensemble de liens une table distincte

qui contient obligatoirement les clés des ensembles d'entités participant à la liaison considérée, appelées clés étrangères, et éventuellement d'autres attributs qui caractérisent cette liaison. Dans notre exemple, la table AFFECTATION contient deux clés étrangères, le numéro de l'employé et celui du département ; la table APPARTENANCE contient deux clés étrangères, le numéro de l'employé et celui du projet, et un attribut supplémentaire, le degré de participation au projet, exprimé en pour-cent.

L'analyse et la conception se réalisent indépendamment des considérations techniques

Nous venons d'exposer de manière sommaire la conduite d'une analyse des données, le développement d'un modèle entité-association et la définition d'un schéma de base de données relationnelle. Le but principal de l'exposé est de souligner l'importance de concevoir une base de données à partir d'un modèle entité-association. Cette approche permet, dans un premier temps, d'identifier et d'étudier avec les utilisateurs les aspects propres à la modélisation des données indépendamment de tout système de bases de données particulier. C'est dans une phase de conception ultérieure qu'on définira ensuite un schéma de base de données en appliquant, dans le cas d'un modèle relationnel, des règles de conversion clairement établies (voir section 2.3).

2.2 Le modèle entité-association

2.2.1 Entités et associations

Les entités sont des objets pouvant être identifiés distinctement

Une *entité* (*entity*, en anglais) est un objet spécifique (c'est-à-dire qui peut être identifié distinctement parmi d'autres objets) dans le monde réel ou dans notre pensée. Elle peut désigner une personne, un objet, un concept abstrait ou un événement. Les entités de même type forment un *ensemble d'entités* caractérisées par un certain nombre d'attributs. Par exemple, la taille, la désignation et le poids sont les propriétés d'une certaine entité et de l'ensemble d'entités auquel elle appartient.

Le rôle de la clé d'identification

Pour chaque ensemble d'entités, nous définissons une clé d'identification, formée d'un attribut ou d'une combinaison d'attributs, qui permet de distinguer chaque entité de manière unique

dans l'ensemble considéré. À cette contrainte d'unicité, s'ajoute la contrainte de combinaison minimale des attributs, expliquée dans la section 1.1 au sujet de la clé d'identification d'une table.

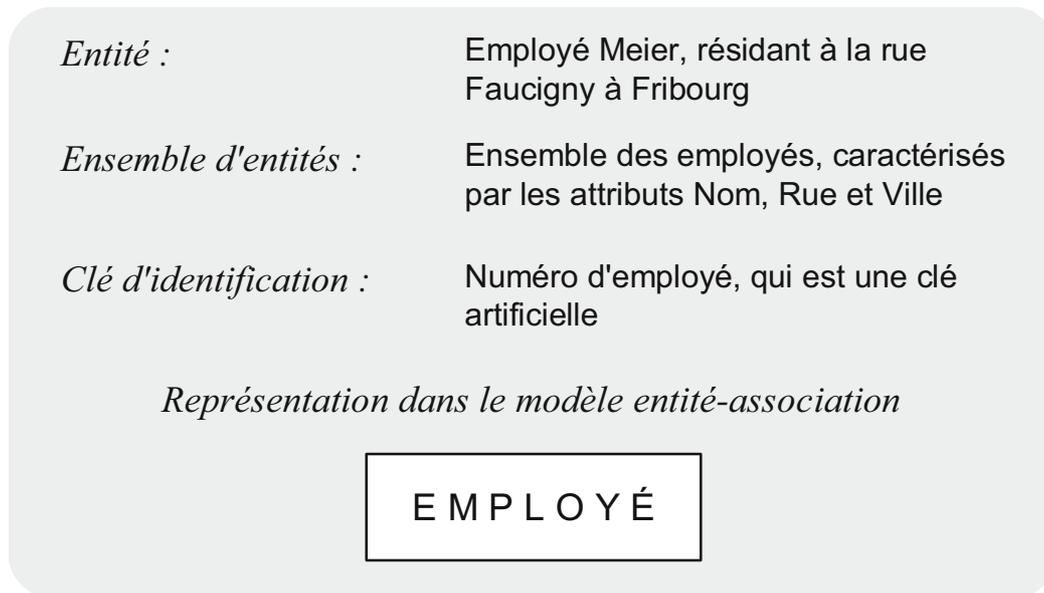


Figure 2-2

Exemple :

l'ensemble

d'entités EMPLOYÉ

Dans la figure 2-2, un employé particulier représente une entité avec ses valeurs d'attributs réelles. Pour assurer la gestion interne des projets, l'entreprise doit rassembler les noms et les adresses de tous ses employés en créant un ensemble d'entités appelé EMPLOYÉ. En plus des trois attributs, Nom, Rue et Ville, un attribut artificiel, le Numéro d'employé, permet d'identifier de manière unique chacun des employés (entités) au sein du personnel (ensemble d'entités) de l'entreprise.

Création d'un

ensemble d'entités
pour les employés

Il est important de découvrir les *liaisons* (*relationships*, en anglais) existant entre les ensembles d'entités. Elles forment également un ensemble. Les ensembles de liaisons ont chacun leurs propres attributs.

Définition des

ensembles de
liaisons

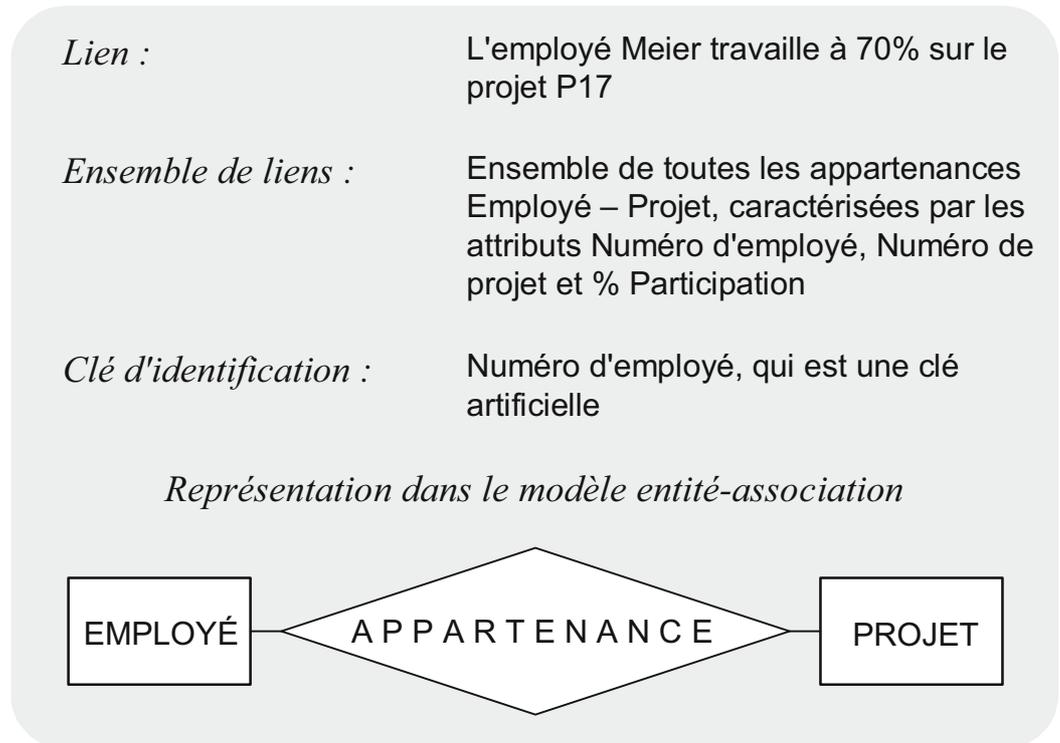
Dans la figure 2-3, l'énoncé «L'employé Meier travaille à 70% sur le projet P17» concrétise une liaison Employé - Projet. L'ensemble de liaisons APPARTENANCE contient toutes les participations des employés aux projets de l'entreprise. Il possède une clé composée de deux clés étrangères, le Numéro d'employé et le Numéro de projet. La concaténation de ces deux attributs permet d'identifier de manière unique chaque appartenance d'un employé à un projet. Outre sa clé

Chaque ensemble

de liaisons a ses
propres attributs

d'identification, l'ensemble de liaisons se caractérise encore par l'attribut «% Participation» qui indique le pourcentage du temps de travail qu'un employé consacre à un projet.

Figure 2-3
La liaison
APPARTENANCE
entre les employés
et les projets



Les liaisons
impliquent deux
associations

En général, les liaisons définissent des associations dans les deux directions. Du point de vue de l'ensemble d'entités EMPLOYÉ, nous interprétons l'ensemble de liaisons APPARTENANCE comme traduisant le fait qu'un employé peut participer à plusieurs projets. Du point de vue de l'ensemble d'entités PROJET, l'ensemble de liaisons nous fait comprendre qu'un projet est réalisé par plusieurs employés.

2.2.2 Les types d'associations

Une liaison
s'exprime à travers
les associations

Une *association* (*association*, en anglais) d'un ensemble d'entités EE_1 à un deuxième ensemble EE_2 définit un lien orienté dans cette direction. Considérons par exemple la liaison CHEF DE DÉPARTEMENT dans la figure 2-4. Nous identifions deux associations dans cet ensemble de liaisons. D'une part, chaque département occupe un employé au poste de directeur ; d'autre part, un certain nombre d'employés exercent la fonction de directeur d'un département.

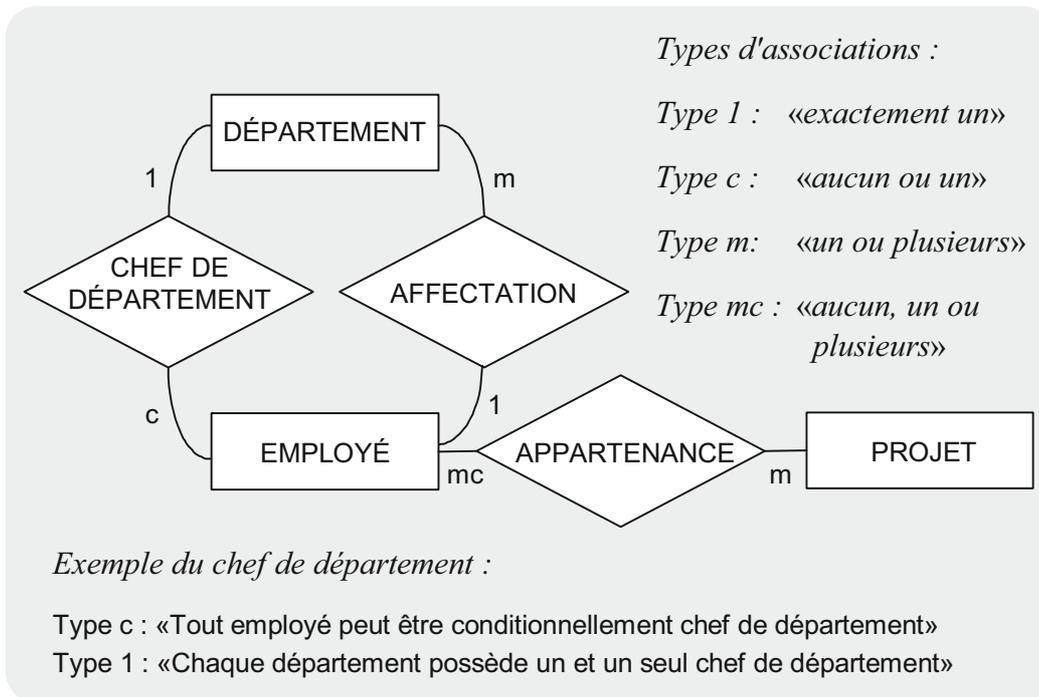


Figure 2-4
Un modèle entité-association avec les types d'associations

Chaque association d'un ensemble d'entités EE_1 à un autre ensemble EE_2 est caractérisée par un type. Le type d'association de EE_1 à EE_2 indique le nombre d'entités provenant de l'ensemble associé EE_2 qui sont reliées à une entité dans EE_1. On distingue essentiellement quatre types d'associations : simple, conditionnelle, multiple et multiple conditionnelle.

Association simple (Type 1)

Dans une association simple (type 1), à chaque entité dans l'ensemble d'entités EE_1 correspond «une et une seule» entité dans l'ensemble EE_2. Par exemple, selon notre analyse des données, chaque employé est affecté à un seul département ; nous ne sommes donc pas en présence d'une «organisation matricielle» de l'entreprise. Par conséquent, dans la figure 2-4, l'association AFFECTATION des employés aux départements est simple (type 1).

Association simple de type 1

Association conditionnelle (Type c)

À chaque entité dans l'ensemble d'entités EE_1 correspond «zéro ou une entité», c'est-à-dire au plus une entité dans l'ensemble EE_2. Cette association est de type *conditionnel* (*conditional*, en anglais). Par exemple, dans la liaison CHEF DE DÉPARTEMENT (voir figure 2-4) existe une association conditionnelle des employés aux départements,

Association conditionnelle de type c

car chaque employé n'exerce pas forcément la fonction de chef de département.

Association multiple (Type m)

Association complexe de type m

Dans une association multiple (type m), à chaque entité dans l'ensemble d'entités EE_1 correspondent «une ou plusieurs» entités dans l'ensemble EE_2. Ce type d'association et celui défini dans le prochain paragraphe sont dits *complexes*, car une entité dans EE_1 peut être reliée à un nombre quelconque d'entités dans EE_2. Dans la figure 2-4, le lien d'APPARTENANCE des projets aux employés est l'exemple d'une association multiple : chaque projet est réalisé par au moins un ou plusieurs employés.

Association multiple conditionnelle (Type mc)

Association conditionnelle complexe de type mc

À chaque entité dans l'ensemble d'entités EE_1 correspondent «aucune, une ou plusieurs» entités dans l'ensemble EE_2. Le *type d'association multiple conditionnelle* se distingue du type d'association multiple par le fait que chaque entité dans EE_1 n'est pas forcément reliée aux entités dans EE_2. Par exemple, dans la figure 2-4, en considérant la liaison APPARTENANCE du point de vue de l'employé, on constate d'une part que chaque employé ne participe pas obligatoirement à un projet, et d'autre part, qu'un employé peut contribuer à plusieurs projets à la fois.

Degré d'une liaison

Les types d'associations déterminent le degré d'une liaison. Comme nous l'avons vu précédemment, chaque liaison comprend deux types d'associations. Dès lors, le *degré d'une liaison* entre les ensembles d'entités EE_1 et EE_2 s'exprime par la *paire de types d'associations* qui la composent sous la forme suivante :

$$\text{Degré} := (\text{Type d'association de EE}_1 \text{ à EE}_2, \text{ Type d'association de EE}_2 \text{ à EE}_1)^2.$$

² La notation «:=» se lit «est défini par ...».

Par exemple, la paire de types d'associations entre EMPLOYÉ et PROJET, notée (mc,m), signifie que le lien APPARTENANCE est «complexe-complexe».

Au lieu de déclarer les types d'associations, nous pouvons indiquer les *limites minimale et maximale* si cela s'avère approprié. À titre d'exemple, au lieu du type d'association multiple des projets aux employés (type m), nous pouvons spécifier (MIN,MAX) := (3,8). La limite inférieure exprime le fait qu'au moins trois employés participent à chaque projet. Inversement, la limite supérieure impose un effectif maximal de huit employés par projet.

Limites minimale et maximale d'une liaison

2.2.3 Généralisation et agrégation

La *généralisation* (*generalization*, en anglais) des ensembles d'entités est un processus d'abstraction qui consiste à généraliser les entités, et donc les ensembles d'entités en un seul ensemble ascendant. Réciproquement, la *spécialisation* consiste à définir des ensembles d'entités descendants ou sous-ensembles d'entités dans une hiérarchie de généralisation. La généralisation des ensembles d'entités donne lieu à plusieurs cas possibles :

On peut généraliser, spécialiser les entités

- Les sous-ensembles d'entités définis par spécialisation *présentent des intersections*. Considérons par exemple l'ensemble d'entités EMPLOYÉ et ses deux sous-ensembles, CLUB DE PHOTO et CLUB DE SPORT. Chaque membre d'un club est donc en même temps un employé. Réciproquement, un employé peut adhérer au club interne de l'entreprise pour la photo tout en étant membre actif du club de sport. En d'autres termes, les sous-ensembles CLUB DE PHOTO et CLUB DE SPORT ont une intersection non vide.
- Les sous-ensembles d'entités définis par spécialisation *ont des intersections, et contiennent tous les éléments* de l'ensemble d'entités ascendant qui les généralise. Nous désignons ce cas par l'expression «avec intersection et complet». Ajoutons par exemple un troisième sous-ensemble d'entités CLUB D'ÉCHEC, et admettons qu'à l'embauche chaque employé adhère au moins à l'un des trois clubs, CLUB DE PHOTO, CLUB DE SPORT ou CLUB

Ensembles d'entités avec intersection

Ensembles d'entités avec intersection et complets

D'ÉCHEC. Par conséquent, ces trois sous-ensembles contiennent tous les éléments de l'ensemble EMPLOYÉ d'une part, et présentent des intersections non vides d'autre part. En effet, un employé qui est membre d'au moins un club peut adhérer à deux ou trois clubs simultanément.

*Ensembles
d'entités disjoints*

- Les sous-ensembles d'entités définis par spécialisation sont *mutuellement disjoints*, c'est-à-dire que leur intersection est vide. Considérons par exemple l'ensemble d'entités EMPLOYÉ et ses deux sous-ensembles spécialisés, CADRE SUPÉRIEUR et SPÉCIALISTE. Puisqu'un employé ne peut pas occuper un poste de cadre supérieur et travailler en même temps comme spécialiste, CADRE SUPÉRIEUR et SPÉCIALISTE sont donc des sous-ensembles spécialisés disjoints.

*Ensembles
d'entités disjoints
et complets*

- Les sous-ensembles d'entités définis par spécialisation sont *mutuellement disjoints*, et contiennent tous les éléments de l'ensemble d'entités ascendant qui les généralise. Nous identifions ce cas par l'expression «disjoint et complet». Pour chaque entité dans l'ensemble d'entités ascendant, il doit exister une sous-entité définie par spécialisation, et vice versa. À titre d'exemple, reprenons l'ensemble d'entités EMPLOYÉ et ses deux sous-ensembles spécialisés, CADRE SUPÉRIEUR et SPÉCIALISTE, et ajoutons un troisième sous-ensemble d'entités APPRENTI. Nous constatons maintenant que chaque employé travaille soit comme cadre supérieur, soit comme spécialiste, soit comme apprenti.

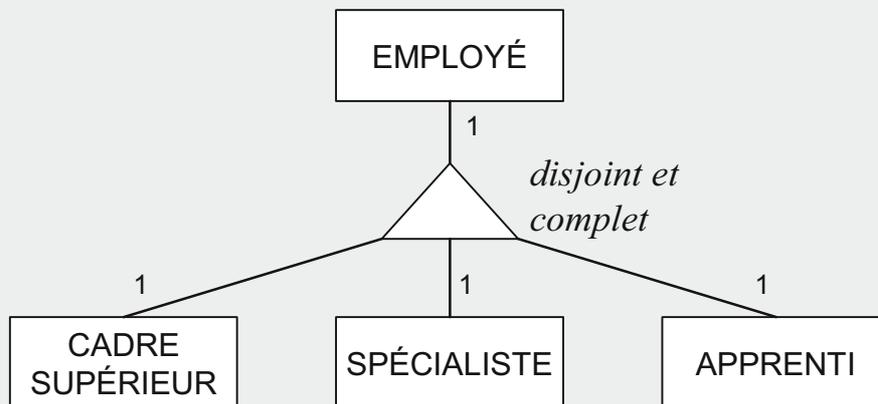
*Représentation
graphique de la
hiérarchie de
généralisation*

Pour représenter graphiquement une hiérarchie de généralisation, nous adoptons le symbole de la fourche, accompagnée de la mention «avec intersection et incomplet», «avec intersection et complet», «disjoint et incomplet» ou «disjoint et complet».

*La généralisation
définit une
structure EST-UN*

La figure 2-5 illustre le cas «disjoint et complet» où l'ensemble d'entités EMPLOYÉ généralise CADRE SUPÉRIEUR, SPÉCIALISTE et APPRENTI. Chaque entité dépendante, telle que le chef de groupe ou le chef de département dans l'ensemble d'entités CADRE SUPÉRIEUR, est de type EMPLOYÉ. C'est une association de type 1. C'est pourquoi, on dit aussi que la généralisation est une structure EST-UN (IS A *structure*,

en anglais) : un chef de groupe «est un» (*is a*, en anglais) employé, un chef de département est également un employé. L'association dans la direction inverse est aussi de type 1 dans le cas «disjoint et complet» d'une hiérarchie de généralisation ; par exemple chaque employé appartient à un et un seul sous-ensemble d'entités.



Exemple de catégorisation des employés :

«Tout employé est soit un cadre supérieur, soit un spécialiste, soit un apprenti et vice versa»

Figure 2-5
EMPLOYÉ, un
exemple de
généralisation

Comme la généralisation, une deuxième structure de liens joue également un rôle important. Elle repose sur le concept d'*agrégation* (*aggregation*, en anglais) qui consiste à réunir les entités en un tout de niveau supérieur. Les propriétés de la structure d'agrégation sont définies dans un ensemble de liens qui en résulte.

Création d'une
structure
d'agrégation

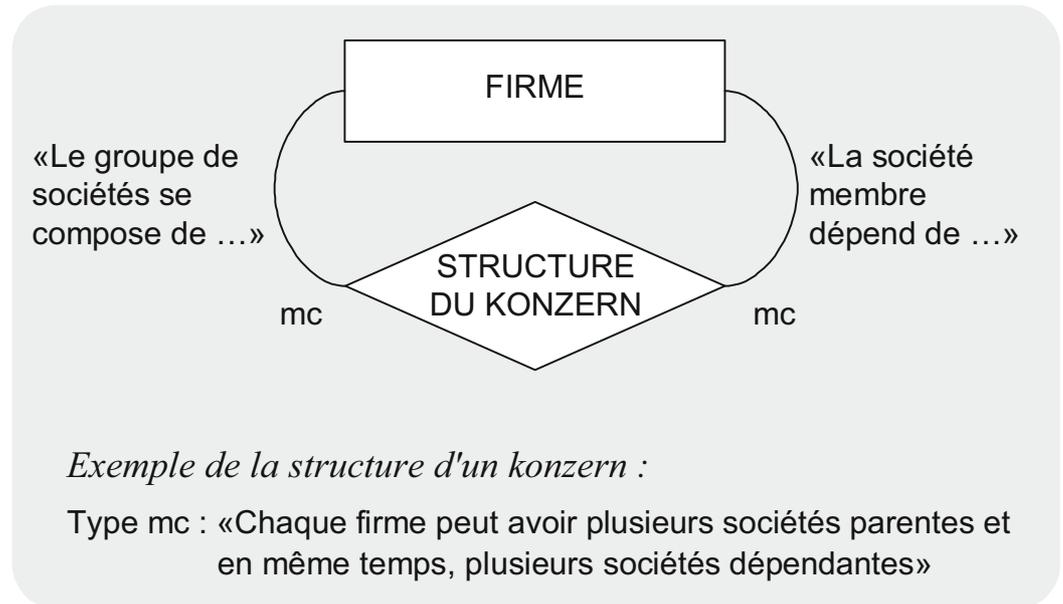
À titre d'exemple, considérons la structure de participation des sociétés dans un konzern³. Pour la modéliser, nous définissons un ensemble de liaisons, nommé STRUCTURE DU KONZERN, qui relie l'ensemble d'entités FIRME à lui-même, comme le montre la figure 2-6. Le numéro de société dans l'ensemble d'entités FIRME sera utilisé deux fois comme clés étrangères dans la STRUCTURE DU KONZERN, pour désigner d'une part la société mère hiérarchiquement supérieure,

Le konzern, un
exemple de
structure
d'agrégation

³ Le konzern désigne un groupe d'entreprises juridiquement indépendantes, liées par des participations financières croisées, l'entreprise F1 possédant une partie du capital en actions de l'entreprise F2, et inversement.

d'autre part la filiale hiérarchiquement inférieure (voir aussi la figure 2-13). D'autres attributs tels que le taux de participation d'une firme viennent compléter les propriétés de la STRUCTURE DU KONZERN.

Figure 2-6
Agrégation en
réseau dans la
STRUCTURE DU
KONZERN



L'agrégation définit
une structure
MEMBRE-DE

L'agrégation permet un regroupement des entités dans une structure dite MEMBRE-DE (PART OF *structure*, en anglais). Ainsi, dans la STRUCTURE DU KONZERN, chaque firme est «membre d'un» (*part of*, en anglais) Konzern particulier. Puisque la STRUCTURE DU KONZERN est définie comme une structure en réseau dans l'exemple, les deux associations reliant les membres supérieurs et inférieurs sont donc de type complexe.

La généralisation
et l'agrégation sont
deux importants
concepts de
structuration

Les deux concepts d'abstraction, la généralisation et l'agrégation, permettent de définir deux structures⁴ importantes dans un modèle de données. Elles sont représentées dans un modèle entité-association soit par des symboles graphiques particuliers, soit par des boîtes spéciales. Ainsi, l'agrégation représentée dans la figure 2-6 peut aussi s'exprimer comme un ensemble d'entités généralisé KONZERN qui inclut implicitement l'ensemble d'entités FIRME et l'ensemble de liens STRUCTURE DU KONZERN.

⁴ Les systèmes de bases de données étendus supportent les concepts de généralisation et d'agrégation pour structurer les données (voir la section 6.4 qui traite des systèmes de bases de données relationnelles-objet).

À la structure en réseau, ajoutons une deuxième variante qui est la structure MEMBRE-DE hiérarchique, illustrée par l'ensemble de liens NOMENCLATURE dans la figure 2-7 : chaque article peut être assemblé à partir de plusieurs composants. Réciproquement, chaque composant, à l'exception de l'article au sommet de la hiérarchie, fait partie d'un article de niveau supérieur. (voir figure 2-15).

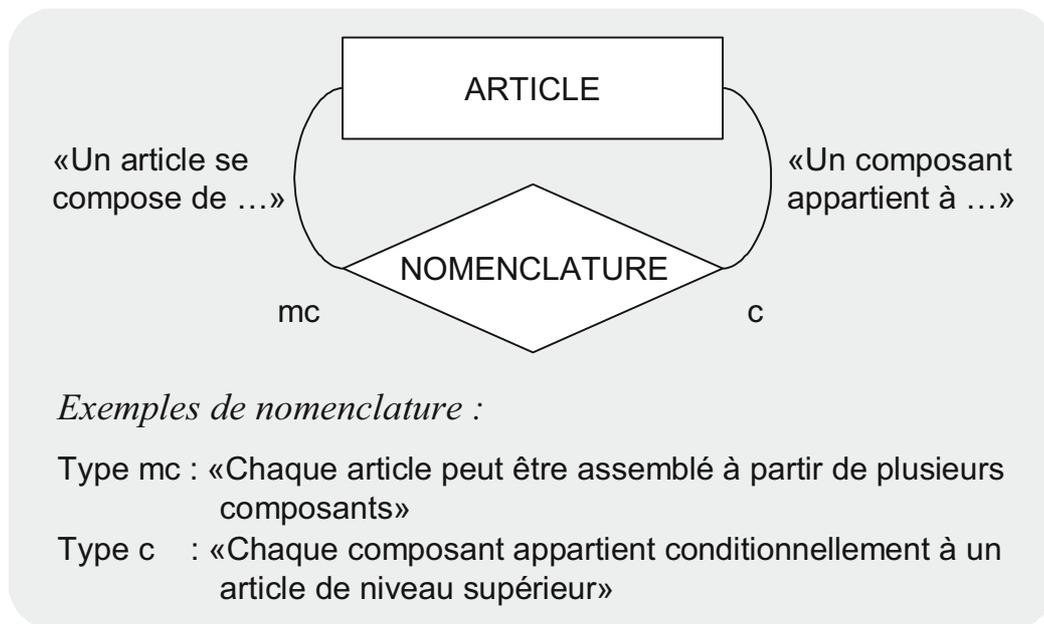


Figure 2-7
Un exemple
d'agrégation
hiérarchique :
NOMENCLATURE

Le modèle entité-association occupe une place importante dans le développement d'outils pour la modélisation de données assistée par ordinateur. Il est intégré à des degrés différents dans les outils CASE (CASE = Computer Aided Software Engineering). Les premières étapes de développement, assisté par ordinateur, consistent à définir des ensembles d'entités et de liens, et à structurer les données par la généralisation et l'agrégation. Ensuite c'est le *passage partiellement automatique du modèle entité-association à un schéma de base de données*. Ce processus de transformation conduit généralement à plusieurs variantes possibles. C'est pourquoi il incombe aux architectes de données de prendre les décisions appropriées. Dans cette optique, les sections suivantes définissent un ensemble de règles simples qui permettent de traduire correctement un modèle entité-association en un schéma de base de données relationnelle.

*La modélisation de
données assistée
par ordinateur*

2.3 Le schéma d'une base de données relationnelle

2.3.1 Le passage du modèle entité-association au schéma de base de données relationnelle

Nous abordons maintenant la traduction d'un modèle entité-association en un schéma de base de données relationnelle. Il s'agit essentiellement de définir une méthode qui permet de représenter les *ensembles d'entités et de liens* par des tables.

Que signifie un schéma de base de données ?

On appelle *schéma de base de données* (*database schema*, en anglais) la description d'une base de données, obtenue en spécifiant les structures de données dont elle est constituée et de leurs contraintes d'intégrité. Un schéma de base de données relationnelle contient donc la définition des tables, des attributs et des clés primaires. Les contraintes d'intégrité imposent des limites aux domaines des attributs, établissent des dépendances entre les tables (nous parlerons d'intégrité référentielle à la section 2.5) et définissent des restrictions sur les données proprement dites.

Les premières règles de passage 1 et 2 sont capitales pour transformer un modèle entité-association en un schéma de base de données relationnelle (voir figure 2-8).

Règle 1 (ensemble d'entités)

Règle impérative pour les ensembles d'entités

Chaque ensemble d'entités *doit* être traduit en une table distincte, dotée d'une clé primaire qui peut être soit la clé correspondante de l'ensemble d'entités, soit une clé candidate. Les autres attributs de l'ensemble d'entités complètent les attributs de la table.

Choix d'une clé candidate comme clé primaire

La définition d'une table (section 1.1) requiert une clé primaire unique. Lorsqu'il existe dans une table plusieurs *clés candidates* (*candidate keys*, en anglais) qui satisfont toutes aux critères d'unicité et de minimalité, c'est l'architecte de données qui décide du choix de l'une d'elles comme clé primaire.

Règle 2 (ensembles de liens)

Chaque ensemble de liens *peut* être traduit en une table distincte. Les clés d'identification des ensembles d'entités participantes doivent y figurer comme *clés étrangères*. La clé primaire de cette table peut être soit une clé d'identification formée par la concaténation des clés étrangères, soit une autre clé candidate en créant par exemple une clé artificielle. Les autres attributs de l'ensemble de liens complètent les attributs de la table.

Règle de passage possible pour les ensembles de liens

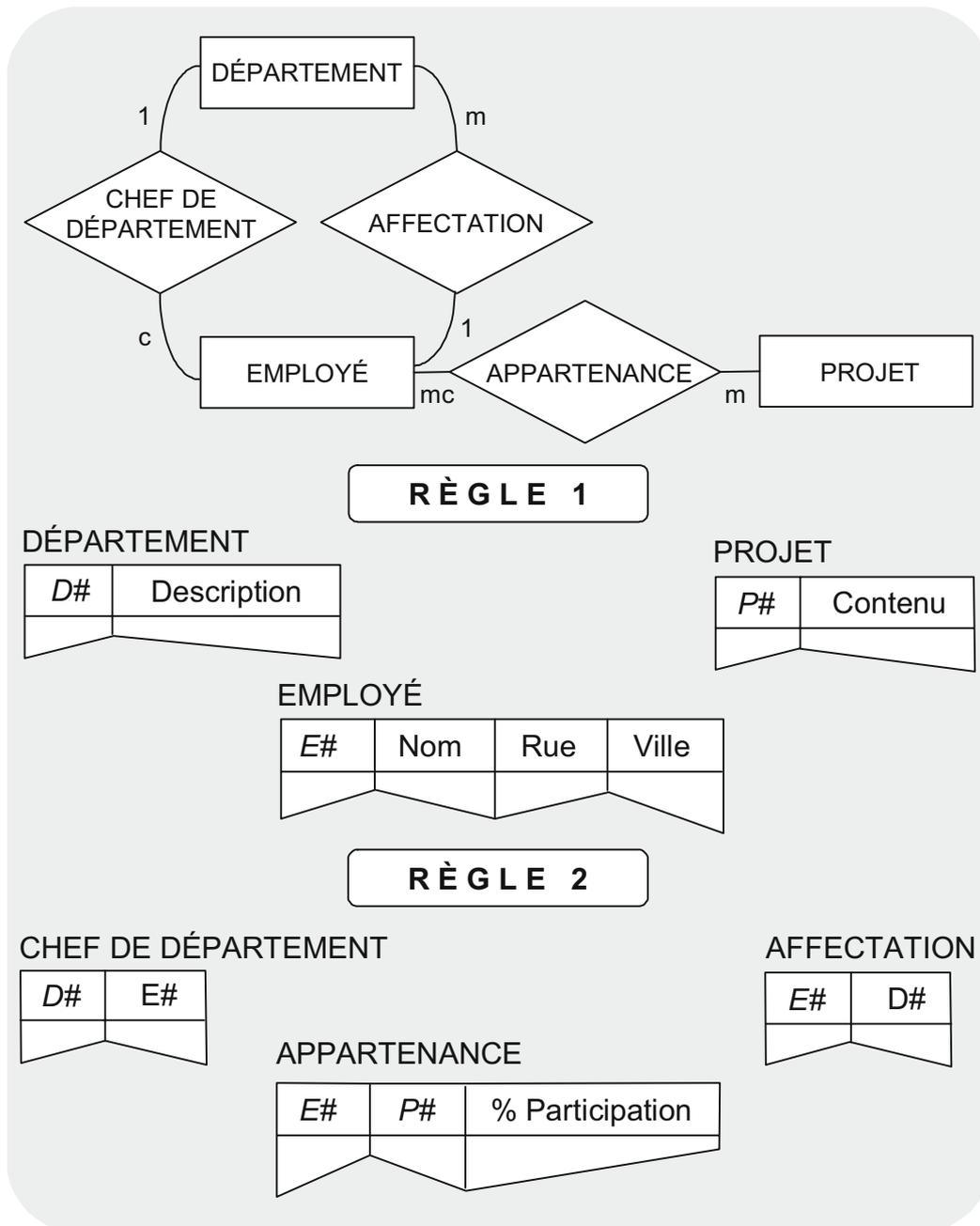


Figure 2-8
Conversion des ensembles d'entités et de liaisons en tables

Les clés étrangères traduisent les liens entre les tables

La *clé étrangère* (*foreign key*, en anglais) dans une table est formée à partir d'un attribut ou d'une concaténation d'attributs qui sert de clé d'identification de la même table ou d'une table différente. En d'autres termes, la clé d'identification d'une table doit donc figurer dans toutes les tables auxquelles elle doit être reliée.

La figure 2-8 illustre l'application des règles 1 et 2 dans un exemple concret. Les ensembles d'entités, DÉPARTEMENT, EMPLOYÉ et PROJET, sont traduits en trois tables portant les mêmes noms respectifs. Nous créons ensuite une table pour chacune des entités de liens, CHEF DE DÉPARTEMENT, AFFECTATION et APPARTENANCE. Dans les tables CHEF DE DÉPARTEMENT et AFFECTATION, les clés étrangères sont le numéro de département et le numéro d'employé. La table APPARTENANCE utilise les clés d'identification des tables EMPLOYÉ et PROJET comme clés étrangères, et contient un attribut supplémentaire appelé «% Participation».

La clé d'identification des tables correspondant aux ensembles de liens

Étant donné que chaque département est dirigé par un seul chef, le numéro de département D# suffit pour former la clé d'identification de la table CHEF DE DÉPARTEMENT. De même, le numéro d'employé E# suffit pour définir la clé d'identification de la table AFFECTATION, car chaque employé est affecté à un seul département.

Contrairement aux tables CHEF DE DÉPARTEMENT et AFFECTATION, la clé d'identification de la table APPARTENANCE doit être formée par la concaténation de deux clés étrangères : le numéro d'employé et le numéro du projet. La raison est qu'un employé peut participer à plusieurs projets, et qu'inversement, un projet peut impliquer plusieurs employés.

Quand faut-il définir une table distincte pour un ensemble de liens ?

L'application des règles de passage 1 et 2 ne conduit pas toujours à un schéma de base de données relationnelle optimal. Selon les circonstances, elle pourrait engendrer un grand nombre de tables. En se référant à la figure 2-8, on peut se demander par exemple s'il était vraiment nécessaire de créer une table distincte pour la fonction de chef de département. Dans la prochaine section, nous verrons qu'en vertu de la règle de passage 5, nous renoncerons en fait à créer la table CHEF DE DÉPARTEMENT. La fonction «Chef de département» sera intégrée dans la table DÉPARTEMENT tout simplement comme un

attribut supplémentaire dont la valeur est le numéro d'employé du chef pour chaque département.

2.3.2 Règles de passage pour les ensembles de liens

Dans cette section, nous étudions les différentes approches pour transformer les ensembles de liens en tables. Nous énoncerons trois nouvelles règles de passage selon les types de liens.

D'après la définition à la section 2.2.2, le degré d'une liaison est déterminé par une paire de types d'associations. Nous parlons de *liaison simple-simple* lorsqu'elle est constituée de deux types d'associations simples ou conditionnelles. La figure 2-9 nous présente les quatre possibilités d'une liaison simple-simple, caractérisées par les degrés de liaison (1,1), (1,c), (c,1) et (c,c).

Liaisons de type simple-simple

*Degré de la liaison avec les types d'associations A1 et A2
Lj := (A1,A2)*

A1 \ A2	1	c	m	mc
1	(1,1)	(1,c)	(1,m)	(1,mc)
c	(c,1)	(c,c)	(c,m)	(c,mc)
m	(m,1)	(m,c)	(m,m)	(m,mc)
mc	(mc,1)	(mc,c)	(mc,m)	(mc,mc)

L1 Liaisons de type simple-simple

L2 Liaisons de type simple-complexe

L3 Liaisons de type complexe-complexe

Par définition, une *liaison* est *simple-complexe* lorsque son degré consiste en un type d'associations simple (type 1 ou c) et un type d'associations complexe (type m ou mc). Cette définition donne lieu à huit possibilités de liaison simple-complexe.

*Figure 2-9
Vue d'ensemble
des degrés de
liaisons*

Liaisons de type simple-complexe

Liaisons de type complexe-complexe

Une *liaison* est dite *complexe-complexe* lorsque son degré est constitué de deux types d'associations complexes. Les cas possibles sont (m,m), (m,mc), (mc,m) et (mc,mc).

Une règle de passage pour chaque type de liaison

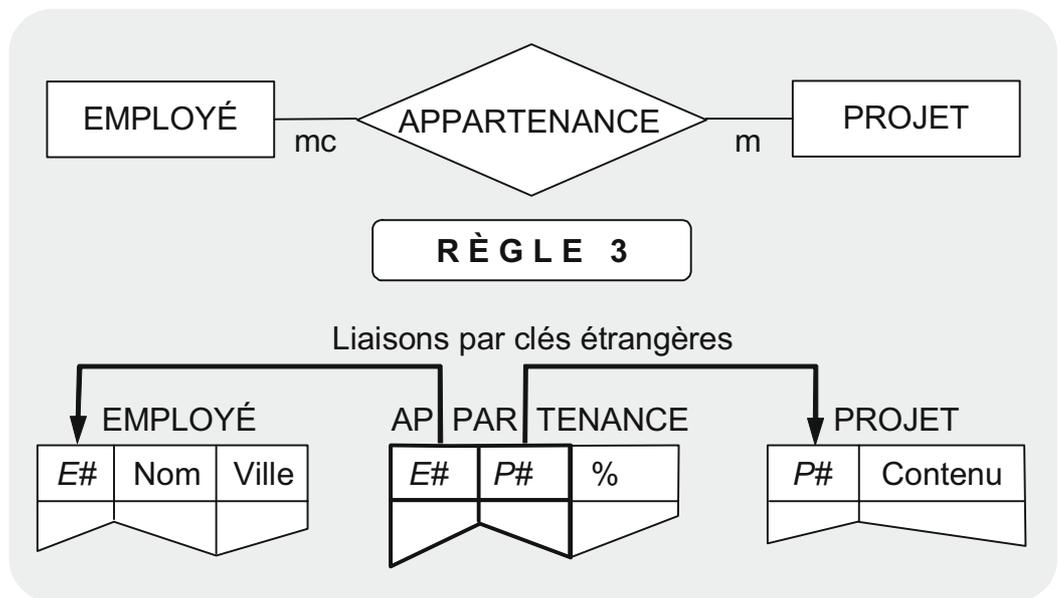
Nous formulons maintenant trois règles qui, fondées sur le degré des liaisons, régissent le passage des ensembles de liens dans un modèle entité-association aux tables dans un schéma de base de données relationnelle. Pour éviter d'accroître inutilement le nombre de tables, l'application de la règle 3 nous amène à considérer tout d'abord les ensembles de liens qu'il faut absolument transformer en tables :

Règle 3 (Liaisons de type complexe-complexe)

Règle impérative pour un ensemble de liens complexe-complexe

Chaque ensemble de liens complexe-complexe doit être transformé en une table distincte. Elle contient au moins, comme clés étrangères, les clés d'identification des ensembles d'entités qui participent à la liaison. La clé primaire de la table est soit la clé d'identification formée par la concaténation des clés étrangères, soit une autre clé candidate. D'autres attributs de l'ensemble de liens deviennent les attributs de la table.

Figure 2-10
Règle de transformation des liaisons de type complexe-complexe



Définition de la clé d'identification par concaténation des attributs

La figure 2-10 illustre l'application de la règle 3 qui transforme l'ensemble de liens APPARTENANCE en une table distincte, pourvue d'une clé primaire. La clé d'identification de la table APPARTENANCE est formée par la combinaison des clés étrangères qui la relie aux

tables EMPLOYÉ et PROJET. L'attribut «% Participation» indique le degré d'appartenance à un projet en pour-cent.

Dans la section précédente, nous avons suivi la règle 2 en créant une table pour l'ensemble de liens AFFECTATION, avec le numéro de département et le numéro d'employé comme clés étrangères. Cette transformation s'avère appropriée au cas où l'entreprise mettra en place une organisation matricielle, abandonnant ainsi le mode d'affectation d'un employé à un seul département, c'est-à-dire l'association de type 1 ; dans ce cas, la liaison entre DÉPARTEMENT et EMPLOYÉ est complexe-complexe. En revanche, si nous avons la certitude que l'organisation ne sera pas matricielle, nous appliquons la règle de passage 4 qui porte sur les liaisons de type simple-complexe.

Règle 4 (liaisons de type simple-complexe)

Un ensemble de liens simple-complexe peut s'exprimer dans les deux tables des ensembles d'entités participantes sans avoir besoin de créer une table distincte. Pour cela, dans la table qui participe à l'association simple (association de type 1 ou c), on définit une clé étrangère qui la relie à la table référencée et l'on ajoute éventuellement d'autres attributs de l'ensemble de liens considéré.

Ainsi, au lieu de transformer une liaison en une table distincte, la règle 4 nous permet de l'exprimer dans une table existante en ajoutant à celle-ci une clé étrangère qui fait référence à l'autre table dans l'association. Pour mettre en évidence une liaison, il convient de suffixer la clé d'identification correspondante par un *nom de rôle* qui permet de comprendre le rôle d'une clé spécifique dans une table étrangère.

La règle 4 est illustrée par la figure 2-11 : au lieu de créer une nouvelle table AFFECTATION pour exprimer la liaison entre départements et employés, nous ajoutons à la table EMPLOYÉ une clé étrangère D#_Affectation. Pour chaque employé, la valeur de cette clé indique le numéro du département auquel il est affecté. La clé étrangère qui exprime la liaison porte un nom d'attribut commençant par la clé d'identification D# suivie du nom de rôle «Affectation».

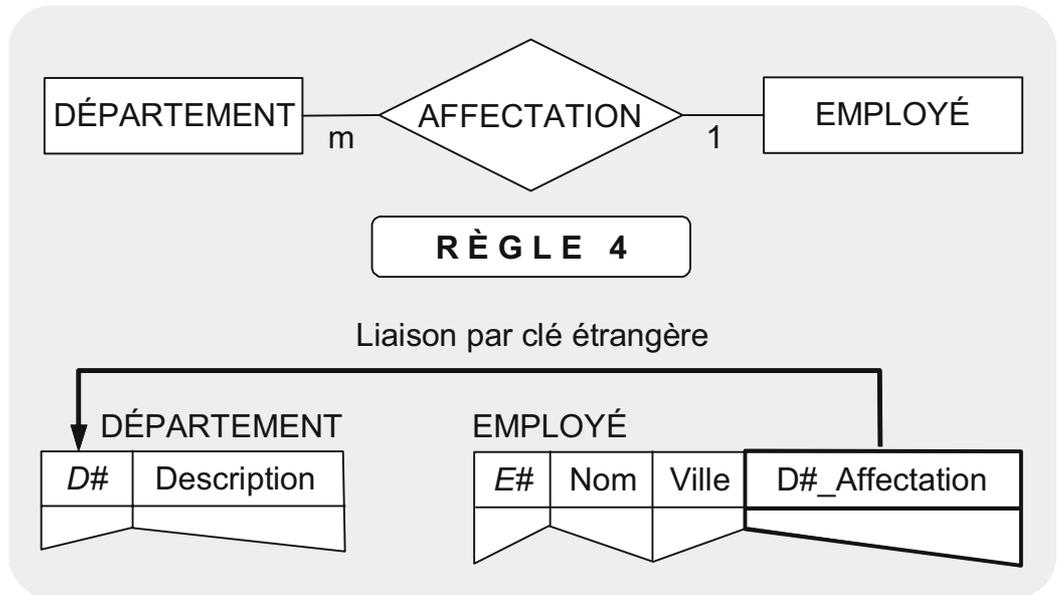
Sommes-nous en présence d'une organisation matricielle ?

Règle de passage possible pour un ensemble de liens simple-complexe

Emploi des noms de rôle

Le nom de la clé étrangère contient un nom de rôle

Figure 2-11
Règle de transformation des liaisons de type simple-complexe



La liaison s'exprime par un attribut supplémentaire dans la table EMPLOYÉ

Le choix de la clé étrangère est évident dans le cas d'un ensemble de liens «simple-complexe». Dans la figure 2-11, en vertu de la règle 4, nous avons choisi le numéro de département comme clé étrangère dans la table EMPLOYÉ. À l'inverse, admettons que le numéro d'employé soit la clé étrangère dans la table DÉPARTEMENT, nous devons alors répéter la description d'un département pour chacun de ses employés. De telles informations superflues ou redondantes sont indésirables. Dans la section suivante, nous aborderons cette question plus en détail dans le cadre de la théorie de la normalisation.

Règle 5 (liaisons de type simple-simple)

Règle de passage possible pour un ensemble de liens simple-simple

Un ensemble de liens simple-simple peut s'exprimer dans les deux tables issues des ensembles d'entités participantes sans avoir besoin de créer une table distincte. L'une des deux tables est la table référencée dont la clé d'identification apparaît comme clé étrangère dans l'autre table.

De nouveau, il est important de désigner correctement la table référencée à laquelle on empruntera une clé candidate pour définir la clé étrangère dans l'autre table. En principe, on introduit la clé étrangère dans la table qui participe à la liaison avec le type d'associations 1.

Les valeurs nulles sont à éviter

Dans la figure 2-12 nous complétons la table DÉPARTEMENT par le numéro d'employé du chef de département. Ainsi, l'ensemble de

liens CHEF DE DÉPARTEMENT s'exprime à travers l'attribut «E#_Chef de département». Chaque valeur de cette clé étrangère identifie un employé dans le rôle de «chef de département». Au lieu de cela, introduisons le numéro de département dans la table EMPLOYÉ. Nous constatons que la valeur de cet attribut est nulle pour la plupart des employés (à ce sujet voir aussi la section 3.6). Un numéro de département non nul existe seulement pour un employé à la tête du département en question. Pour cette raison, nous décidons de créer le rôle de chef de département dans la table DÉPARTEMENT.

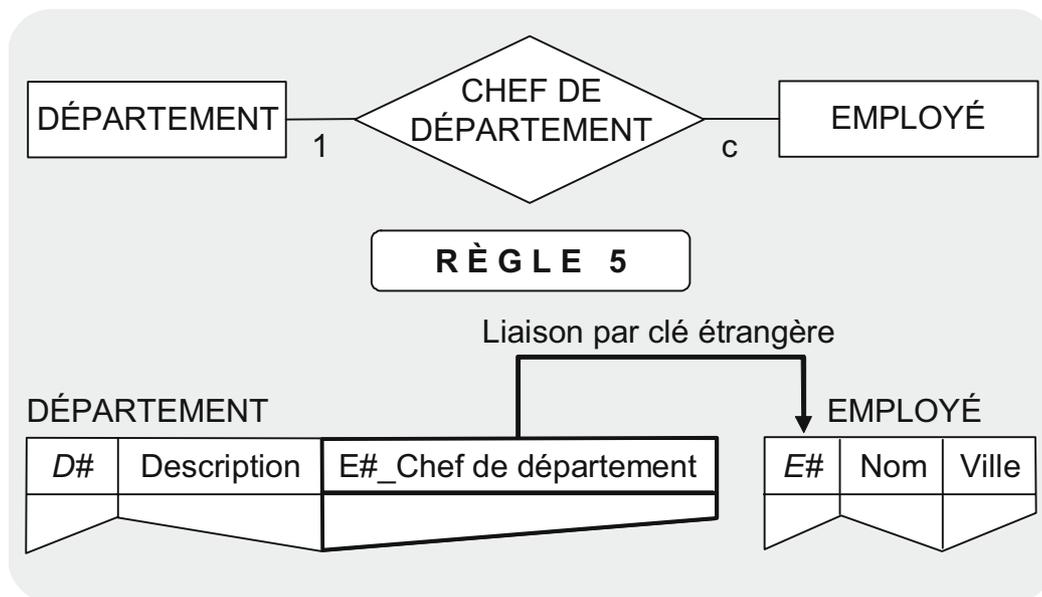


Figure 2-12
Règle de transformation des liaisons de type simple-simple

2.3.3 Règles de passage pour la généralisation et l'agrégation

Nous traitons maintenant du passage d'une hiérarchie de généralisation ou d'une structure d'agrégation dans un modèle entité-association vers un schéma de base de données relationnelle. Ces liaisons particulières contiennent certes les types d'associations définis précédemment, mais leurs règles de passage au schéma relationnel diffèrent de celles déjà énoncées.

Règle 6 (généralisation)

Chaque *ensemble d'entités* dans une hiérarchie de généralisation donne lieu à une *table distincte*. La clé primaire de la table ascendante est aussi celle des tables au niveau inférieur.

Règle de passage d'une structure EST-UN

Les différentes hiérarchies de généralisation

La structure des liens dans une hiérarchie de généralisation n'est pas prise en charge directement par le modèle relationnel. C'est pourquoi nous devons exprimer ses propriétés de manière indirecte. Dans une généralisation de type «avec intersection et incomplet» ou «avec intersection et complet», «disjoint incomplet» ou «disjoint complet», les clés d'identification des tables spécialisées doivent toujours correspondre à celle de la table ascendante. Aucune règle de test n'est nécessaire pour garantir la propriété des ensembles spécialisés avec intersection non vide. L'implémentation des ensembles disjoints dans le modèle relationnel requiert en revanche une attention particulière. Une méthode consiste à introduire dans la table ascendante un nouvel attribut Catégorie qui permet de réaliser une classification : chaque valeur de l'attribut désigne l'ensemble spécialisé représentant une catégorie. En outre, dans une généralisation de type disjoint complet, il faut assurer qu'à chaque occurrence dans la table ascendante correspond une occurrence dans l'une des tables spécialisées, et vice versa.

Exemple d'une généralisation de type disjoint complet

Dans la figure 2-13, les données sur les employés forment une hiérarchie de généralisation. La règle 6 nous amène à créer quatre tables, EMPLOYÉ, CADRE SUPÉRIEUR, SPÉCIALISTE et APPRENTI. Nous devons définir la même clé d'identification E# pour toutes les tables subordonnées à la table EMPLOYÉ. Afin qu'un employé particulier ne puisse appartenir simultanément à plusieurs catégories, nous introduisons l'attribut Catégorie dont les valeurs possibles sont : «Cadre supérieur», «Spécialiste» et «Apprenti». Par cette méthode, nous garantissons la propriété d'une hiérarchie de généralisation disjointe (au sens défini dans la section 2.2.3) ; c'est-à-dire que l'intersection des ensembles d'entités spécialisés est vide. La propriété de complétude ne peut pas s'exprimer explicitement dans un schéma de base de données, et doit donc être implémentée sous la forme d'une contrainte d'intégrité spéciale.

Règle 7 (agrégation)

Règle de passage d'une structure MEMBRE-DE

Dans une structure d'agrégation, si le type de liens est *complexe-complexe*, une *table distincte* doit être créée pour l'ensemble d'entités et une deuxième pour l'ensemble de liens. La table de l'ensemble de liens contient deux fois la clé provenant de la table de l'ensemble

d'entités. La concaténation de ces deux clés, suffixées par des noms de rôles appropriés, définit la clé d'identification de la table de l'ensemble de liens. Si la liaison est de type *simple-complexe* (structure hiérarchique), nous pouvons combiner en *une seule table* l'ensemble d'entités et l'ensemble de liens.

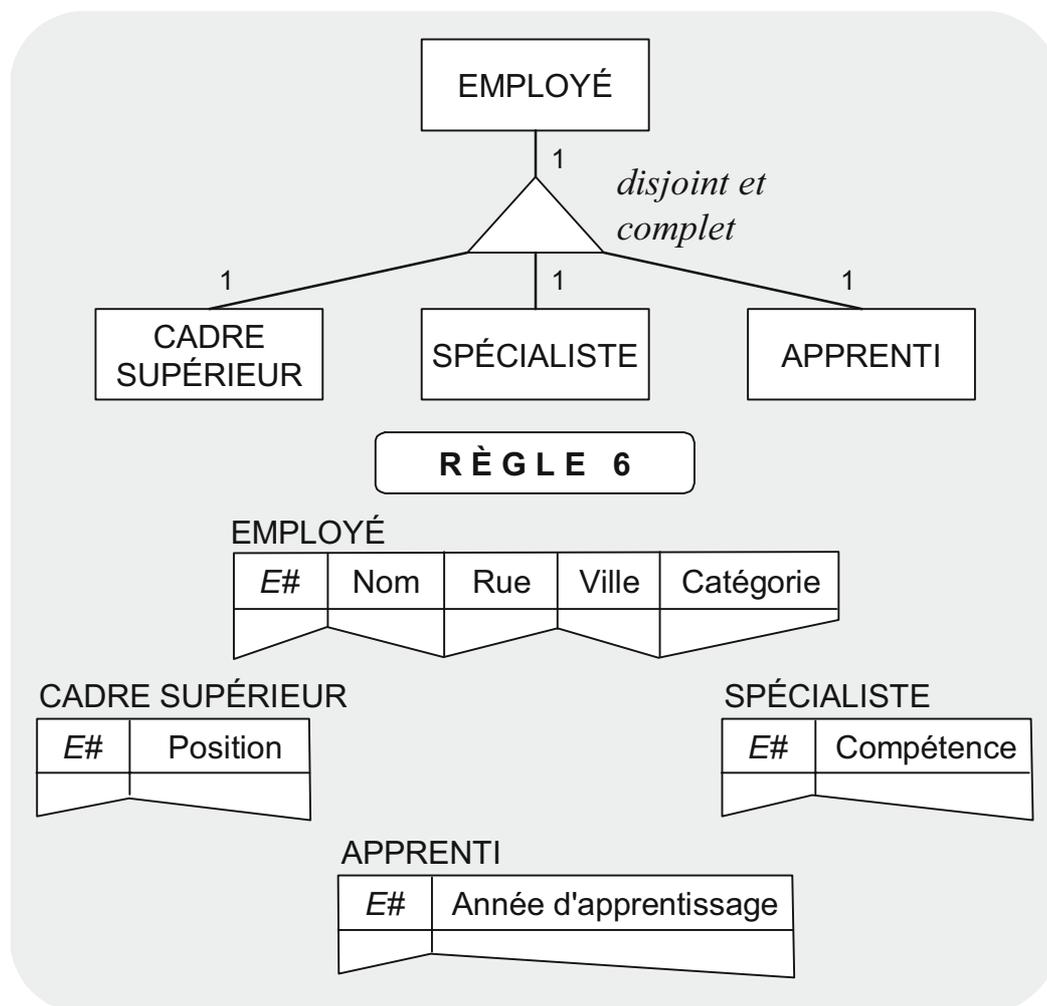
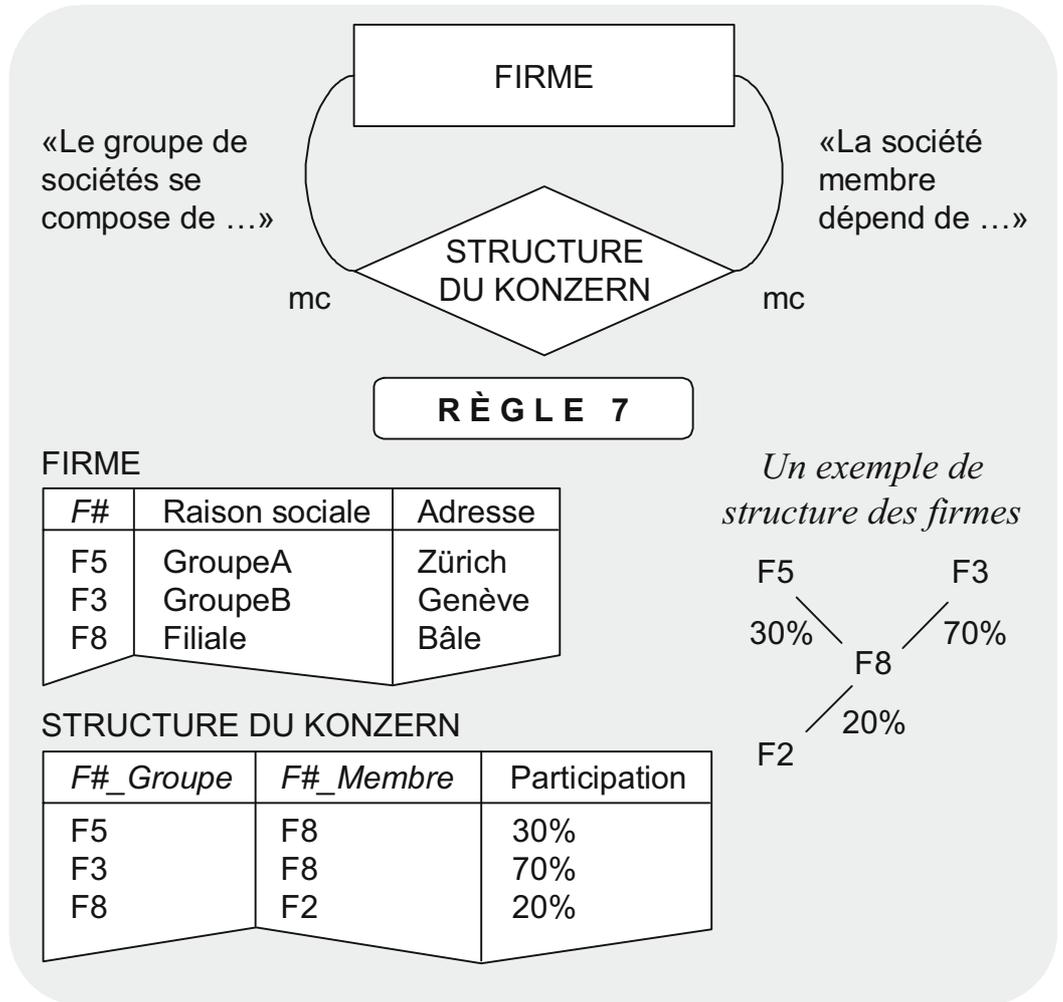


Figure 2-13
La généralisation
sous forme de
tables

Dans l'exemple de la figure 2-14, la structure du Konzern se caractérise par un degré de liaison de type (mc,mc). En vertu de la règle 7, deux tables doivent être créées : FIRME et STRUCTURE DU KONZERN. Pour chaque occurrence dans la table STRUCTURE DU KONZERN de l'ensemble de liens, la valeur de la clé d'identification contient deux informations : d'une part, la société directement affiliée à un groupe de firmes, et d'autre part, le groupe auquel appartient directement une société membre donnée. Outre les liens établis par les deux clés étrangères, l'attribut Participation complète la table STRUCTURE DU KONZERN.

Figure 2-14
Représentation
tabulaire de la
structure des
firmes en réseau



La structure hiérarchique d'une nomenclature

L'exemple de la figure 2-15 illustre une structure d'agrégation hiérarchique. La table ARTICLE enregistre les attributs de tous les composants. La table NOMENCLATURE décrit la structure hiérarchique des liens entre les sous-ensembles d'articles. Ainsi, l'article T7 est assemblé à partir de deux sous-ensembles, T9 et T11. Le sous-ensemble T11 est lui-même constitué des pièces T3 et T4.

La nomenclature peut être définie par une seule table

On peut fusionner les deux tables ARTICLE et NOMENCLATURE en une seule appelée STRUCTURE DES ARTICLES. Chaque article est caractérisé par un attribut supplémentaire qui donne le numéro de l'article de niveau immédiatement supérieur.

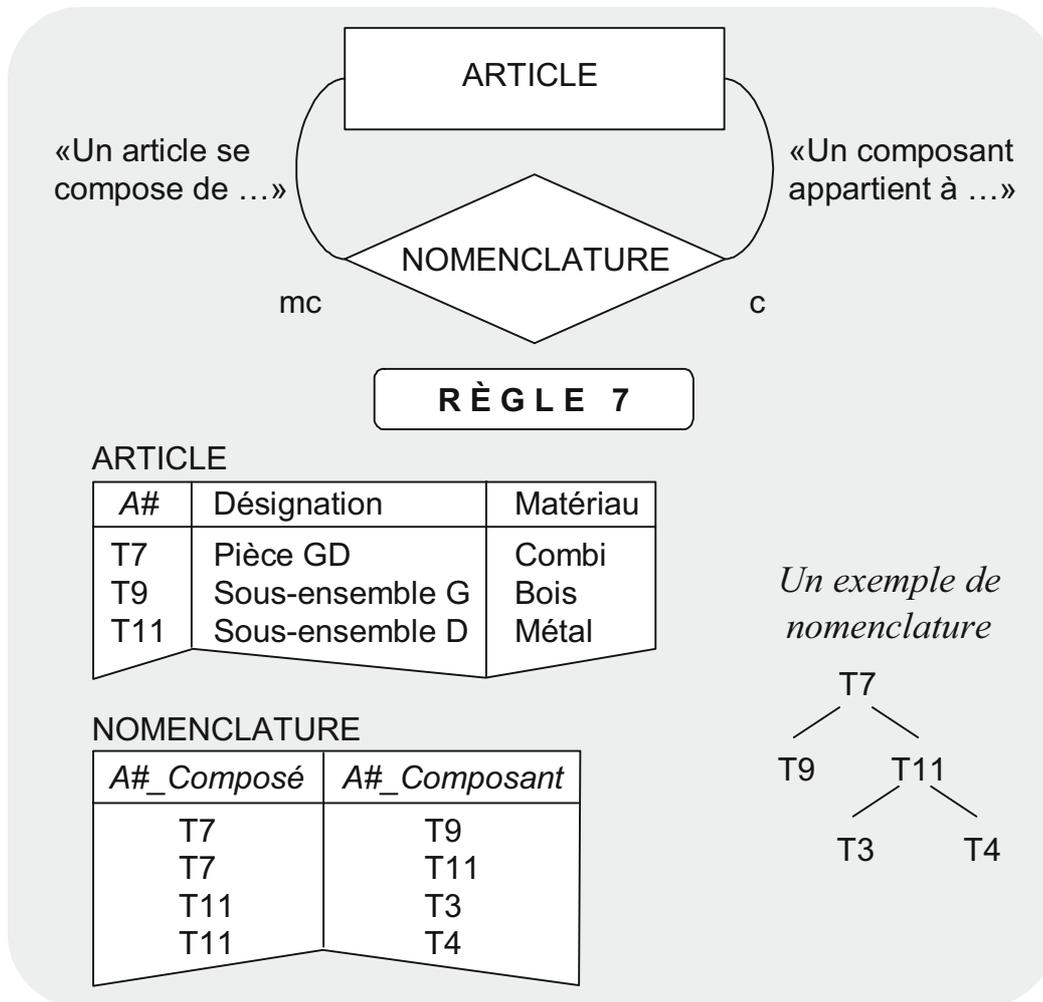


Figure 2-15
Représentation
tabulaire de la
structure
hiérarchique des
articles

2.4 Les dépendances entre données et les formes normales

2.4.1 La signification et le but des formes normales

La recherche dans le domaine relationnel a permis d'élaborer une véritable théorie des bases de données qui traite des aspects formels du modèle relationnel, en partie détachés du monde réel. Elle accorde une place importante à l'étude des *formes normales* (*normal forms*, en anglais). L'objectif consiste à détecter et à analyser les dépendances à l'intérieur des tables pour en éliminer les informations redondantes et les anomalies qui en résultent.

Théorie des formes normales

Redondance d'un attribut

Quand un attribut est-il redondant ?

Un attribut dans une table est redondant lorsque ses valeurs peuvent être *éliminées* de cette table *sans perte d'informations*.

À titre d'exemple, considérons une table, EMPLOYÉ DU DÉPARTEMENT, qui contient, pour chaque employé, le numéro d'employé, le nom, la rue, la ville, ainsi que le numéro et la description du département auquel il est affecté.

La description des départements est redondante

Dans la figure 2-16, le nom du département Finances apparaît pour chaque employé dans le département D6. Il en est de même pour les autres départements car, selon notre modèle de données, plusieurs employés peuvent travailler dans un même département. Puisque le nom d'un département apparaît plusieurs fois dans la table, l'attribut Description est donc redondant. Pour supprimer cette redondance, nous enregistrons une fois pour toutes la description de chaque département dans une table séparée au lieu de la répéter pour chaque employé.

Figure 2-16
Table contenant des redondances et des anomalies

EMPLOYÉ DU DÉPARTEMENT					
E#	Nom	Rue	Ville	D#	Description
E19	Savoy	avenue de la Gare	Romont	D6	Finances
E1	Meier	rue Faucigny	Fribourg	D3	Informatique
E7	Humbert	route des Alpes	Bulle	D5	Personnel
E4	Brodard	rue du Tilleul	Fribourg	D6	Finances

Une table redondante entraîne des anomalies

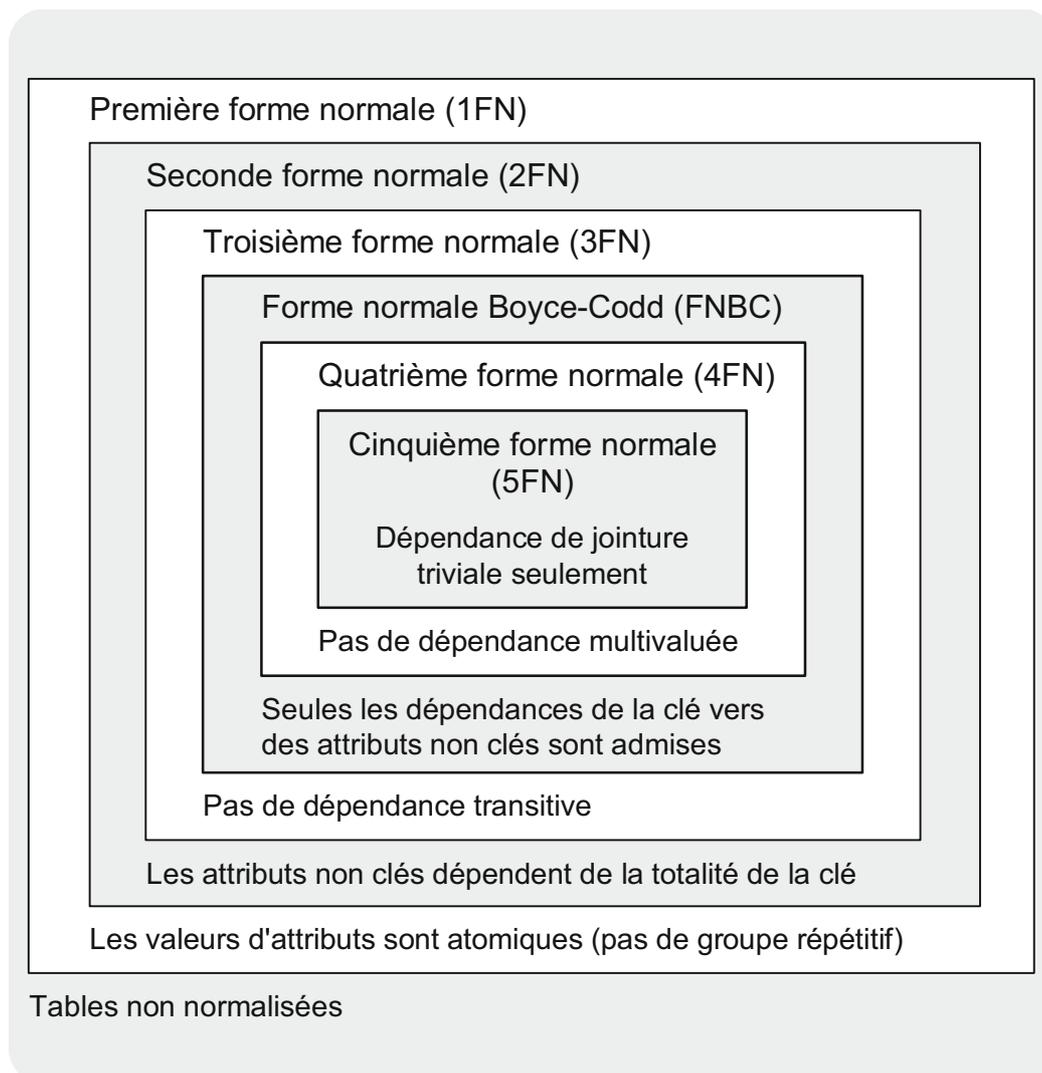
Une table contenant des informations redondantes peut entraîner des *anomalies de mutation*. Admettons qu'un nouveau département de marketing D9 vient d'être créé. Nous constatons qu'il est impossible de le définir dans la table EMPLOYÉ DU DÉPARTEMENT à la figure 2-16. Il existe une *anomalie d'insertion*, car nous ne pouvons insérer aucune nouvelle ligne dans la table sans donner un numéro d'employé unique.

Nous parlons d'une *anomalie de suppression* lorsque nous perdons des informations involontairement. En supprimant tous les employés dans notre table EMPLOYÉ DU DÉPARTEMENT, nous

perdons simultanément les numéros et les désignations des départements.

Enfin, il existe des *anomalies de mise à jour*. Si la description du département D3 devient «Traitement de l'information» au lieu «d'Informatique», nous devons appliquer ce changement de nom à l'ensemble des employés du département concerné. En d'autres termes, même si la modification ne porte que sur une seule pièce d'information, elle entraîne une mise à jour de la table EMPLOYÉ DU DÉPARTEMENT à plusieurs endroits différents. Cet inconvénient s'appelle anomalie de mise à jour.

Problèmes de mise à jour des tables redondantes



*Figure 2-17
Vue d'ensemble des formes normales et de leurs propriétés*

Les formes normales que nous abordons maintenant permettent d'éliminer les anomalies, et par là même les redondances qui en sont la cause. La figure 2-17 donne une vue d'ensemble des formes

Les dépendances engendrent des conflits

normales et de leur signification. Nous y constatons notamment que chaque forme normale traite d'un cas particulier de dépendance et de la situation conflictuelle qu'elle engendre. Nous étudierons ces dépendances de manière approfondie dans les prochaines sections.

Comme le montre la figure 2-17, les formes normales limitent l'ensemble de tables admises en fournissant des règles de plus en plus strictes. Par exemple, une table ou un schéma de base de données en troisième forme normale doit satisfaire les critères des première et deuxième formes normales, et en plus, ne doit contenir aucune dépendance transitive d'une clé vers les attributs non clés.

L'intérêt pratique de la troisième forme normale

L'étude des formes normales nous montrera qu'elles ne revêtent pas toutes la même importance. *Dans la pratique, on se limite généralement aux trois premières formes normales* car les dépendances multivaluées et les dépendances de jointure se présentent moins fréquemment ; les exemples d'application sont rares. C'est pourquoi les quatrième et cinquième formes normales feront seulement l'objet d'une brève discussion.

La compréhension des formes normales permet de mieux saisir les règles de passage du modèle entité-association au schéma de base de données relationnelle, présentées dans les sections précédentes. En effet, comme nous le verrons plus tard, les critères des formes normales sont automatiquement satisfaits lorsqu'un modèle entité-association est bien construit et que les règles de passage sont correctement appliquées. En d'autres termes, nous pouvons *nous dispenser de vérifier les formes normales à chaque étape de la conception* si, partant d'un modèle entité-association bien conçu, nous appliquons les règles de passage 1 à 7 pour construire le schéma de base de données correspondant.

2.4.2 Les dépendances fonctionnelles

La première forme normale est le point de départ vers les autres formes normales. Elle se définit comme suit :

Première forme normale (1FN)

Une table satisfait à la première forme normale si les *domaines de ses attributs sont constitués de valeurs atomiques*. La première forme normale exige que chaque valeur d'un attribut provienne d'un domaine non structuré. Par conséquent, la définition du domaine d'un attribut ne doit pas contenir d'ensembles, de types énumérés ou de groupes répétitifs.

La 1FN exige des valeurs d'attributs atomiques

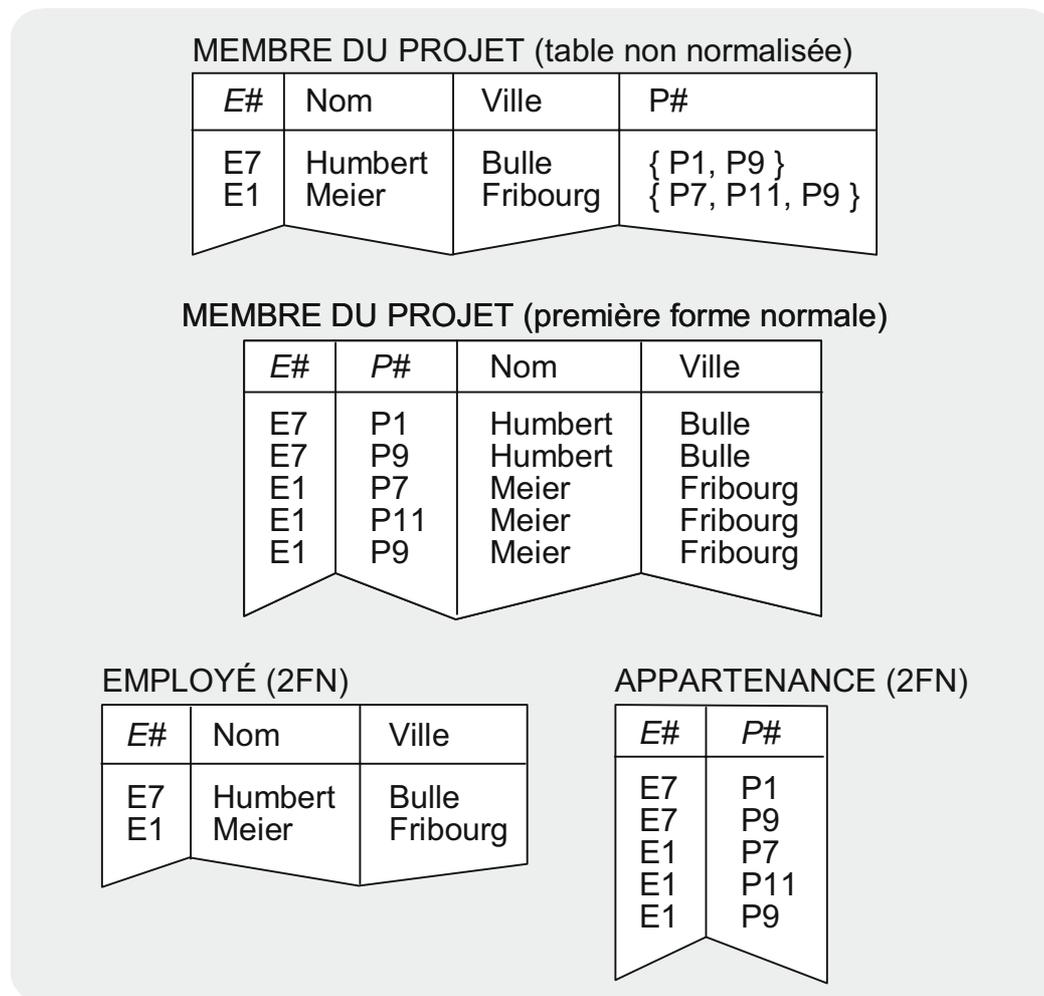


Figure 2-18

Tables en première et seconde formes normales

Dans la figure 2-18, la table MEMBRE DU PROJET n'est pas encore normalisée car chacun de ses tuples contient plusieurs numéros de projets auxquels participe un employé. La technique pour transformer une table non normalisée en première forme normale consiste simplement à créer un tuple distinct pour chaque engagement dans un projet. Lors de cette transformation, la clé de la table MEMBRE DU PROJET doit être redéfinie. En effet, afin d'identifier un tuple de manière unique, nous avons besoin à la fois du numéro

Mise en première forme normale

d'employé et du numéro de projet. Les deux attributs qui forment ensemble une clé composée sont généralement (mais pas obligatoirement) placés dans les premières colonnes adjacentes de la table (Figure 2-18).

Paradoxalement, la transformation en première forme normale nous donne une table qui comporte des redondances. Dans la figure 2-18, le nom et la ville d'un employé sont redondants, car ils se répètent à chacune de ses participations aux projets. La deuxième forme normale nous permet d'y remédier.

Deuxième forme normale (2FN)

La 2FN exige la dépendance fonctionnelle totale

Une table satisfait à la deuxième forme normale si elle est déjà en première forme normale et s'il existe une *dépendance fonctionnelle totale* reliant la clé à chaque attribut non clé.

Dépendance fonctionnelle entre les attributs

Un attribut B est *fonctionnellement dépendant* de l'attribut A, si à chaque valeur de A correspond une et une seule valeur de B (noté conventionnellement $A \rightarrow B$). Par conséquent, la *dépendance fonctionnelle* (*functional dependency*, en anglais) de A vers B exprime le fait que chaque valeur de A détermine de manière unique une valeur de B. Une propriété connue des clés d'identification est que les attributs non clés dépendent de la clé de manière unique. Dans une table donnée, il existe donc une dépendance fonctionnelle $S \rightarrow B$ entre la clé d'identification S et un attribut quelconque B.

Dépendance fonctionnelle totale envers une clé composée

Dans le cas des clés composées, nous devons compléter la notion de dépendance fonctionnelle par celle de la dépendance fonctionnelle totale. Un attribut B est en *dépendance fonctionnelle totale* envers une clé composée de S1 et S2 (notée $(S1, S2) \Rightarrow B$), si B est fonctionnellement dépendant de toute la clé composée, pas seulement d'une partie de cette clé. La dépendance fonctionnelle totale exprime donc le fait que la totalité de la clé composée détermine de manière unique les attributs non clés. En revanche, nous disons qu'il existe une dépendance fonctionnelle $(S1, S2) \rightarrow B$, si $S1 \rightarrow B$ (S1 détermine B) ou $S2 \rightarrow B$ (S2 détermine B). Dans la *dépendance fonctionnelle totale* (*full functional dependency*, en anglais) d'une clé vers un attribut, celui-ci ne doit donc pas dépendre des parties de cette clé.

Considérons la première forme normale de la table MEMBRE DU PROJET dans la figure 2-18. Elle contient la clé composée (E#,P#) dont il faut maintenant vérifier la dépendance fonctionnelle totale vers les attributs non clés, le nom et la ville des membres de différents projets. Nous constatons qu'il existe deux dépendances fonctionnelles (E#,P#)→Nom et (E#,P#)→Ville. Chaque combinaison du numéro d'employé et du numéro de projet détermine de manière univoque le nom et la ville d'un employé. Néanmoins, il est évident que le nom et la ville d'un employé n'ont aucun rapport avec les numéros de ses projets. Cela signifie que les deux attributs non clés sont fonctionnellement dépendants d'une partie de la clé seulement, à savoir E#→Nom et E#→Ville, ce qui est contraire à la définition de la dépendance fonctionnelle totale. Nous en concluons que la table MEMBRE DU PROJET n'est pas à la deuxième forme normale.

Les dépendances partielles ne doivent pas être présentes

Une table dotée d'une clé composée, qui n'est pas à la deuxième forme normale, doit être décomposée en sous-tables. D'abord, on crée une table distincte constituée des attributs qui dépendent d'une partie de la clé, et de cette dite partie. Ensuite, on définit la deuxième table formée de la clé composée et d'éventuels attributs qui caractérisent la liaison.

Les tables redondantes doivent être décomposées

Dans l'exemple de la figure 2-18, l'éclatement donne lieu à deux tables, EMPLOYÉ et APPARTENANCE, qui sont toutes en première et deuxième formes normales. La table EMPLOYÉ ne contient plus de clé composée, et satisfait évidemment la deuxième forme normale. La table APPARTENANCE ne possède aucun attribut non clé, ce qui rend inutile la vérification des critères de la deuxième forme normale.

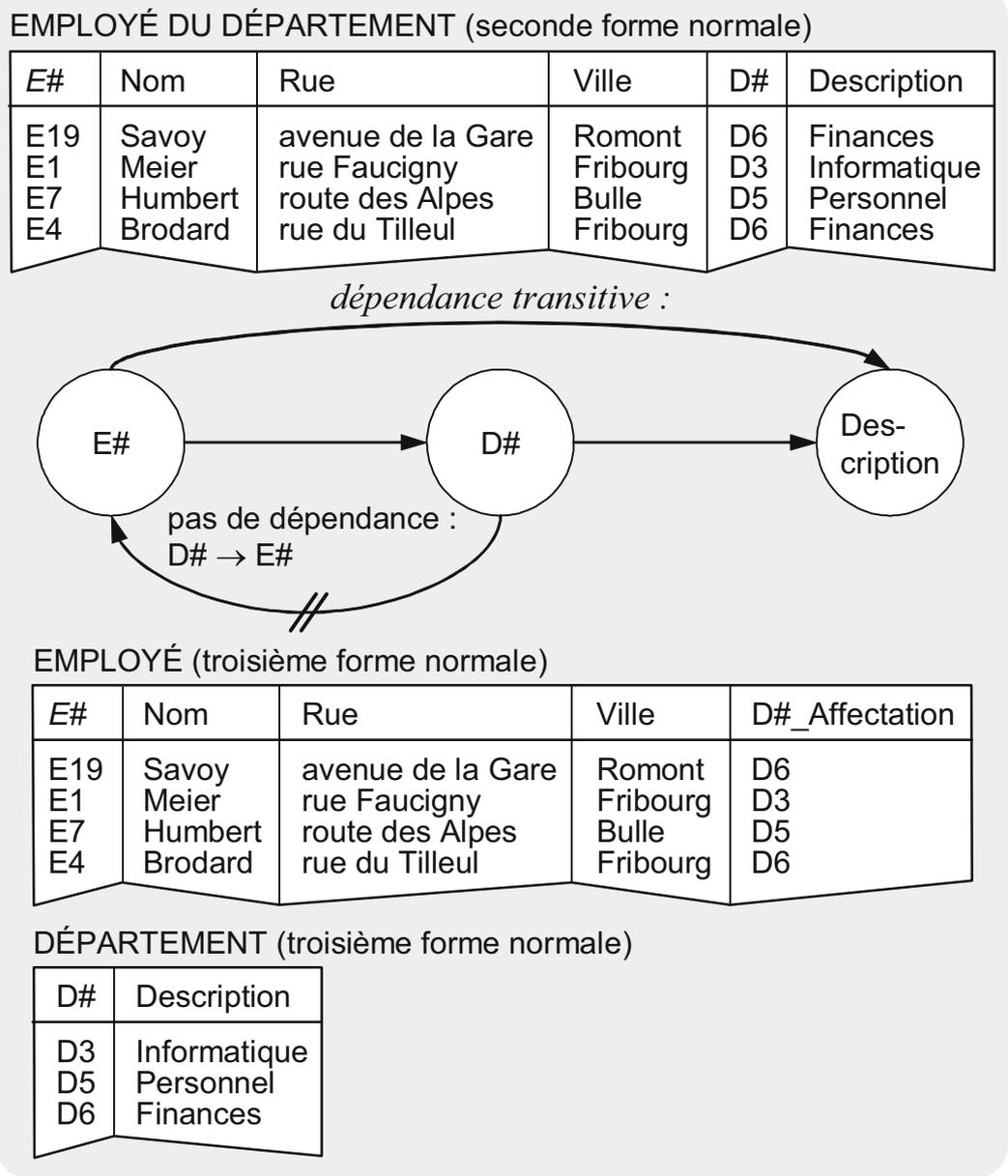
2.4.3 Les dépendances transitives

La figure 2-19 présente la table EMPLOYÉ DU DEPARTEMENT dont nous avons discuté auparavant. Elle contient les coordonnées des employés ainsi que les informations sur les départements. Nous constatons immédiatement qu'elle est en première et deuxième formes normales. Comme la table ne possède pas de clé composée, nous n'avons pas besoin de vérifier la propriété de la dépendance fonctionnelle totale. Néanmoins, l'attribut Description est

Une table redondante, bien qu'en deuxième forme normale

manifestement redondant. La troisième forme normale, expliquée dans cette section, nous permettra d'éliminer ce défaut.

Figure 2-19
Dépendance transitive et troisième forme normale



Troisième forme normale (3FN)

La 3FN interdit toute dépendance transitive

Une table est en troisième forme normale si elle est déjà en deuxième forme normale et qu'aucun attribut non clé ne dépend d'une clé quelconque par transitivité.

Que signifie la dépendance transitive ?

De nouveau, nous définissons une forme normale fondée sur le critère de dépendance. La dépendance transitive est une dépendance fonctionnelle déduite des autres dépendances. Par exemple, l'attribut

Description dépend fonctionnellement du numéro d'employé par le biais du numéro de département. En effet, nous savons qu'il existe une dépendance fonctionnelle entre le numéro d'employé et le numéro de département d'une part, et entre le numéro de département et sa description d'autre part. De ces deux dépendances fonctionnelles, $E\# \rightarrow D\#$ et $D\# \rightarrow \text{Description}$, nous dérivons la dépendance transitive $E\# \rightarrow \text{Description}$ par le biais du numéro de département.

En principe, étant donné deux dépendances fonctionnelles $A \rightarrow B$ et $B \rightarrow C$ avec un attribut commun B, on en déduit par transitivité la dépendance fonctionnelle $A \rightarrow C$. Plus précisément, si, de manière univoque, A détermine B et B détermine C, la détermination de C par A hérite également de cette propriété. Par conséquent, la dépendance $A \rightarrow C$ est fonctionnelle ; en outre, elle est dite transitive si A ne dépend pas fonctionnellement de B en même temps que les dépendances fonctionnelles $A \rightarrow B$ et $B \rightarrow C$ ⁵. Nous arrivons ainsi à la définition suivante de la *dépendance transitive* (*transitive dependency*, en anglais) : l'attribut C dépend de A par transitivité si B dépend fonctionnellement de A, C dépend fonctionnellement de B et A ne dépend pas fonctionnellement de B.

Dans la figure 2-19, la table EMPLOYÉ DU DÉPARTEMENT contient l'attribut Description qui dépend de E# par transitivité. Donc, en vertu de la définition précédente, elle n'est pas en troisième forme normale. Nous éliminons la dépendance transitive par éclatement de la table EMPLOYÉ DU DÉPARTEMENT en deux tables distinctes : EMPLOYÉ et DÉPARTEMENT. Ce dernier contient l'attribut redondant Description et le numéro de département. Dans l'autre table EMPLOYÉ, nous conservons le numéro de département comme clé étrangère en l'appelant D#_Affectation. De cette manière, le lien entre l'employé et son département est préservé.

Vers une définition de la dépendance transitive

Suppression des dépendances transitives par décomposition en sous-tables

⁵ Les clés candidates dépendent toujours de la clé d'identification et inversement. C'est le cas, par exemple, lorsque la table EMPLOYÉ contient le numéro d'employé et le numéro AVS.

2.4.4 Les dépendances multivaluées

Les deuxième et troisième formes normales nous ont permis d'éliminer les redondances parmi les attributs non clés. Cependant, la détection des informations redondantes ne doit pas se limiter aux attributs non clés, car les clés composées peuvent aussi être redondantes.

Forme normale de Boyce-Codd

Une extension de la troisième forme normale s'est avérée nécessaire et résulte des travaux de deux auteurs qui ont introduit une forme normale portant leurs noms, la «*forme normale de Boyce-Codd*» ou FNBC. La question de la forme normale de Boyce-Codd se pose lorsqu'une table possède plusieurs clés candidates. Il existe ainsi des tables où les clés se chevauchent et qui, tout en étant en troisième forme normale, transgressent la forme normale de Boyce-Codd. Dans ce cas, pour mettre une table en FNBC, il faut la décomposer d'après les clés candidates. Dans la section 2.8, une notice bibliographique oriente le lecteur dans la littérature qui traite de ce sujet.

Un exemple de dépendance multivaluée

L'analyse des *dépendances multivaluées* (*multi-valued dependency*, en anglais) entre les attributs formant une clé conduit à une autre forme normale. Quoiqu'en pratique, les dépendances multivaluées ne jouent qu'un rôle secondaire, nous allons illustrer brièvement cette forme normale à l'aide d'un exemple simple dans la figure 2-20. Considérons la table initiale MÉTHODE qui est non normalisée, car à chaque méthode peuvent correspondre plusieurs auteurs et plusieurs concepts. Ainsi, la méthode du structogramme a pour auteurs Nassi et Shneiderman. En outre, elle est caractérisée par plusieurs valeurs de l'attribut Concept, à savoir les trois éléments du langage, la séquence, l'itération et la sélection.

Après avoir transformé la table non normalisée en première forme normale, les ensembles {Nassi, Shneiderman} et {Séquence, Itération, Sélection} disparaissent. Nous constatons que la nouvelle table consiste en attributs clés uniquement. Elle est non seulement en première forme normale, mais aussi en deuxième et troisième formes normales. Cependant, malgré ces propriétés, la table contient encore des informations redondantes. Par exemple, nous observons que les trois éléments du langage (concepts), la séquence, l'itération et la

sélection, s'appliquent à chaque auteur du structogramme. Inversement, les noms des deux auteurs, Nassi et Shneiderman, se répètent pour chaque concept du structogramme. En d'autres termes, nous sommes ici en présence d'une paire de dépendances multivaluées qu'il faut éliminer.

MÉTHODE (table non normalisée)

Méthode	Auteur	Concept
Structogramme	{Nassi, Shneiderman}	{Séquence, Itération, Sélection}
Modèle de données	{Chen}	[Ensemble d'entités, Ensemble de liens]

MÉTHODE (troisième forme normale)

Méthode	Auteur	Concept
Structogramme	Nassi	Séquence
Structogramme	Nassi	Itération
Structogramme	Nassi	Sélection
Structogramme	Shneiderman	Séquence
Structogramme	Shneiderman	Itération
Structogramme	Shneiderman	Sélection
Modèle de données	Chen	Ensemble d'entités
Modèle de données	Chen	Ensemble de liens

INVENTEUR (4FN)

Méthode	Auteur
Structogramme	Nassi
Structogramme	Shneiderman
Modèle de données	Chen

MÉTHODOLOGIE (4FN)

Méthode	Concept
Structogramme	Séquence
Structogramme	Itération
Structogramme	Sélection
Modèle de données	Ensemble d'entités
Modèle de données	Ensemble de liens

Figure 2-20
Table contenant
des dépendances
multivaluées

Soit une table avec trois attributs A, B et C. La dépendance multivaluée se définit comme suit : il existe une *dépendance multivaluée* de l'attribut A vers l'attribut C (notée $A \twoheadrightarrow C$), si chaque combinaison d'une valeur spécifique de A avec une valeur quelconque de B détermine un ensemble identique de valeurs de C.

*Définition des
attributs en
dépendance
fonctionnelle
multivaluée*

*Exemple illustrant
deux dépendances
multivaluées*

L'exemple en figure 2-20 nous montre que l'attribut Méthode multidétermine l'attribut Concept : Méthode \twoheadrightarrow Concept. La méthode Structogramme, combinée à l'auteur Nassi ou à Shneiderman, détermine dans les deux cas le même ensemble {Séquence, Itération, Sélection}. En outre, il existe une dépendance multivaluée entre les attributs Auteur et Méthode, dans laquelle Méthode multidétermine Auteur : Méthode \twoheadrightarrow Auteur. En combinant une méthode spécifique, le structogramme par exemple, à n'importe quel concept tel que la séquence, nous obtenons deux auteurs, Nassi et Shneiderman. Ce même ensemble d'auteurs s'obtient en combinant le structogramme au concept d'itération ou de sélection.

*La 4FN interdit la
présence de deux
dépendances
multivaluées dans
la même table*

Les dépendances multivaluées dans une table entraînent des redondances et des anomalies. Pour les supprimer, nous devons considérer une autre forme normale. La *quatrième forme normale* n'admet pas qu'une table contienne en même temps deux dépendances multivaluées. Pour revenir à notre exemple, nous devons décomposer la table MÉTHODE en deux sous-tables, INVENTEUR et MÉTHODOLOGIE. La première réunit les attributs Méthode et Auteur, la deuxième Méthode et Concept. Les deux tables ne contiennent plus de redondances et sont en quatrième forme normale.

*Concept de
dépendance de
jointure*

Il n'est pas du tout évident de réussir à décomposer une table en sous-tables sans perte d'informations. C'est pourquoi des critères ont été définis afin de garantir une décomposition de tables *sans perte d'informations*. Plus précisément, les informations originelles doivent se retrouver dans les nouvelles tables obtenues par décomposition (concept de *dépendance de jointure* ; *join dependency*, en anglais).

*La 5FN assure la
décomposition de
tables sans perte
d'information*

La *cinquième forme normale*, appelée aussi forme normale de projection-jointure, nous indique les circonstances où une table peut être décomposée sans problème et, le cas échéant, reconstruite sans restriction. La décomposition s'effectue à l'aide de l'opérateur de projection, et la reconstruction par l'opérateur de jointure. Le prochain chapitre explique ces opérateurs en détail.

Il existe des cas où la projection entraîne une perte d'informations originelles qu'il est impossible de restaurer plus tard, même à l'aide d'une opération de jointure. Comme indiqué ci-dessus,

la cinquième forme normale définit clairement les critères pour juger si l'on peut poursuivre la décomposition des tables sans perte. Cela signifie aussi que, sous certaines hypothèses, il n'existe aucune autre forme normale, supérieure à celles étudiées jusqu'à présent (voir notices bibliographiques dans la section 2.8).

Dans la conception des bases de données, une question intéressante consiste à se demander s'il existe aussi une approche synthétique à l'inverse des règles de décomposition et des formes normales (méthode analytique). Des algorithmes ont été effectivement développés pour composer une table en troisième forme normale au moins, à partir de plusieurs sous-tables. Des règles de composition sont en partie nécessaires pour élaborer un schéma de base de données depuis un ensemble de dépendances. Elles montrent par là que les approches analytique (descendante, «top down») et synthétique (ascendante, «bottom up») contribuent toutes les deux au développement réussi d'un schéma de base de données relationnelle. Malheureusement, à l'heure actuelle, peu d'outils CASE supportent simultanément ces deux approches, et dans une large mesure, c'est l'architecte de données qui doit vérifier manuellement la correction de son modèle (voir section 2.7).

Les deux approches, analytique et synthétique

2.5 Les contraintes d'intégrité structurelles

Le concept d'*intégrité* ou de *cohérence* (*integrity, consistency*, en anglais) signifie l'absence d'incohérence dans les ensembles de données. Si les données enregistrées étaient introduites sans erreur et qu'elles délivrent les informations désirées de manière correcte, on dit que la base de données en question est intègre ou cohérente. En revanche, l'intégrité d'une base de données est violée si les données présentent des ambiguïtés ou des incohérences. Quand nous certifions par exemple que la table EMPLOYÉ est cohérente, nous admettons par là que les noms des employés, les noms des rues et des villes, etc., sont corrects et existent aussi dans la réalité.

L'intégrité garantit l'absence d'incohérence

Quelles sont les contraintes d'intégrité structurelles ?

Les *contraintes d'intégrité structurelles* établissent des règles qui doivent s'appliquer à un schéma de base de données pour assurer sa cohérence. Dans le contexte des bases de données relationnelles, les contraintes d'intégrité structurelles sont classées en trois catégories :

- *contrainte d'unicité* : chaque table possède une clé d'identification (un attribut ou une combinaison d'attributs) qui sert à différencier les tuples dans la table de manière unique ;
- *contrainte de domaine* : un attribut dans une table ne peut prendre que des valeurs appartenant à un domaine prédéfini ;
- *contrainte d'intégrité référentielle* : chaque valeur d'une clé étrangère doit correspondre à une valeur existante de la clé dans la table référencée.

L'unicité doit être assurée par le système de gestion des bases de données

La *contrainte d'unicité* requiert une clé par table. Lorsqu'il existe plusieurs clés candidates dans une même table, la contrainte d'unicité nous oblige à en déclarer une comme clé primaire. C'est le système de gestion des bases de données qui vérifie l'unicité des valeurs d'une clé primaire.

Comment vérifier les contraintes de domaine ?

En revanche, ce système ne peut pas garantir le respect total d'une *contrainte de domaine*. Certes, nous pouvons définir un domaine par colonne dans une table, mais les domaines ne représentent qu'une petite partie des règles de validation. Par exemple, la définition d'un domaine n'a pas de sens lorsqu'il s'agit de vérifier la correction des noms de ville ou de rue. En effet, la spécification «CHARACTER (20)» qui désigne le type et la taille d'une chaîne de caractères ne permet pas de vérifier si le nom d'une ville ou d'une rue existe. Il incombe aux utilisateurs, dans une large mesure, de définir jusqu'à quel point le contenu des tables doit être validé.

Importance des contraintes de domaine

La définition d'une contrainte de domaine basée sur les *types énumérés* est une technique importante qui consiste à créer une liste de toutes les valeurs possibles d'un attribut. Comme exemples de types énumérés, nous pouvons imposer aux attributs Profession et Année les contraintes de domaine suivantes : Profession = {Programmeur, Analyste, Organisateur}, Année = {1950..1990}. La

plupart des systèmes de gestion de bases de données actuels prennent en charge ce genre de règles de validation.

Le concept d'*intégrité référentielle* (*referential integrity*, en anglais) introduit une classe importante de règles de validation des données. Une base de données relationnelle respecte la contrainte d'intégrité référentielle si, pour toute valeur d'une clé étrangère, il existe une valeur identique de la clé primaire correspondante. Considérons l'exemple dans la figure 2-21. La table DÉPARTEMENT admet le numéro de département D# comme clé primaire. Celle-ci est utilisée dans la table EMPLOYÉ comme clé étrangère associée à l'attribut D#_Affectation qui détermine le département auquel un employé est affecté. Le lien entre les clés primaire et secondaire respecte la contrainte d'intégrité si, dans la table EMPLOYÉ, tous les numéros de département de la clé étrangère existent comme valeurs de la clé primaire dans la table DÉPARTEMENT. Dans notre exemple, nous constatons qu'aucune affectation ne transgresse la règle de l'intégrité référentielle.

Que signifie l'intégrité référentielle ?

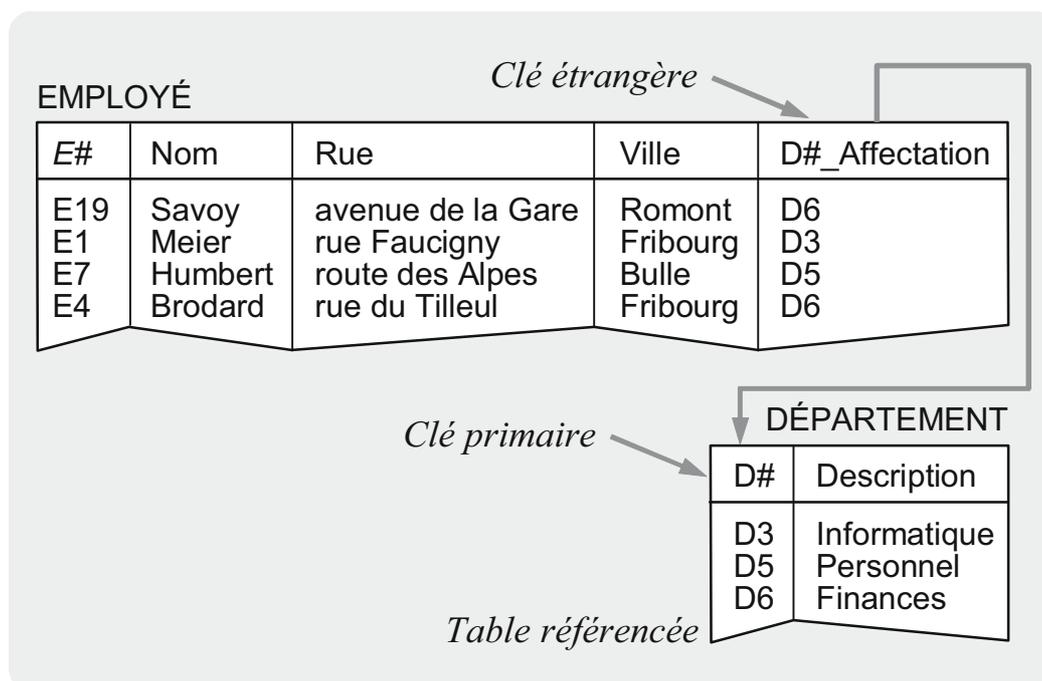


Figure 2-21
La garantie de l'intégrité référentielle

Soit «E20, Morel, chemin du Cerisier, Marly, D7» un nouveau tuple que nous désirons insérer dans la table EMPLOYÉ. Cette insertion sera rejetée par un système de gestion de base de données

Violation de l'intégrité

qui supporte l'intégrité référentielle. La valeur D7 sera déclarée invalide, car elle n'existe pas dans la table référencée DÉPARTEMENT.

La garantie de l'intégrité référentielle déclenche des actions spécifiques non seulement lors de l'insertion des données, mais aussi dans les autres opérations sur une base de données. Lorsqu'on veut supprimer un tuple dans une table, et que ce tuple est référencé par d'autres tuples dans une table étrangère, le système peut se comporter de plusieurs manières.

*Suppression
restreinte*

En mode de *suppression restreinte* (*restricted deletion*, en anglais), l'opération ne sera pas exécutée tant que le tuple à supprimer est référencé par un tuple dans une autre table. Par exemple, si nous voulons détruire le tuple «D6, Finances» dans la figure 2-21, notre opération sera refusée en vertu de la règle de suppression restreinte, car les deux employés Savoy et Brodard travaillent dans le département D6.

*Suppression en
cascade*

Au lieu de la suppression restreinte, nous pouvons opter pour le mode de *suppression en cascade* (*cascaded deletion*, en anglais) dans la spécification des deux tables EMPLOYÉ et DÉPARTEMENT. En mode cascade, la suppression d'un tuple entraîne celle de tous les tuples dépendants. Dans notre exemple de la figure 2-21, si nous demandons la suppression en cascade du tuple «D6, Finances», alors les deux tuples «E19, Savoy, avenue de la Gare, Romont, D6» et «E4, Brodard, rue du Tilleul, Fribourg, D6» seront détruits en même temps.

*Suppression avec
mise à la valeur
nulle*

Dans le cadre de l'intégrité référentielle, une troisième option consiste à donner la valeur «inconnue» aux clés étrangères référencées lors d'une suppression. Nous reviendrons à cette troisième règle en détail au chapitre 3 lors de notre discussion sur les valeurs nulles. Enfin, les opérations de mise à jour peuvent aussi être soumises à des contraintes qui garantissent en permanence l'intégrité référentielle d'une base de données.

2.6 L'architecture de données d'entreprise est vitale

Selon de nombreuses études, durant la phase de définition et de développement d'un système d'information, les futurs utilisateurs expriment leurs besoins en mettant l'accent sur des fonctions complexes et des procédures sophistiquées à réaliser, alors qu'à l'usage du système, ils attachent au contraire *une importance stratégique aux données et à la disponibilité des informations tenues à jour*. C'est pourquoi il est impératif qu'un architecte de données examine les questions suivantes dès le départ : Quelles sont les données que l'entreprise devra elle-même collecter ? Quelles sont les données qui seront alimentées par des fournisseurs d'informations externes ? Comment faut-il classer et structurer les ensembles de données en tenant compte des réalités nationales et internationales ? Qui est responsable de la maintenance et de la mise à jour des données réparties géographiquement ? Quelles sont les dispositions relatives à la protection et à la sécurité des données dans le contexte international ? Quels sont les droits et les obligations en matière d'échange et de transmission des données ? Ces questions montrent la nécessité vitale d'une architecture de données pour l'entreprise et placent le modèle de données de l'entreprise au centre de ses préoccupations.

Face aux besoins sans cesse croissants des utilisateurs, les activités d'analyse et de conception sont menées la plupart du temps pour ne mettre en œuvre que des fonctions additionnelles, ou au mieux, des applications dans un domaine d'activité spécifique de l'entreprise. Dans la réalisation des bases de données relationnelles, une telle approche comporte le risque de créer une multitude de tables ad hoc ou de satisfaire uniquement les besoins d'une application locale. Il en résultera une *prolifération incontrôlée de tables* pleines de redondances et d'ambiguïtés. Le chaos des données est inévitable. L'existence d'un système valable pour l'ensemble des applications de l'entreprise devient problématique, ou ne peut être envisagée qu'à grands frais.

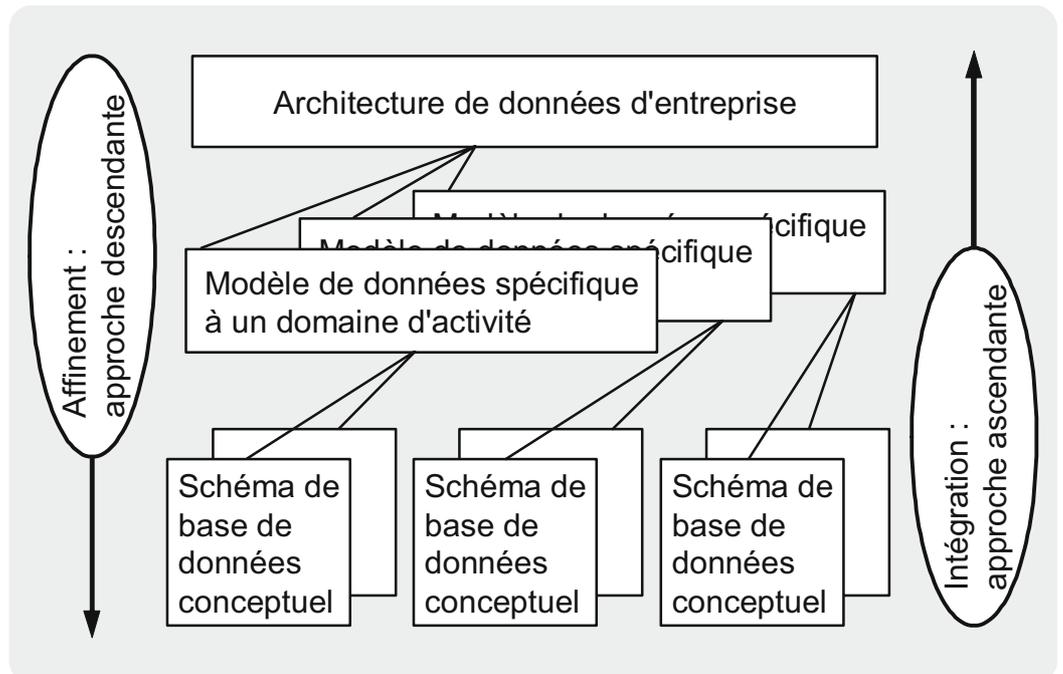
De la pérennité des données

Comment se dirige-t-on vers un chaos des données ?

Nécessité d'une
architecture de
données
d'entreprise

Le remède à cette situation réside dans une *architecture de données d'entreprise* (*corporate-wide data architecture*, en anglais) où sont définis les principaux ensembles d'entités et de liens dans une perspective globale à long terme. Elle englobe tous les secteurs d'activités de l'entreprise, dépasse les vues locales de données et permet de comprendre l'interdépendance globale des données de l'entreprise. L'architecture de données d'entreprise et le modèle de données qui en résulte constituent la *base d'un développement intégré des systèmes d'information de l'entreprise*.

Figure 2-22
Niveaux
d'abstraction dans
l'architecture de
données
d'entreprise



Les niveaux d'une
architecture de
données

Le schéma dans la figure 2-22 présente la relation entre le modèle de données d'entreprise et les modèles de données spécifiques aux applications. L'architecture de données d'entreprise définit les classes de données indispensables à la vie de l'entreprise et leurs liens. À partir de là on élabore des modèles de données propres à chaque domaine d'activité. Ces modèles seront ensuite convertis en schémas conceptuels de bases de données par application. Naturellement, dans la pratique, ces affinements successifs ne se réalisent pas en appliquant exclusivement la méthode descendante (*top down*, en anglais), car le temps y manque. Pour modifier un système d'information existant ou pour développer une nouvelle application, nous adoptons plutôt l'approche ascendante (*bottom up*, en anglais) afin de concevoir des schémas de bases de données conceptuels qui

soient conformes à la fois aux modèles de données sectoriels existants, incomplets, et à l'architecture de données d'entreprise. Ainsi conçus par étapes successives, les schémas de bases de données s'alignent sur le développement à long terme de l'entreprise.

Comme alternative au développement interne de ses modèles de données par domaines d'activité, l'entreprise peut se procurer des *modèles de données par branches*, commercialisés sur le marché des logiciels. Grâce aux efforts de normalisation des modèles de données, l'intégration des logiciels d'application achetés verra son coût diminuer. En outre, les modèles de données par branches facilitent l'échange d'informations inter-entreprises ou au sein d'un Konzern.

Emploi des modèles de données par branches

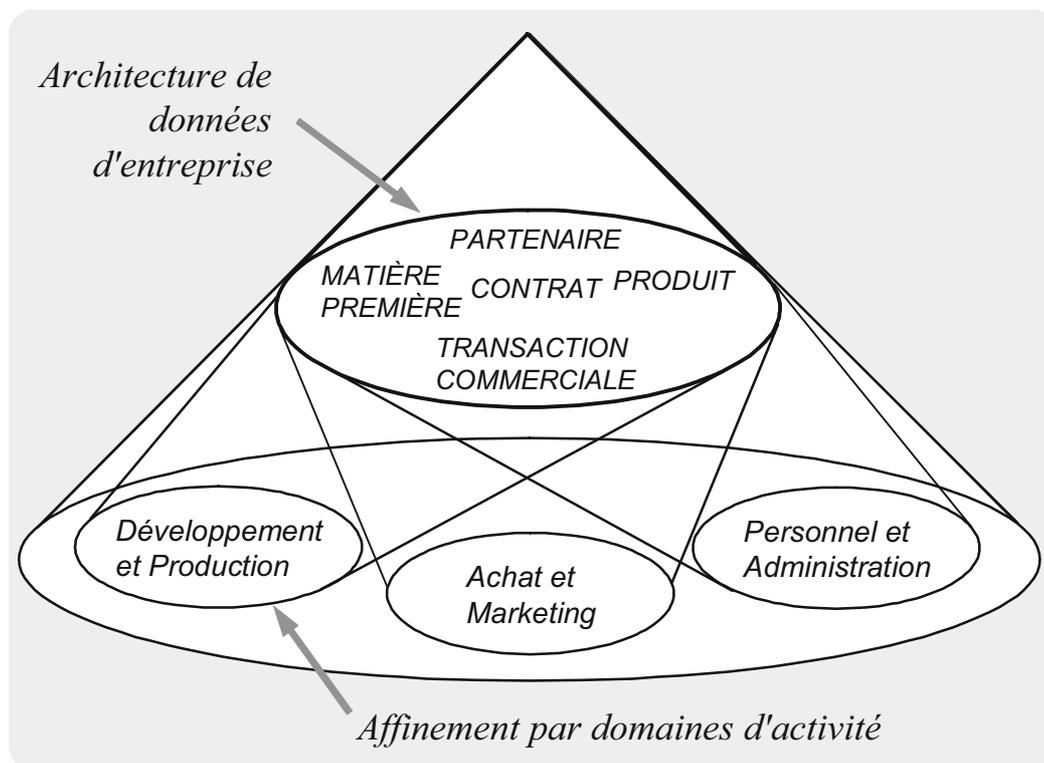


Figure 2-23
Les domaines d'activité de l'entreprise : vue orientée données

Illustrons l'architecture de données globale d'une entreprise à travers un exemple simplifié en nous limitant à cinq ensembles d'entités suivants : PARTENAIRE, MATIÈRE PREMIÈRE, PRODUIT, CONTRAT et AFFAIRE (voir figure 2-23) :

Les principaux composants

- Clients et fournisseurs appartiennent à l'ensemble PARTENAIRE*
- L'ensemble d'entités PARTENAIRE comprend toutes les personnes physiques et morales qui intéressent l'entreprise et dont les informations lui sont indispensables dans la conduite des affaires. Ce sont notamment les clients, les employés, les fournisseurs, les actionnaires, les personnes morales de droit public, les institutions et les firmes.
- L'ensemble MATIÈRE PREMIÈRE englobe les biens matériels et numériques*
- L'ensemble d'entités MATIÈRE PREMIÈRE inclut les matières premières, les métaux, les devises, les papiers-valeurs, les biens immobiliers disponibles sur le marché et achetés, traités ou transformés par l'entreprise. En principe, il s'agit de valeurs matérielles ou immatérielles. Par exemple, une société de conseil peut acquérir une technologie déterminée et un savoir-faire correspondant.
- L'ensemble PRODUIT définit les prestations offertes*
- L'ensemble d'entités PRODUIT est défini par l'assortiment des produits de l'entreprise et par la gamme de ses prestations de services. Ici aussi, les biens produits peuvent être matériels ou immatériels selon les secteurs d'activités. La différence par rapport à MATIÈRE PREMIÈRE réside dans le fait que c'est le PRODUIT qui caractérise le développement et la production de biens et services propres à chaque entreprise.
- Les accords font partie de l'ensemble CONTRAT*
- Un CONTRAT est une convention juridique exécutoire. Cet ensemble d'entités comprend aussi bien les contrats d'assurances, de gestion et de financement que les accords commerciaux, les contrats de consultation, de licences et de ventes.
- L'ensemble TRANSACTION COMMERCIALE se rapporte aux processus*
- Une TRANSACTION COMMERCIALE est une opération exécutée dans le cadre d'un contrat relatif à une affaire déterminée. Il s'agit par exemple d'un paiement, d'une écriture comptable, d'une facturation ou d'une livraison. L'ensemble d'entités TRANSACTION COMMERCIALE enregistre les mouvements dans les ensembles d'entités précédents.
- Découvrir les principaux liens*
- Outre les ensembles d'entités générales PARTENAIRE, MATIÈRE PREMIÈRE, PRODUIT, CONTRAT et TRANSACTION COMMERCIALE, nous devons identifier et définir *les principaux liens entre ces ensembles dans la perspective de l'entreprise*. Ainsi, il faut par

exemple examiner les questions suivantes : Auprès de quel partenaire doit-on acheter telle matière première (*gestion de la chaîne logistique*) ? Qui doit fabriquer tel produit de l'entreprise ? Quelles sont les termes à stipuler au contrat pour un partenaire ou une marchandise ?

Bien entendu, cette ébauche sommaire de l'architecture de données globale d'une entreprise ne permet pas encore de concevoir ni de mettre en œuvre un système d'information. Les ensembles d'entités et de liens doivent être affinés par étapes successives en fonction des secteurs d'activités ou des domaines d'application spécifiques dans l'entreprise. Dans cette approche orientée données, il est primordial que *chaque modèle de données propre à une application particulière soit réalisé en conformité avec l'architecture de données d'entreprise*. C'est la voie unique pour maîtriser le développement des systèmes d'information qui s'alignent sur les objectifs à long terme de l'entreprise.

Conformité à l'architecture de données d'entreprise

2.7 Guide de la construction d'une base de données

Dans cette section, nous récapitulons les connaissances acquises sur la modélisation des données sous la forme d'un guide pratique. La figure 2-24 présente les dix étapes de développement dont la séquence d'exécution diffère selon les phases de réalisation d'un projet. L'expérience pratique nous montre que, durant la phase d'*étude préalable*, l'analyse de données consiste à développer un modèle entité-association sommaire. Dans les phases de *conception sommaire* et de *conception détaillée*, le modèle de données sera affiné, puis traduit en un schéma de base de données relationnelle. On abordera ensuite les questions liées à la cohérence et à l'implantation de la base de données. Étudions de plus près les caractéristiques des dix étapes de développement proposées.

De l'étude préalable à la conception détaillée

Tout d'abord, dans l'analyse de données, il faut commencer par dresser une liste de toutes les *informations factuelles pertinentes*. L'activité de développement étant essentiellement un processus itératif, cette liste sera complétée et affinée en collaboration avec les

Les quatre étapes de l'étude préalable

futurs utilisateurs lors des étapes ultérieures. La seconde étape vise à découvrir les *ensembles d'entités et de liens* et à définir leurs clés d'identification et leurs attributs. On complète le modèle entité-association en y indiquant les différents types d'associations. Dans la troisième étape, on se concentre tout particulièrement sur la construction des hiérarchies de généralisation et des structures d'agrégation. Au cours de la quatrième étape, on confronte le modèle entité-association à l'architecture de données d'entreprise et on apporte au modèle les ajustements nécessaires afin de garantir le développement coordonné des systèmes d'information en harmonie avec les objectifs à long terme de l'entreprise.

Figure 2-24
Du général au
détail en dix étapes
de développement

Les étapes de la construction d'une base de données	Étude préalable	Conception	
		sommaire	détaillée
1. Analyse de données	x	x	x
2. Ensembles d'entités et de liens	x	x	x
3. Généralisation et agrégation	x	x	x
4. Conformité à l'architecture de données globale de l'entreprise	x	x	x
5. Schéma de base de données relationnelle		x	x
6. Normalisation		x	x
7. Intégrité référentielle		x	x
8. Contraintes de cohérence		x	x
9. Chemins d'accès			x
10. Structure de données physique			x

La conception
sommaire produit
un modèle de
données logique

À la cinquième étape, le modèle entité-association sera traduit en un *schéma de base de données relationnelle*. Les règles de passage qui ont été étudiées précédemment seront appliquées aux ensembles d'entités et de liens, aux hiérarchies de généralisation et aux structures

d'agrégation. La sixième étape a pour but d'analyser le schéma de base de données par rapport aux propriétés des *formes normales*. Une investigation approfondie des différents types de dépendances permet de détecter et d'éliminer des incohérences dans le schéma. Dans les septième et huitième étapes, on se focalise sur les *contraintes d'intégrité*. On vérifie tout d'abord les liens entre clés primaires et étrangères par rapport aux règles de l'intégration référentielle concernant les manipulations de données. On définit ensuite d'autres contraintes de cohérence, même si elles ne sont pas toutes prises en charge par un système de bases de données particulier. La spécification des contraintes de cohérence permet de mettre en œuvre des règles de validation particulières au niveau du système, évitant ainsi aux utilisateurs l'obligation de contrôler individuellement l'intégrité des données.

La neuvième étape vise à déterminer les *chemins d'accès* dans les fonctions applicatives majeures. Il faudra analyser ici les attributs les plus fréquemment accédés dans le futur, et les rassembler dans une matrice d'accès. Établie pour l'ensemble des tables du schéma de base de données relationnelle, cette matrice nous renseigne sur le degré auquel les attributs ou combinaisons d'attributs seront sollicités par des opérations d'insertion, de mise à jour et de suppression. La conception physique des ensembles et la *définition de la structure physique des données* sont les préoccupations de la dixième étape. On étudie à ce stade les chemins d'accès physiques et éventuellement les étapes inverses de la normalisation (dénormalisation) en vue d'optimiser la performance des applications futures (voir chapitre 4).

Le guide pratique présenté en figure 2-24 est essentiellement orienté données. Naturellement, les fonctions jouent aussi un rôle important dans le développement des systèmes d'information. C'est pourquoi, au cours des dernières années, parmi les outils CASE sont apparus ceux qui supportent non seulement la conception des bases de données, mais aussi celle des fonctions. Le lecteur qui s'intéresse aux méthodologies de développement des applications trouvera dans la prochaine section une bibliographie commentée à ce sujet.

La conception détaillée produit un modèle de données physique

La conception orientée données se complète par celle des fonctions

2.8 Notes bibliographiques

Apparition du modèle entité-association

Le modèle entité-association fut introduit par les travaux de Senko et Chen (voir Chen 1976). Depuis 1979, des conférences internationales sont régulièrement organisées pour discuter des propositions d'extension et de raffinement du modèle entité-association.

Méthodes de modélisation des données

Pour modéliser les données, la plupart des outils CASE actuels offrent la possibilité de construire des modèles entité-association et de représenter graphiquement les ensembles d'entités et de liens et les types d'associations ; citons par exemple les études réalisées par Balzert (1993), Olle et al. (1988) et Martin (1990). Tsichritzis et Lochovsky (1982) présentent un aperçu d'autres modèles de données logiques.

Analyse et conception orientées objet

Booch (1993), Rumbaugh et al. (1991), Coad et Yourdon (1991) développent la conception orientée objet. Ferstl et Sinz (1991) figurent parmi les auteurs de langue allemande qui proposent l'approche orientée objet dans le développement des systèmes d'information. Balzert (1999) traite de l'analyse orientée objet en combinant les approches méthodologiques de Coad, Booch et Rumbaugh. Stein (1994) présente une étude comparative des méthodes d'analyse orientée objet. Hitz et Kappel (2002) donnent une introduction au langage UML (Unified Modelling Language), axée sur le développement des logiciels.

Généralisation et agrégation

Smith et Smith (1977) ont introduit les concepts de généralisation et d'agrégation dans le domaine des bases de données. Ces structures étaient déjà connues auparavant dans le domaine des systèmes de bases de connaissances, par exemple pour représenter les réseaux sémantiques (voir Findler 1979).

Travaux sur les formes normales

L'étude des formes normales a contribué à l'élaboration d'une véritable théorie des bases de données dans ce domaine. Les travaux de Maier (1983), d'Ullman (1982, 1988) et de Paredaens et al. (1989) figurent parmi les ouvrages théoriques de référence sur les formes normales. Dutka et Hanson (1989), Reingruber et Gregory (1994), Simsion et Witt (2005) développent les formes normales de manière

étendue dans une présentation concise et instructive. Les travaux devenus classiques de Date (2004), d'Elmasri et Navathe (2004), de Kemper et Eickler (2001), de Silberschatz et al. (2005) consacrent une large place à la normalisation.

Les questions touchant à l'architecture de données d'entreprise sont traitées par Dippold et al. (2001), Meier et al. (1991) et Scheer (1991). Meier et Johner (1991), Ortner et al. (1990) définissent les tâches et les responsabilités dans l'administration et la modélisation des données. Silverston (2001a, 2001b) rassemble en deux volumes une collection de modèles de données génériques par types d'entreprises et d'industries.

Architecture de données d'entreprise

Les aspects liés à la modélisation des données sont traités en profondeur dans Dürr et Radermacher (1990), Nanci et Espinasse (2001), Reingruber et Gregory (1994), Schlageter et Stucky (1983), Simsion et Witt (2005), Vossen (2000) et Vetter (1998).

Modélisation des données

3 Langages de requête et de manipulation des données

3.1 Exploitation d'une base de données

Le succès de l'exploitation d'une base de données repose sur l'existence d'un langage pour bases de données, capable de satisfaire les besoins variés des utilisateurs. Les langages relationnels d'interrogation et de manipulation de données offrent un avantage déterminant (voir figure 3-1) : un et un seul langage permet de mettre en œuvre une base de données, de gérer les autorisations d'accès, d'interroger et de mettre à jour les tables.

Nécessité d'un langage pour bases de données

Chargé de la gestion des définitions de tables et d'attributs pour l'entreprise, *l'administrateur de données* utilise un langage relationnel de base de données pour accomplir ses tâches, assisté notamment par un système de dictionnaire de données. En collaboration avec *l'architecte de données*, il doit veiller à l'unicité et la cohérence des définitions de données en conformité avec l'architecture de données globale de l'entreprise et en assurer la maintenance avec l'aide éventuelle d'un outil CASE adéquat. Au-delà du contrôle des formats de données, il est chargé de gérer les autorisations pour restreindre l'usage des données à certaines tables voire à certains attributs précis d'une part, et d'autre part, pour limiter une opération déterminée, telle que la suppression ou la mise à jour d'une table, à un groupe d'utilisateurs bien défini.

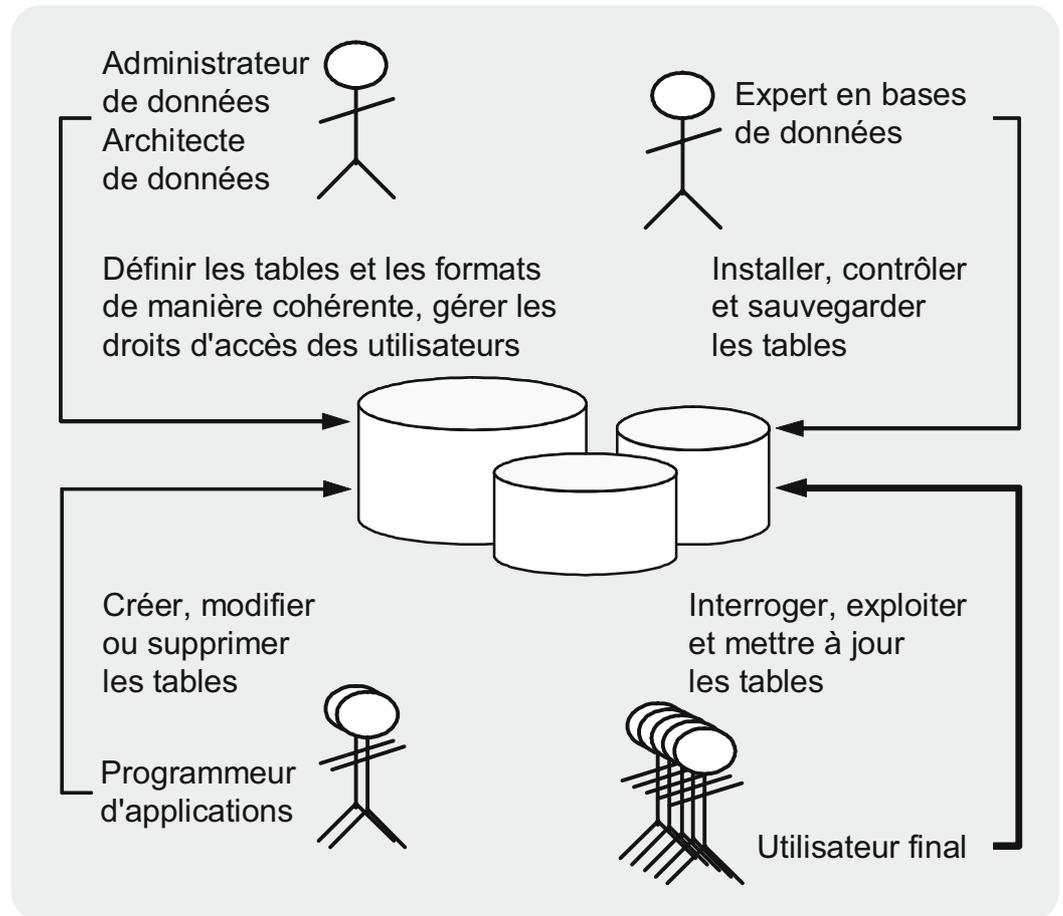
L'administrateur et l'architecte de données définissent les tables et les formats de données

L'expert en bases de données utilise, dans la mesure du possible, le même langage pour définir, installer et contrôler les bases de données à l'aide de tables systèmes prévues à ces fins. Ces tables constituent le catalogue du système qui contient toutes les descriptions d'une base et les données statistiques nécessaires à son exploitation. L'expert en bases de données formule des requêtes d'interrogation pour extraire des informations du système qui lui

L'expert en bases de données installe et contrôle les bases de données

donnent une image actuelle de l'ensemble des bases de données, sans se préoccuper des tables contenant les données des utilisateurs. En vertu de la protection des données, l'accès aux bases de données en cours d'exploitation ne devrait lui être accordé qu'à titre exceptionnel, par exemple pour la correction des erreurs.

Figure 3-1
Usage d'un langage pour bases de données à des fins variées



Le programmeur d'application développe les systèmes d'information

Le programmeur d'application utilise le langage de base de données pour exploiter et mettre à jour une base de données. Puisqu'un langage relationnel d'interrogation et de manipulation de données est de nature ensembliste, le programmeur doit saisir l'importance de la *notion de curseur* (voir la section 3.5). Le curseur permet à un programme de traiter un ensemble de tuples enregistrement par enregistrement. Le langage relationnel est également utile au programmeur pour tester les applications. C'est en effet un moyen commode pour vérifier les bases de données de test qui seront rapidement livrées aux futurs utilisateurs sous forme de prototypes.

Enfin, un langage relationnel permet à *l'utilisateur final* de satisfaire ses *besoins d'informations au quotidien*. Par utilisateur final, nous entendons l'ensemble des utilisateurs dans des départements fonctionnels de l'entreprise. Avec des connaissances limitées en informatique, ils désirent effectuer eux-mêmes certaines analyses de manière spontanée dans leur domaine d'activité.

L'utilisateur final effectue l'analyse de données

Nous venons de montrer qu'un langage relationnel pour bases de données répond aux besoins de différentes catégories d'utilisateurs. Depuis l'administrateur de base de données jusqu'à l'utilisateur final en passant par l'expert en bases de données et le développeur d'applications, tous utilisent le même langage pour accomplir leurs tâches. Ainsi, les applications de bases de données, les analyses de données, ainsi que les opérations techniques pour assurer la sécurité ou réorganiser des bases de données reposent sur un langage unique. L'uniformité du langage contribue à la réduction du coût de formation, et favorise en outre l'échange d'expériences entre divers groupes d'utilisateurs.

Un langage de base de données pour répondre à des besoins variés

3.2 Les bases de l'algèbre relationnelle

3.2.1 Vue d'ensemble des opérateurs

L'*algèbre relationnelle* (*relational algebra*, en anglais) définit le *cadre formel des langages relationnels pour bases de données*. Elle introduit une collection d'opérateurs algébriques qui s'appliquent toujours à des tables. De nos jours, la plupart des langages relationnels pour bases de données n'utilisent pas directement ces opérateurs. Dans le cadre du modèle relationnel, nous les qualifions de langages relationnels complets seulement lorsqu'ils offrent toutes les possibilités originelles de l'algèbre relationnelle.

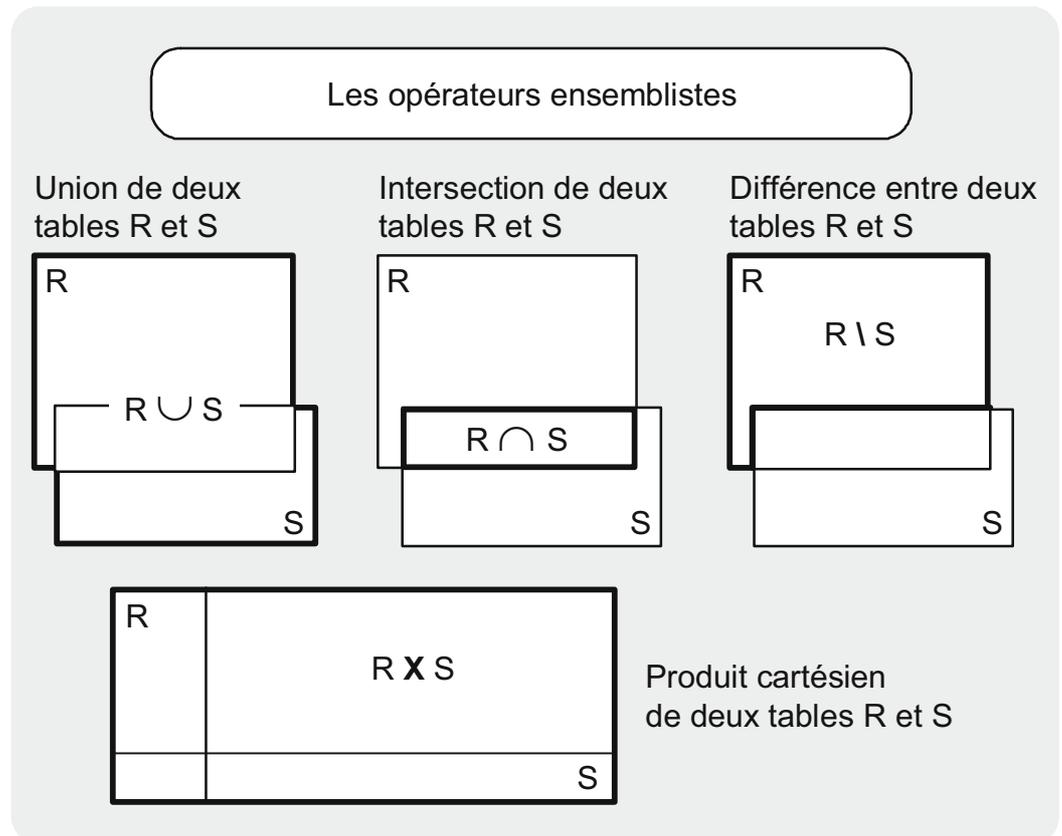
L'algèbre relationnelle constitue le fondement des langages relationnels

Nous présentons maintenant une vue d'ensemble des opérateurs de l'algèbre relationnelle en les appliquant à deux tables, R et S. Ils sont classés en deux catégories : les opérateurs *ensemblistes* et les opérateurs *relationnels*. Tous les opérateurs portent sur une ou deux tables et produisent une nouvelle table résultat. Cette uniformité

Les opérateurs peuvent être combinés

(propriété algébrique) permet d'appliquer aux tables des combinaisons d'opérateurs.

Figure 3-2
Opérations sur les tables : union, intersection, différence et produit cartésien



Déterminer la somme, la différence et le produit des tables

Les *opérateurs ensemblistes* correspondent aux opérations traditionnelles sur les ensembles (figure 3-2 et section 3.2.2) : l'union, désignée par le symbole « \cup », l'intersection « \cap », la différence « \setminus » et le produit cartésien « \times ». En vertu d'un critère de compatibilité qui sera défini dans la section 3.2.2, on peut déterminer l'union de deux tables R et S ($R \cup S$), leur intersection ($R \cap S$) et leur différence ($R \setminus S$) ; enfin, le produit de deux tables quelconques R et S peut s'effectuer sans aucune condition préalable ($R \times S$). Le résultat de ces opérations est lui-même un ensemble de tuples, c'est-à-dire une table.

Réduire ou combiner les tables

Selon la figure 3-3, les *opérateurs relationnels* portent sur des tables. Ils seront expliqués en détail dans la section 3.2.3. L'opérateur de projection, symbolisé par le caractère grec π (Pi), génère une sous-table à partir de la table traitée. Par exemple, l'expression $\pi_M(R)$ produit, à partir de la table R, une sous-table contenant un ensemble d'attributs M. L'opérateur de sélection $\sigma_F(R)$, symbolisé par le caractère grec σ (Sigma), extrait des tuples de la table R d'après un

nous puissions calculer l'union, l'intersection ou la différence de deux tables, elles doivent être *compatibles avec l'union*.

Compatibilité avec l'union

Deux tables sont compatibles avec l'union si elles satisfont les deux conditions suivantes : primo, elles contiennent le même nombre d'attributs ; secundo, les attributs correspondants ont un format de données identique.

Figure 3-4
Les tables compatibles avec l'union, CLUB DE SPORT et CLUB DE PHOTO

CLUB DE SPORT			
E#	Nom	Rue	Ville de domicile
E1	Meier	rue Faucigny	Fribourg
E7	Humbert	route des Alpes	Bulle
E19	Savoy	avenue de la Gare	Romont
....			

CLUB DE PHOTO			
E#	Membre	Rue	Ville
E4	Brodard	rue du Tilleul	Fribourg
E7	Humbert	route des Alpes	Bulle

Exemple de tables compatibles avec l'union

Un exemple est donné à la figure 3-4 : à partir d'un fichier des employés d'une entreprise, on a créé, pour chacun de ses deux clubs internes, une table dont les attributs sont le numéro, le nom et l'adresse de l'employé. Quoique certains attributs portent des noms différents, les deux tables CLUB DE SPORT et CLUB DE PHOTO sont compatibles avec l'union. En effet, elles présentent le même nombre d'attributs d'une part, et d'autre part les valeurs d'attributs appartiennent aux mêmes domaines puisqu'elles proviennent d'un seul fichier des employés.

Opérateur d'union

En référence à la théorie des ensembles, *l'union* (*union*, en anglais) $R \cup S$ permet de combiner deux tables R et S compatibles avec l'union. Cette opération *insère dans la table résultat toutes les occurrences de R et toutes celles de S*. Les enregistrements de données identiques sont éliminés de sorte que nous ne pouvons plus les différencier dans l'ensemble résultat $R \cup S$ sur la base des valeurs d'attributs.

La table MEMBRE DE CLUB (Figure 3-5) résulte de l'union des tables CLUB DE SPORT et CLUB DE PHOTO. Chaque tuple dans la table résultat est présent soit dans la table CLUB DE SPORT soit dans CLUB DE PHOTO soit dans les deux à la fois. L'un des membres, Humbert, n'apparaît qu'une seule fois, car la table résultant de l'union n'admet pas d'entrées identiques.

MEMBRE DE CLUB = CLUB DE SPORT \cup CLUB DE PHOTO

E#	Nom	Rue	Ville
E1	Meier	rue Faucigny	Fribourg
E7	Humbert	route des Alpes	Bulle
E19	Savoy	avenue de la Gare	Romont
E4	Brodard	rue du Tilleul	Fribourg

Figure 3-5
Union des deux
tables CLUB DE
SPORT et CLUB DE
PHOTO

Les autres opérateurs ensemblistes sont définis de manière analogue : *l'intersection* (*intersection*, en anglais) $R \cap S$ combine deux tables R et S compatibles avec l'union. Cette opération insère dans la table résultat seulement les occurrences *présentes à la fois dans R et dans S*. Parmi les tuples que nous avons extraits des tables CLUB DE SPORT et CLUB DE PHOTO, Humbert est l'unique membre actif dans deux clubs à la fois. Par conséquent, la table résultant de CLUB DE SPORT \cap CLUB DE PHOTO contient un seul élément, car il n'existe qu'un employé ayant un double statut de membre.

Opérateur
d'intersection

Considérons enfin la différence entre deux tables compatibles avec l'union. La *différence* (*difference*, en anglais) $R \setminus S$ s'obtient *en éliminant de R toutes les occurrences qui sont aussi présentes dans S*. Appliquée à notre exemple, la différence CLUB DE SPORT \setminus CLUB DE PHOTO produit une relation qui contient seulement deux membres, Meier et Savoy. Le membre Humbert est éliminé, car il est aussi membre du CLUB DE PHOTO. Ainsi, l'opérateur de différence permet de déterminer les membres du CLUB DE SPORT qui n'appartiennent pas simultanément au CLUB DE PHOTO.

Opérateur de
différence

Rapport entre les opérateurs d'intersection et de différence

En principe, le rapport suivant existe entre les opérateurs d'intersection et de différence de deux tables compatibles avec l'union :

$$R \cap S = R \setminus (R \setminus S).$$

Par conséquent, *l'intersection est une opération qui peut être ramenée à la différence entre des tables*. Le lecteur peut le vérifier facilement dans notre exemple des clubs de sport et de photo.

Produit cartésien de deux tables

Parmi les opérateurs d'ensembles il nous reste à définir le produit cartésien de deux tables quelconques, R et S, qui ne doivent pas nécessairement être compatibles avec l'union. Le *produit cartésien (cartesian product, en anglais) R×S* de deux tables R et S est *l'ensemble de toutes les combinaisons possibles des tuples de R avec ceux de S*.

*Figure 3-6
La table TOURNOI,
un exemple de
produit cartésien*

TOURNOI = (CLUB DE SPORT \ CLUB DE PHOTO) x CLUB DE PHOTO

E#	Nom	Rue	Ville de domicile	E#	Membre	Rue	Ville
E1	Meier	rue Faucigny	Fribourg	E4	Brodard	rue du Tilleul	Fribourg
E1	Meier	rue Faucigny	Fribourg	E7	Humbert	route des Alpes	Bulle
E19	Savoy	avenue de la Gare	Romont	E4	Brodard	rue du Tilleul	Fribourg
E19	Savoy	avenue de la Gare	Romont	E7	Humbert	route des Alpes	Bulle

Exemple d'une table pour organiser un tournoi

À titre d'exemple, considérons la table TOURNOI dans la figure 3-6, qui contient une combinaison de membres obtenue par l'opération (CLUB DE SPORT \ CLUB DE PHOTO) × CLUB DE PHOTO. La table TOURNOI contient toutes les combinaisons possibles des membres du club de sport (qui ne sont pas dans le club de photo) avec ceux du club de photo. Elle présente une liste typique des rencontres opposant les membres de deux clubs dans un tournoi. Ainsi par exemple, Humbert, membre de deux clubs à la fois, ne peut pas évidemment jouer contre lui-même. En effectuant la différence CLUB DE SPORT \ CLUB DE PHOTO, nous décidons qu'il jouera pour le club de photo.

Produit des tables

C'est une opération appelée produit (cartésien), car le nombre d'entrées dans la table résultat s'obtient en multipliant les nombres de

tuples dans les tables de départ. Soient m le nombre d'entrées de la table R et n celui de la table S . Le produit cartésien $R \times S$ contient m fois n tuples au total, à condition qu'aucun tuple ne soit présent dans les deux tables à la fois.

3.2.3 Les opérateurs relationnels

Après les opérateurs ensemblistes, nous poursuivons notre étude en abordant maintenant les opérateurs relationnels. Comme pour le produit cartésien, les tables ne doivent pas nécessairement être compatibles avec l'union. L'opérateur de projection (projection operator, en anglais) $\pi_M(R)$ construit, à partir de la table R , une sous-table dont les noms d'attributs sont définis dans M . À titre d'exemple, considérons une table R avec les attributs (A,B,C,D) . L'expression $\pi_{A,C}(R)$ signifie que R sera réduite à deux colonnes A et C . Dans une projection, les noms de colonnes peuvent apparaître dans un ordre quelconque. Ainsi, $R' := \pi_{C,A}(R)$ est la projection de la table $R = (A,B,C,D)$ sur $R' = (C,A)$.

Opérateur de projection

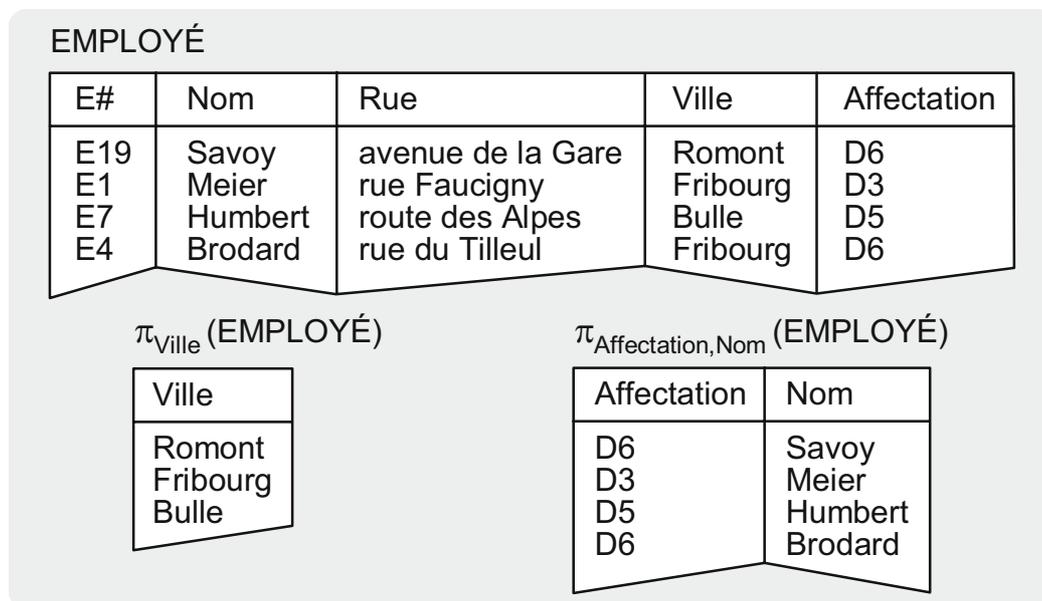


Figure 3-7
Opérateur de projection appliqué à la table EMPLOYÉ

Dans la figure 3-7, la première projection $\pi_{\text{Ville}}(\text{EMPLOYÉ})$ produit une table dont l'unique colonne contient, sans duplication, toutes les villes extraites de la table des employés. La seconde projection $\pi_{\text{Affectation,Nom}}(\text{EMPLOYÉ})$ génère une sous-table

Exemples de projection

contenant tous les numéros de département et les noms des employés qui y travaillent.

Opérateur de sélection

Un deuxième opérateur important $\sigma_F(R)$ permet la *sélection* (*selection*, en anglais) des tuples d'une table R d'après une formule F . La formule F contient un nombre déterminé de noms d'attributs ou de constantes liés entre eux par des opérateurs de comparaison tels que «<», «>» ou «=», ou par des opérateurs logiques AND, OR et NOT. Par conséquent, $\sigma_F(R)$ nous donne comme résultat des tuples de R qui satisfont la condition de sélection F .

Figure 3-8
Exemples
d'application de
l'opérateur de
sélection

$\sigma_{\text{Ville}=\text{Fribourg}}$ (EMPLOYÉ)				
E#	Nom	Rue	Ville	Affectation
E1	Meier	rue Faucigny	Fribourg	D3
E4	Brodard	rue du Tilleul	Fribourg	D6

$\sigma_{\text{Affectation}=\text{D6}}$ (EMPLOYÉ)				
E#	Nom	Rue	Ville	Affectation
E19	Savoy	avenue de la Gare	Romont	D6
E4	Brodard	rue du Tilleul	Fribourg	D6

$\sigma_{\text{Ville}=\text{Fribourg AND Affectation}=\text{D6}}$ (EMPLOYÉ)				
E#	Nom	Rue	Ville	Affectation
E4	Brodard	rue du Tilleul	Fribourg	D6

Exemples de sélection

La figure 3-8 montre des exemples de sélection de tuples depuis la table EMPLOYÉ. Le premier consiste à sélectionner tous les employés qui remplissent la condition «Ville = Fribourg», c'est-à-dire qui habitent à Fribourg. Le deuxième exemple, avec la condition «Affectation = D6», sélectionne uniquement les employés qui travaillent au département D6. Enfin, le troisième exemple combine les deux critères de sélection précédents par l'opérateur logique AND dans la formule «Ville = Fribourg AND Affectation = D6». La table qui en résulte contient un élément unique, car Brodard est le seul employé qui vienne de Fribourg et qui, en même temps, travaille au département D6.

Il est naturellement possible de combiner les opérateurs de l'algèbre relationnelle que nous venons d'étudier. Par exemple, si, après avoir extrait tous les employés du département D6 par la sélection $\sigma_{\text{Affectation}=\text{D6}}(\text{EMPLOYÉ})$, nous appliquons une projection de la table résultante sur l'attribut Ville par l'opérateur $\pi_{\text{Ville}}(\sigma_{\text{Affectation}=\text{D6}}(\text{EMPLOYÉ}))$, le résultat final sera une table contenant deux villes, Romont et Fribourg.

Il est possible de combiner les opérateurs

Considérons maintenant *l'opérateur de jointure (join operator, en anglais)* qui permet de composer deux tables en une seule. La jointure $R \bowtie_P S$ de deux tables R et S d'après le prédicat P est une *combinaison de tous les tuples de R avec ceux de S, qui satisfont le prédicat de jointure P*. Le prédicat de jointure contient un attribut de la table R et un attribut de S. Ces deux attributs sont liés par des opérateurs de comparaison, «<», «>» ou «=», définissant ainsi le critère de combinaison des tables R et S. Si le prédicat de jointure P contient l'opérateur de comparaison «=», on parle d'une *équi-jointure (equi-join, en anglais)*.

Opérateur de jointure (équi-jointure)

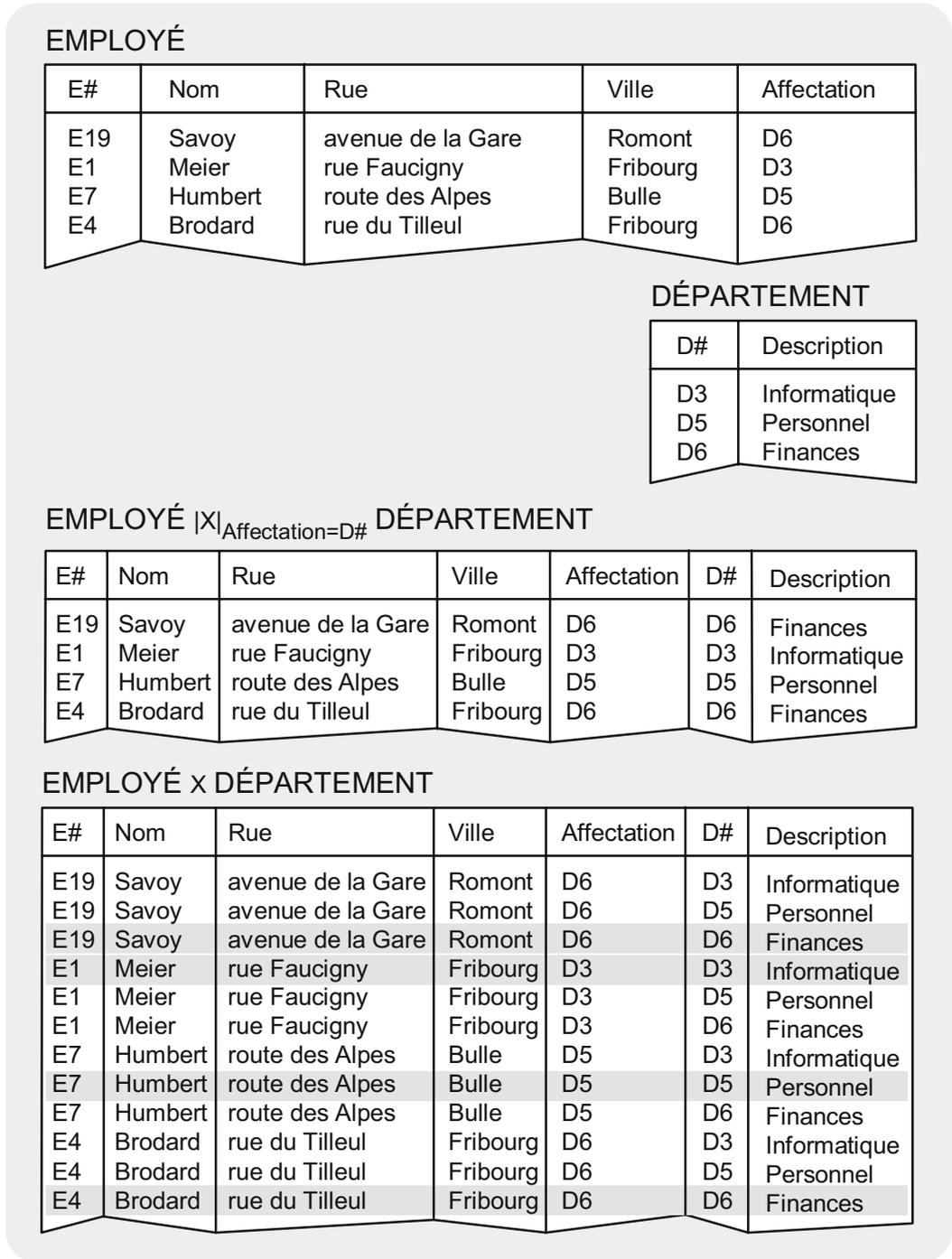
On éprouve souvent des difficultés à comprendre l'opérateur de jointure qui risque par conséquent de fournir des résultats erronés ou indésirés. L'oubli ou la définition incorrecte d'un prédicat pour combiner deux tables en sont généralement la cause.

La jointure demande des précautions !

La figure 3-9 donne deux exemples de jointure, le prédicat de jointure étant présent dans le premier et absent dans le second. En spécifiant $\text{EMPLOYÉ} \bowtie_{\text{Affectation}=\text{D}\#} \text{DÉPARTEMENT}$, nous joignons les deux tables EMPLOYÉ et DÉPARTEMENT pour compléter les données de chaque employé par des informations sur son département. Dans la seconde jointure où nous omettons le prédicat de jointure P en spécifiant seulement $\text{EMPLOYÉ} \times \text{DÉPARTEMENT}$, le résultat est un produit cartésien des tables EMPLOYÉ et DÉPARTEMENT. Cette combinaison des deux tables n'a pas beaucoup de sens, car elle joint tous les employés à tous les départements. Ainsi, nous constatons que la table résultat contient également des employés combinés avec des départements auxquels ils ne sont pas affectés (voir la table TOURNOI dans la figure 3-6 à titre de comparaison).

La définition d'une jointure demande des précautions

Figure 3-9
Jointure de deux
tables avec et sans
prédicat de jointure



La jointure vue
comme un produit
cartésien restreint

Comme les exemples à la figure 3-9 le montrent, l'opérateur de jointure \times avec un prédicat de jointure P n'est rien d'autre qu'une restriction du produit cartésien. En principe, une jointure de deux tables R et S sans le prédicat de jointure équivaut à un produit cartésien de R et S. En d'autres termes, un prédicat vide, $P=\{\}$, entraîne l'égalité suivante :

$$R \times_{P=\{\}} S = R \times S.$$

Si, dans une sélection, nous utilisons un prédicat de jointure comme prédicat de sélection, nous obtenons l'équation suivante :

$$R \bowtie_P S = \sigma_P(R \times S).$$

Selon cette formule générale, chaque jointure équivaut donc à une suite de deux opérations : la première effectue un produit cartésien et la seconde une sélection.

Considérons l'exemple de la figure 3-9. La jointure désirée EMPLOYÉ $\bowtie_{\text{Affectation=D\#}}$ DÉPARTEMENT peut être calculée en deux étapes : d'abord, nous évaluons le produit cartésien des tables EMPLOYÉ et DÉPARTEMENT ; ensuite, nous effectuons la sélection $\sigma_{\text{Affectation=D\#}}(\text{EMPLOYÉ} \times \text{DÉPARTEMENT})$ sur la table résultat intermédiaire pour en extraire les tuples qui satisfont le prédicat de jointure «Affectation=D#». Le résultat final contient les mêmes tuples que ceux obtenus par évaluation directe de la jointure EMPLOYÉ $\bowtie_{\text{Affectation=D\#}}$ DÉPARTEMENT (voir les tuples grisés, c'est-à-dire sélectionnés, dans la figure 3-9).

Les deux étapes de calcul d'une jointure

La *division* (*division*, en anglais) d'une table R par une table S est possible à condition que S soit une sous-table contenue dans R. L'opérateur de division $R \div S$ produit une sous-table R' à partir de R, telle que toutes les combinaisons de tuples r' dans R' avec les tuples s dans S sont contenues dans la table R. En d'autres termes, le produit cartésien R' \times S doit être contenu dans la table R.

Opérateur de division

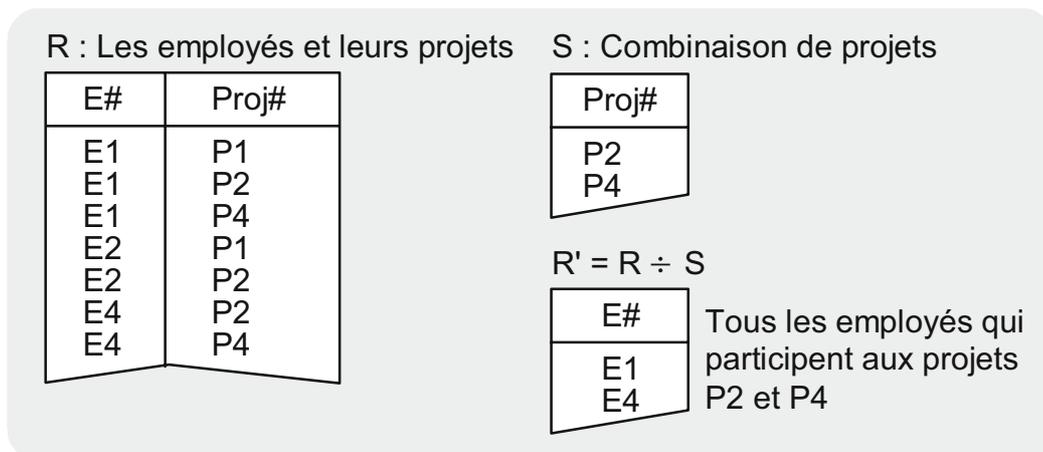


Figure 3-10 Exemple d'application de l'opérateur de division

*Un exemple
de division*

Dans la figure 3-10, la table R nous montre la participation des employés aux divers projets. Nous désirons maintenant connaître les employés qui travaillent simultanément sur les projets P2 et P4. À cet effet, nous créons une table S contenant les numéros de projet P2 et P4. Il est évident que S est contenu dans R. Par conséquent, la division $R' := R \div S$ est possible et nous donne comme résultat la table R' contenant les employés E1 et E4. Une rapide vérification nous confirme que E1 et E4 travaillent sur les projets P2 et P4 en même temps, car les tuples (E1,P2) et (E1,P4), ainsi que (E4,P2) et (E4,P4), existent dans la table R.

On peut exprimer l'opérateur de division en fonction des opérateurs de projection, de différence et du produit cartésien. Pour cette raison, en algèbre relationnelle il figure parmi les opérateurs substituables qui comprennent aussi les opérateurs d'intersection et de jointure.

*Ensemble minimal
d'opérateurs*

En résumé, l'union, la différence, le produit cartésien, la projection et la sélection constituent l'ensemble minimal des opérateurs de base qui offrent toutes les fonctionnalités de l'algèbre relationnelle. Les opérateurs d'intersection, de jointure et de division peuvent être dérivés de ces cinq opérateurs de base, tout en admettant que la substitution est parfois complexe.

*Conception des
ordinateurs de
bases de données*

Les opérateurs de l'algèbre relationnelle ne présentent pas seulement un intérêt sur le plan théorique. Leur portée pratique est aussi importante. Ainsi, nous en aurons besoin pour optimiser les requêtes au niveau du langage des systèmes de bases de données relationnelles (voir 4.2.2). En outre, ils trouvent leur application dans la conception des ordinateurs de bases de données : au lieu d'être mis en œuvre sous forme logicielle, les opérateurs de l'algèbre relationnelle et leurs formes dérivées sont implantés directement dans des composants matériels de l'ordinateur.

3.3 Les langages relationnels complets

Les langages relationnels complets sont des langages qui intègrent au moins les opérations prévues dans l'algèbre relationnelle, ou dans le calcul dit relationnel qui se révèle aussi puissant que l'algèbre relationnelle introduite dans la section précédente.

Que signifie la complétude relationnelle ?

Le *calcul relationnel* (*relational calculus*, en anglais) est fondé sur la logique des prédicats. Il permet de former des expressions qui traduisent une condition de sélection définie par l'utilisateur (appelée prédicat). Les prédicats portent sur une variable tuple d'une table donnée, et peuvent se combiner entre eux à l'aide des opérateurs logiques AND, OR et NOT. En outre, nous pouvons introduire des quantificateurs, par exemple, «pour tout ...» (quantificateur «quel que soit») ou «il existe ...» (quantificateur d'existence). Avec le quantificateur «quel que soit», tous les tuples d'une table doivent vérifier le prédicat correspondant. Le quantificateur d'existence examine si la table contient au moins un tuple satisfaisant le prédicat.

Le calcul relationnel

Les langages utilisés en pratique dans les systèmes de bases de données relationnelles s'appuient sur l'algèbre relationnelle ou le calcul relationnel :

Par exemple, le langage SQL (Structured Query Language) dont nous avons déjà parlé est considéré comme un mélange de l'algèbre relationnelle et du calcul relationnel (voir section 3.4.1).

SQL normalisé

Le langage QUEL (Query Language) illustre l'approche du calcul relationnel de tuples (3.4.2).

Langage QUEL

QBE (Query by Example) est un langage qui permet de formuler des requêtes et d'effectuer des manipulations de données à l'aide d'une interface graphique (3.4.3). QBE s'appuie aussi sur le calcul relationnel tout en offrant aux utilisateurs la convivialité de l'approche graphique pour travailler avec les tables.

Langage graphique QBE

SQL, QUEL et QBE sont des langages dont la puissance d'expression est fondée sur l'algèbre relationnelle et le calcul

relationnel. Ils sont donc considérés à ce titre comme langages relationnels complets.

La *complétude relationnelle d'un langage de base de données* signifie qu'on doit pouvoir exprimer les opérateurs de l'algèbre relationnelle (ou les opérateurs équivalents du calcul relationnel) dans un tel langage.

Critère de complétude

Langages relationnels complets

Un langage d'interrogation de bases de données est dit *relationnel complet* au sens de l'algèbre relationnelle lorsqu'il permet *au moins* l'emploi *des opérateurs ensemblistes*, l'union, la différence et le produit cartésien, *et des opérateurs relationnels*, la projection et la sélection.

Qualité requise d'un langage relationnel

Le critère de complétude est déterminant pour vérifier si un langage de bases de données est relationnel ou non. En effet, un langage qui permet de manipuler des tables n'est pas forcément relationnel complet. Par exemple, si nous ne pouvons pas combiner plusieurs tables par leurs attributs communs, le langage utilisé n'est pas équivalent à l'algèbre relationnelle ou au calcul relationnel. Nous ne pouvons donc pas le qualifier de langage de base de données relationnel complet.

Extension du langage relationnel

Dans un langage relationnel de base de données, l'algèbre relationnelle et le calcul relationnel constituent les fondements du module d'interrogation. Naturellement, outre les opérations d'analyse de données, nous désirons aussi manipuler des tables ou des parties de tables. Nous entendons par là la possibilité d'insérer, de supprimer ou de modifier des ensembles de tuples. C'est pourquoi, pour qu'ils soient utiles en pratique, une extension des langages relationnels complets s'impose en y incluant les fonctionnalités suivantes :

Fonctionnalités additionnelles des langages relationnels

- Le langage doit permettre de créer des tables, d'effectuer des opérations d'insertion, de modification et de suppression.
- Le langage doit inclure des fonctions d'agrégation, permettant de calculer par exemple la somme, le maximum, le minimum ou la moyenne des valeurs dans une colonne d'une table.

- Le langage doit permettre de formater et de présenter les tables d'après différents critères, tels que la séquence de tri ou les ruptures de séquence par groupes.
- Les langages de bases de données relationnelles doivent absolument comporter des éléments pour gérer les autorisations d'accès et pour assurer la protection des bases de données (voir 3.7).
- Les langages de bases de données relationnelles doivent prendre en considération l'environnement multiutilisateur et disposer de commandes pour garantir la sécurité des données.
- Il est avantageux de disposer d'un langage de base de données relationnelle qui prend en charge des expressions ou des calculs arithmétiques.

Nous venons de définir le cadre formel des langages de bases de données relationnelles à travers la présentation de l'algèbre relationnelle et du calcul relationnel. Dans la pratique, ces langages formels ne sont pas directement utilisés comme tel. Dès le début, l'accent porte plutôt sur la conception de *langages relationnels conviviaux pour bases de données*. Puisque la manipulation des opérateurs algébriques dans leur forme brute ne peut pas être imposée à l'utilisateur d'un système de bases de données, il faut substituer ces opérateurs par un langage de bases de données qui soit expressif, intuitif à l'usage. SQL, QUEL et QBE, présentés dans les sections suivantes, sont trois exemples de langages qui permettent d'atteindre cet objectif.

*Importance de
l'interface
utilisateur*

3.4 Aperçu des langages relationnels

3.4.1 SQL

Au milieu des années 1970, le langage SQL fut créé pour un système expérimental appelé «System R». C'était un des premiers systèmes de bases de données relationnelles qui soient opérationnels. Plus tard, SQL a été normalisé par l'ISO, et devient aujourd'hui le plus important langage dans ce domaine.

*SQL, langage
normalisé par l'ISO*

Comme nous l'avons déjà présentée dans la section 1.2, la structure de base du langage de requête SQL est la suivante :

<i>Structure de base de SQL</i>	SELECT	Attributs
	FROM	Tables
	WHERE	Prédicat de sélection

La clause SELECT correspond à l'opérateur de projection en algèbre relationnelle et permet de déclarer une liste d'attributs. En nous reportant à l'exemple en figure 3-7, la requête $\pi_{\text{Affectation, Nom}}(\text{EMPLOYÉ})$, formulée à l'aide d'un opérateur de projection, équivaut en SQL à :

<i>Projection</i>	SELECT	Affectation, Nom
	FROM	EMPLOYE

Dans la clause FROM, nous déclarons toutes les tables nécessaires à l'exécution de la requête. Par exemple, le produit cartésien des tables EMPLOYÉ et DÉPARTEMENT se traduit en SQL par :

<i>Produit cartésien</i>	SELECT	E#, Nom, Rue, Ville, Affectation, D#, Designation
	FROM	EMPLOYE, DEPARTEMENT

Cette commande produit une table résultat identique à celle qui a été générée, dans la figure 3-9, par les opérateurs équivalents $\text{EMPLOYÉ} \times_{\text{P}=\{\}} \text{DÉPARTEMENT}$ ou $\text{EMPLOYÉ} \times \text{DÉPARTEMENT}$.

Si nous formulons le prédicat de jointure «Affectation = D#» dans la clause WHERE, la commande SQL précédente devient une équi-jointure des tables EMPLOYÉ et DÉPARTEMENT :

<i>Jointure</i>	SELECT	E#, Nom, Rue, Ville, Affectation, D#, Designation
	FROM	EMPLOYE, DEPARTEMENT
	WHERE	Affectation = D#

Nous pouvons affiner une requête de sélection en spécifiant, dans la clause WHERE, plusieurs conditions de recherche liées par les opérateurs logiques AND et OR. À titre d'exemple, la sélection des

employés définie par $\sigma_{\text{Ville}=\text{Fribourg} \text{ AND } \text{Affectation}=\text{D6}}(\text{EMPLOYÉ})$ dans la figure 3-8, correspond en SQL à :

```
SELECT      *
FROM        EMPLOYE
WHERE       Ville = 'Fribourg' AND Affectation = 'D6'
```

Sélection

Un «*» dans la clause SELECT signifie que tous les attributs de la table considérée seront sélectionnés ; par conséquent, nous obtiendrons une table résultat contenant les attributs E#, Nom, Rue, Ville et Affectation. La clause WHERE contient le prédicat de sélection désirée. L'exécution de la requête ci-dessus extrait de la base de données l'employé Brodard de Fribourg, travaillant au département D6.

En plus des opérateurs habituellement reconnus de l'algèbre relationnelle et du calcul relationnel, il existe en SQL des *fonctions prédéfinies ou intégrées (built-in functions, en anglais)* qui peuvent être appelées dans la clause SELECT. Elles s'opèrent sur une colonne déterminée dans une table : COUNT pour compter le nombre de valeurs dans une colonne, SUM pour faire la somme, AVG pour calculer la moyenne (*average, en anglais*), MAX pour déterminer la valeur maximale et MIN la valeur minimale.

*Fonctions SQL
prédéfinies*

Par exemple, l'instruction SQL suivante permet de calculer l'effectif du département D6 :

```
SELECT      COUNT (E#)
FROM        EMPLOYE
WHERE       Affectation = 'D6'
```

Le résultat est une table à un seul élément qui prend la valeur 2. L'examen de la table EMPLOYÉ montre qu'il s'agit bien de l'effectif du département D6 où travaillent les employés Savoy et Brodard.

La commande SQL CREATE permet de définir une table. Elle s'écrit comme suit pour créer par exemple la table EMPLOYÉ :

Définition de tables

```
CREATE TABLE EMPLOYE
(E#      CHAR(6) NOT NULL,
Nom     CHAR(20),
... )
```

Supprimer une table avec précaution

L'opération inverse consiste à supprimer la définition d'une table par la commande `DROP TABLE`. Il convient de noter que cette commande supprime en même temps son contenu tout entier et les droits d'accès des utilisateurs à la table considérée (voir 3.7).

Une fois la table EMPLOYÉ créée, la commande

```
INSERT INTO EMPLOYE
VALUES ('E20', 'Morel', 'chemin du Cerisier', 'Marly', 'D6')
```

permet d'y insérer de nouveaux tuples, c'est-à-dire de nouvelles lignes.

Mise à jour des données

Pour manipuler la table EMPLOYÉ, nous utilisons la commande `UPDATE` :

```
UPDATE EMPLOYE
SET Ville = 'Friburgo'
WHERE Ville = 'Fribourg'
```

Cette commande repère dans la table EMPLOYÉ tous les tuples dont l'attribut Ville a la valeur Fribourg, et la remplace par la nouvelle valeur, Friburgo. La mise à jour par la commande `UPDATE` est une opération ensembliste, elle s'opère donc sur un ensemble de tuples.

Suppression de données

Enfin, le contenu entier ou partiel d'une table est supprimé par la commande `DELETE` :

```
DELETE
FROM EMPLOYE
WHERE Ville = 'Friburgo'
```

La commande de suppression `DELETE` affecte un ensemble de tuples si plusieurs entrées dans la table considérée vérifient le prédicat de sélection. Lorsque l'intégrité référentielle (voir 3.8) doit être

respectée, la suppression peut également affecter les tables dépendantes.

3.4.2 QUEL

Le langage QUEL qui repose sur le calcul relationnel de tuples a été conçu pour le système de bases de données «Ingres» au milieu des années 1970. Chaque requête de recherche s'exprime selon le schéma de base suivant :

QUEL requiert une variable tuple

RANGE OF	Variable tuple IS Table
RETRIEVE	(Attribut, ...)
WHERE	Prédicat de sélection

Chaque variable tuple, symbolisée par une lettre minuscule, est déclarée dans une clause distincte RANGE OF où l'on doit spécifier le nom de la table qui lui est associée. Inversement, il faut utiliser une variable tuple pour chaque table impliquée dans la sélection. La clause RETRIEVE contient entre parenthèses les attributs à sélectionner des tables correspondantes. La dernière clause WHERE définit le prédicat de sélection exprimant les conditions que les données extraites doivent vérifier.

Traitement d'une requête QUEL

À titre d'exemple, une projection de la table EMPLOYÉ sur les attributs Affectation (numéro du département) et Nom de l'employé s'exprime comme suit en QUEL :

RANGE OF	e IS EMPLOYE
RETRIEVE	(e.Affectation, e.Nom)

Projection

La variable tuple e parcourt la table EMPLOYÉ. Elle est liée aux deux attributs e.Affectation et e.Nom. Dans ce cas, la variable tuple e retourne seulement les valeurs des attributs Affectation et Nom de l'employé.

Si l'on veut obtenir une liste des employés du département D6 qui habitent à Fribourg, la clause WHERE se présente comme suit :

<i>Projection combinée avec une sélection</i>	RANGE OF	e IS EMPLOYE
	RETRIEVE	(e.E#, e.Nom)
	WHERE	e.Ville = 'Fribourg' AND e.Affectation = 'D6'

Pour joindre les deux tables EMPLOYÉ et DÉPARTEMENT, nous définissons d'abord deux variables tuples, e et d, que nous utilisons ensuite dans la clause WHERE pour former correctement le prédicat de jointure :

<i>Jointure de deux tables</i>	RANGE OF	e IS EMPLOYE
	RANGE OF	d IS DEPARTEMENT
	RETRIEVE	(e.E#, e.Nom, d.Designation)
	WHERE	e.Affectation = d.D#

Si nous nous intéressons seulement au nombre d'employés qui travaillent au département D6, la réponse est donnée par la fonction COUNT ci-après :

<i>Fonctions prédéfinies</i>	RANGE OF	e IS EMPLOYE
	RETRIEVE	COUNT (e.E#)
	WHERE	e.Affectation = 'D6'

D'autres fonctions existent aussi pour calculer la somme, la moyenne, le maximum et le minimum : SUM, AVG, MAX et MIN.

Il est simple de générer des tables temporaires. Par exemple, la requête suivante nous permet d'insérer des valeurs désirées dans une nouvelle table après les avoir extraites d'une table existante :

<i>Tables temporaires</i>	RANGE OF	e IS EMPLOYE
	RETRIEVE	INTO TABLE_TEMP (e.E#, e.Nom)
	WHERE	e.Affectation = 'D6'

Comme la clause RETRIEVE, l'insertion de tuples dans une table peut se faire par la clause APPEND. Pour continuer l'exemple précédent, si nous voulons ajouter dans la table TABLE_TEMP les employés domiciliés à Bulle, nous écrivons la requête suivante :

```
RANGE OF      e IS EMPLOYE
APPEND TO     TABLE_TEMP (e.E#, e.Nom)
WHERE         e.Ville = 'Bulle'
```

*Extension
d'une table*

Les opérateurs ensemblistes de l'algèbre relationnelle sont ainsi mis en œuvre dans les deux clauses RETRIEVE et APPEND.

3.4.3 QBE

QBE est un langage de base de données, doté d'une interface graphique, avec lequel l'utilisateur conçoit et exécute ses requêtes étape par étape. Tout d'abord, l'utilisateur visualise la structure générale d'une table par la commande suivante :

*Traitement des
requêtes par
l'approche intuitive*

```
DRAW          EMPLOYE
```

Le résultat est une représentation graphique de la table EMPLOYÉ avec tous ses attributs :

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation

Ce squelette sert maintenant à formuler des sélections en le complétant par des commandes sous la forme de codes tels que «P.», et par des variables ou des constantes. Par exemple, une liste de noms des employés et des numéros de département s'obtient en remplissant le squelette comme suit :

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
		P.			P.

Projection

La commande «P.» (initiale de *print*) ordonne l'affichage de toutes les valeurs de données contenues dans les colonnes choisies. C'est ainsi que nous réalisons des projections sur les attributs d'une table.

Si l'utilisateur désire sélectionner tous les attributs, il écrit le code de commande directement sous le nom de la table considérée :

Sélection

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
P.					

Nous formulons des requêtes complexes sur la table EMPLOYÉ en introduisant des critères de sélection. C'est, par exemple, lorsque l'utilisateur veut obtenir tous les noms des employés domiciliés à Fribourg et travaillant au département D6 (voir figure 3-8). À cette fin, il écrit la constante Fribourg dans la colonne Ville et le numéro de département D6 dans la colonne Affectation :

Sélection conditionnelle

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
		P.		'Fribourg'	'D6'

Lorsque plusieurs critères de sélection sont définis sur la même ligne, ils sont liés par l'opérateur AND. Pour exprimer des critères liés par l'opérateur OR, il faut les placer sur deux lignes comme dans l'exemple suivant :

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
		P.		'Fribourg'	
		P.			'D6'

La requête permet de sélectionner tous les noms des employés qui habitent à Fribourg ou qui travaillent au département D6. Le résultat est une table qu'on peut aussi obtenir par l'expression équivalente suivante en algèbre relationnelle :

$$\pi_{\text{Ville}}(\sigma_{\text{Ville}=\text{Fribourg} \text{ OR } \text{Affectation}=\text{D6}}(\text{EMPLOYÉ}))$$

Les variables pour l'opérateur de jointure

QBE nous permet d'utiliser non seulement des constantes, mais aussi des variables pour formuler les requêtes. Une variable doit toujours commencer par un trait souligné «_», suivi d'une chaîne de caractères quelconque. L'emploi des variables est par exemple nécessaire dans les opérations de jointure. Considérons les tables EMPLOYÉ et DÉPARTEMENT en visualisant leurs structures par la commande DRAW. L'utilisateur désire obtenir les noms et les adresses des employés du département Informatique. Avec QBE, la requête d'interrogation se présente comme suit :

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
		P.	P.	P.	_D

DÉPARTEMENT	D#	Description
	_D	'Informatique'

*Jointure de
deux tables*

La jointure des deux tables EMPLOYÉ et DÉPARTEMENT se réalise par la variable «_D» qui représente le prédicat de jointure «Affectation=D#».

Pour insérer un nouvel employé (E20, Morel, chemin du Cerisier, Marly, D6) dans la table EMPLOYÉ, l'utilisateur place la commande d'insertion «I.» (initiale de *insert*) dans la colonne contenant le nom de la table, et remplit les colonnes restantes avec les valeurs des attributs :

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
I.	'E20'	'Morel'	'chemin du Cerisier'	'Marly'	'D6'

*Opérateur
d'insertion*

L'utilisateur peut modifier un ensemble de tuples en écrivant la commande «U.» (initiale de *update*) dans les colonnes appropriées. La commande «D.» (initiale de *delete*), placée sous le nom d'une table, permet d'y supprimer un ensemble de tuples.

Dans l'exemple suivant, l'utilisateur construit une requête pour supprimer les employés habitant à Fribourg :

EMPLOYÉ	E#	Nom	Rue	Ville	Affectation
D.				'Fribourg'	

*Opérateur de
suppression*

Comme l'expression «Query by Example» nous le fait comprendre, QBE n'offre pas la possibilité de créer des tables et de gérer les droits d'accès des utilisateurs à la différence des langages SQL et QUEL. QBE se limite aux interrogations et à la manipulation des bases de données, et dans ce cadre, c'est un langage relationnel complet au même titre que SQL et QUEL.

La mise en œuvre de QBE dans la pratique a montré que la lisibilité des requêtes complexes formulées dans ce langage est moins bonne que celle des instructions SQL. C'est pourquoi il n'est pas

surprenant que des utilisateurs occasionnels préfèrent le langage SQL pour exprimer leurs requêtes avancées.

3.5 Les langages immergés

SQL immergé

Les langages relationnels d'interrogation et de manipulation de données ne sont pas réservés uniquement à l'usage en tant que langages autonomes en mode conversationnel. Ils sont aussi intégrés dans des langages de programmation (langages hôtes). Nous étudions dans cette section les éléments indispensables à l'intégration des langages relationnels dans un environnement de programmation.

Notion de curseur

Il convient d'expliquer le concept du langage immergé à l'aide d'un exemple en SQL. Pour qu'un programme puisse lire une table par la commande SELECT, il doit pouvoir avancer *d'un tuple à l'autre* dans la table considérée. Cette progression fait appel à la notion de CURSEUR.

Un CURSEUR *est un pointeur* qui parcourt un ensemble de tuples dans un certain ordre prédéterminé par le système de bases de données. Puisqu'un programme conventionnel ne peut traiter une table entière d'un seul trait, le concept du CURSEUR permet de suivre une approche de traitement tuple à tuple.

Pour faire la sélection dans une table, il faut définir un CURSEUR dans le programme comme suit :

Déclaration d'un curseur

```
DECLARE nom du curseur CURSOR FOR
      commande SELECT
```

Pour traduire notre intention de modifier les données dans une table, nous définissons un CURSEUR par l'instruction suivante :

```
DECLARE nom du curseur CURSOR FOR
      commande SELECT FOR
      UPDATE OF champ [,champ]
```

Dans cette approche, le traitement d'une table se fait enregistrement par enregistrement, c'est-à-dire un tuple à la fois.

Lorsque cela arrive, la mise à jour porte sur une partie ou la totalité des données du tuple vers lequel pointe le curseur. S'il faut traiter une table d'après une séquence de tri déterminée, nous ajoutons une clause ORDER BY dans la déclaration du curseur.

Pour naviguer dans les tables, nous pouvons utiliser plusieurs CURSEURS dans un programme. Chaque curseur doit être déclaré, puis activé par la commande OPEN, et désactivé après l'usage par la commande CLOSE. L'accès proprement dit à une table et le transfert des valeurs de données aux variables appropriées dans un programme se réalisent par la commande FETCH. Les types de variables, définis dans le langage de programmation choisi, doivent être compatibles avec les formats des champs de la table considérée. La syntaxe de la commande FETCH se présente comme suit :

FETCH nom du curseur INTO variable hôte [, variable hôte]

Possibilité de déclarer plusieurs curseurs

Commande Fetch

Chaque commande FETCH avance la position du CURSEUR au tuple suivant, soit selon l'ordre physique des tuples dans la table en question, soit selon une clause ORDER BY. Si le curseur ne trouve plus de tuples, le programme émet un code d'état associé à cette situation.

Le concept de CURSEUR permet *d'intégrer un langage de requête et de manipulation de données ensembliste dans un langage hôte procédural*. Ainsi, les constructions du langage SQL peuvent être utilisées aussi bien en mode interactif (ad-hoc) qu'en mode programmation (langage immergé). Pour tester des modules de programme en langage immergé, le SQL interactif présente un avantage certain car nous pouvons l'utiliser à tout moment pour analyser et vérifier le contenu des tables de test. Cependant, il convient aussi de constater que l'intégration d'un langage relationnel de requête et de manipulation de données dans un langage hôte donne lieu généralement à des modules de programme qui manquent de clarté ou qui s'écartent du style de programmation structurée.

Langage interactif ou immergé

Avant la traduction et l'exécution d'un programme contenant des éléments d'un langage relationnel immergé, un précompilateur a pour tâche de vérifier la syntaxe du langage SQL. Il doit par exemple signaler une erreur lorsque dans une instruction SELECT, le nombre

Traduction et optimisation

d'attributs choisis diffère du nombre de variables déclarées dans la clause INTO. Lorsqu'il n'existe plus d'erreurs, des modules d'interrogation et d'accès à la base de données sont générés. Une stratégie de traitement optimale sera définie dans un plan d'exécution stocké dans les tables du système. Elle sera appliquée lors de l'exécution du programme (voir chapitre 4).

3.6 Traitement des valeurs nulles

Justification des valeurs nulles

En travaillant avec une base de données, nous rencontrons souvent des données dont les valeurs ne sont pas, ou pas encore, connues dans une table. Par exemple, un employé doit être inséré dans la table EMPLOYÉ sans qu'on sache son adresse complète. Dans un cas pareil, il s'avère judicieux d'utiliser des *valeurs* dites *nulles* (*null value*, en anglais) au lieu de valeurs peu significatives, voire fausses.

Valeurs nulles

La valeur nulle représente une valeur de donnée qui n'est pas (ou pas encore) connue dans une colonne d'une table.

La valeur nulle est une donnée de remplissage fictive

Il ne faut pas confondre la valeur nulle - symbolisée par le point d'interrogation «?» - et le chiffre «Zéro» ou le caractère «Espace» (Space). Dans une base de données relationnelle, les deux dernières valeurs expriment un contenu informationnel précis, alors que la valeur nulle est une *donnée de remplissage fictive* (*dummy*, en anglais).

Dans la figure 3-11, la table EMPLOYÉ contient plusieurs fois la valeur nulle pour les attributs Rue et Ville. Naturellement, les attributs d'une table ne doivent pas tous admettre des données de valeur nulle, sinon des conflits surgiront inévitablement. Par définition, les clés primaires telles que le numéro d'employé dans la table EMPLOYÉ, ne doivent pas être nulles. Pour la clé étrangère «Affectation», c'est l'architecte ou l'administrateur de base de données qui décide de l'admissibilité des valeurs nulles. L'observation du monde réel est alors déterminante.

EMPLOYÉ

E#	Nom	Rue	Ville	Affectation
E19	Savoy	avenue de la Gare	Romont	D6
E1	Meier	?	?	D3
E7	Humbert	route des Alpes	Bulle	D5
E4	Brodard	?	?	D6

```

SELECT *
FROM EMPLOYE
WHERE Ville = 'Fribourg'
UNION
SELECT *
FROM EMPLOYE
WHERE NOT Ville = 'Fribourg'

```

TABLE RÉSULTAT

E#	Nom	Rue	Ville	Affectation
E19	Savoy	avenue de la Gare	Romont	D6
E7	Humbert	route des Alpes	Bulle	D5

Figure 3-11
Le traitement
délicat des
valeurs nulles

Normalement les clés étrangères n'admettent pas de valeurs nulles, sauf si elles sont soumises à une règle d'intégrité référentielle spéciale. Ainsi, par exemple, dans la règle de suppression pour la table référencée DÉPARTEMENT, nous pouvons spécifier si les valeurs de la clé étrangère dans la table référençante doivent être mises à la valeur nulle ou pas. La règle d'intégrité référentielle «Mettre à la valeur nulle» signifie que *les valeurs d'une clé étrangère deviennent nulles lors de la suppression des tuples référencés*. Supprimons par exemple le tuple (D6, Finances) dans la table DÉPARTEMENT soumise à la règle d'intégrité référentielle «Mettre à la valeur nulle». À ce moment, les valeurs de la clé étrangère «Affectation» deviennent nulles pour les employés Savoy et Brodard dans la figure 3-11. Cette règle complète les règles de suppression restreinte et en cascade déjà énoncées dans la section 2.5.

Règle de
suppression avec
mise à la valeur
nulle

La gestion des valeurs nulles n'est pas sans poser des problèmes. Au-delà des valeurs TRUE et FALSE, la valeur nulle introduit notamment la nouvelle information UNKNOWN. Nous nous écartons donc de la logique bivalente selon laquelle chaque prédicat ne peut être que vrai ou faux.

Prudence en
présence de
valeurs nulles

La figure 3-11 présente une requête pour sélectionner de la table EMPLOYÉ tous les individus qui habitent soit à Fribourg soit en dehors de Fribourg. Or la table résultat ne contient qu'une liste incomplète des employés extraits de la table originelle car les adresses de certains employés sont inconnues. Ce résultat est manifestement contraire à la logique classique selon laquelle l'union du sous-ensemble «Employés domiciliés à Fribourg» et de son complément «Employés NON domiciliés à Fribourg» doit produire l'ensemble de tous les employés.

*La valeur nulle
mène à la logique
trivalente*

La logique des prédicats à trois valeurs de vérité (TRUE, FALSE et UNKNOWN) est aussi appelée logique trivalente, car chaque prédicat peut être «vrai», «faux» ou «inconnu». Cette logique est moins familière et exige des connaissances adéquates de la part des utilisateurs de bases de données relationnelles, car l'interprétation des résultats de requêtes portant sur des tables qui admettent la valeur nulle est délicate. C'est pourquoi dans la pratique, soit on renonce à l'usage des valeurs nulles, soit on introduit des valeurs par défaut pour représenter les données manquantes. Dans l'exemple de la table EMPLOYÉ, l'adresse de l'entreprise peut être insérée comme valeur par défaut lorsque l'adresse privée d'un employé est inconnue.

3.7 La protection des données

*Protection contre
l'utilisation non
autorisée des
données*

La *protection des données* (*data protection*, en anglais) vise à protéger les données contre l'accès et l'utilisation non autorisés. Les mesures de protection englobent des techniques pour identifier une personne de manière unique et gérer les autorisations d'accès aux données, ainsi que des méthodes cryptographiques pour le stockage et la transmission des informations confidentielles.

*Protection contre la
destruction et la
perte des données*

À la différence de la protection des données, la notion de *sécurité des données* (*data security*, en anglais) concerne les mesures techniques et les solutions logicielles destinées à protéger un système contre la corruption, la destruction ou la perte des données. Au chapitre 4 nous traiterons des méthodes pour sauvegarder et restaurer les bases de données.

Une mesure fondamentale de protection des données consiste en ceci : seules les tables ou parties de tables nécessaires à l'accomplissement de leurs tâches sont mises à la disposition des utilisateurs dûment autorisés. Dans ce but nous créons des *vues* (*views*, en anglais) sur les tables. Une vue est définie par la commande SELECT à partir d'une ou de plusieurs tables physiques :

CREATE VIEW nom de la vue AS commande SELECT

PERSONNEL

E#	Nom	Ville	Salaire	Affectation
E19	Savoy	Romont	75'000	D6
E1	Meier	Fribourg	50'000	D3
E7	Humbert	Bulle	80'000	D5
E4	Brodard	Fribourg	65'000	D6

```
CREATE VIEW
EMPLOYE AS
SELECT E#, Nom, Ville, Affectation
FROM PERSONNEL
```

E#	Nom	Ville	Affectation
E19	Savoy	Romont	D6
E1	Meier	Fribourg	D3
E7	Humbert	Bulle	D5
E4	Brodard	Fribourg	D6

```
CREATE VIEW
GROUPE_D AS
SELECT E#, Nom, Salaire, Affectation
FROM PERSONNEL
WHERE Salaire BETWEEN 70'000 AND 90'000
```

E#	Nom	Salaire	Affectation
E19	Savoy	75'000	D6
E7	Humbert	80'000	D5

Le concept de vue permet de restreindre une table

Définir une vue

*Figure 3-12
Définition des vues pour protéger les données*

La figure 3-12 donne à titre d'exemple deux vues définies à partir de la table de base PERSONNEL. La vue EMPLOYÉ contient tous les attributs sauf les salaires. Dans la vue GROUPE_D figurent uniquement les employés qui gagnent entre 70'000 et 90'000 francs suisses par année. De manière analogue, nous pouvons créer d'autres vues, par

exemple pour mettre les données confidentielles à la disposition des responsables du personnel selon les classes de salaires qu'ils supervisent. Les deux vues dans la figure 3-12 illustrent un important mécanisme de protection des données : d'une part l'accès aux tables peut être personnalisé par groupes d'utilisateurs grâce aux projections sur des attributs déterminés par leurs besoins. D'autre part le contrôle d'accès aux classes de salaires, par exemple, peut se baser sur des valeurs fournies au moment de l'exécution d'une application. Ces valeurs détermineront les vues autorisées grâce à la clause WHERE.

La modification des vues est problématique

Nous interrogeons les vues comme nous le faisons avec les tables. En revanche, la manipulation des vues ne s'exécute pas de manière unique à chaque opération. Lorsqu'une vue résulte, par exemple, de la jointure de plusieurs tables, le système de bases de données rejette toute opération de mise à jour de la vue en question.

Les vues ne sont pas matérialisées

Il est primordial d'éviter une administration redondante de la table de base et de ses différentes vues qui, à cette fin, ne doivent donc pas stocker les données de la table en question. Il s'agit plutôt de créer simplement des *définitions de vues*. C'est au moment où l'on interroge une vue par la commande SELECT qu'une table résultat sera générée avec des valeurs de données autorisées provenant de la table de base correspondante. Pour cette raison, les deux vues EMPLOYÉ et GROUPE_D sont représentées en hachures dans la figure 3-12.

Gestion des droits de l'utilisateur

Une protection efficace des données ne se limite pas à la seule création des vues pour restreindre les tables. Les fonctions qui opèrent sur les tables doivent aussi être définies par catégories d'utilisateurs. À cette fin, les commandes SQL GRANT et REVOKE permettent de gérer les droits des utilisateurs.

Les privilèges accordés à l'utilisateur par GRANT peuvent lui être retirés par REVOKE :

Attribution et révocation des droits

GRANT privilege ON relation TO user

REVOKE privilege ON relation FROM user

La commande GRANT *met à jour la liste des privilèges* pour que l'ayant droit puisse lire, insérer ou supprimer les données dans des

tables ou des vues spécifiques. À l'inverse, la commande REVOKE permet de retirer à l'utilisateur un privilège qui lui a été accordé.

Par exemple, si nous voulons seulement autoriser la lecture de la vue EMPLOYÉ dans la figure 3-12, il faut écrire :

```
GRANT SELECT ON EMPLOYE TO PUBLIC
```

*Droit de lecture
accordé à
tout le monde*

En spécifiant PUBLIC au lieu d'une liste d'utilisateurs, nous accordons le droit de lecture à tous sans aucune restriction. Ce droit leur permet de consulter une partie de la table de base à travers la vue EMPLOYÉ.

Les privilèges peuvent être accordés de manière sélective. Dans l'exemple suivant, le droit de mise à jour de la vue GROUPE_D (figure 3-12) est accordé exclusivement à un responsable du personnel identifié par le numéro d'utilisateur ID37289 :

```
GRANT UPDATE ON GROUPE_D TO ID37289  
WITH GRANT OPTION
```

*Droit d'écriture
limité*

L'utilisateur ID37289 peut alors modifier la vue GROUPE_D. En outre, avec GRANT OPTION, il a la compétence de transmettre ce privilège ou un droit de lecture restreint aux autres utilisateurs, et aussi de les retirer ultérieurement. Ce concept nous permet de définir et de gérer l'interdépendance des droits.

*Transmission
des droits*

En mettant à la disposition de l'utilisateur final un langage relationnel de requête et de manipulation de données, l'administrateur de base de données ne doit pas sous-estimer l'ampleur des tâches administratives pour gérer l'attribution et la révocation des droits, même s'il dispose des commandes GRANT et REVOKE. La pratique montre que d'autres outils de contrôle sont nécessaires pour accomplir efficacement les tâches quotidiennes de mise à jour et de surveillance des droits des utilisateurs. En outre, des autorités de contrôle internes et externes peuvent exiger la mise en œuvre de dispositifs particuliers pour garantir l'utilisation légale des données sous protection (examiner à ce sujet le cahier des charges d'un responsable de la protection des données).

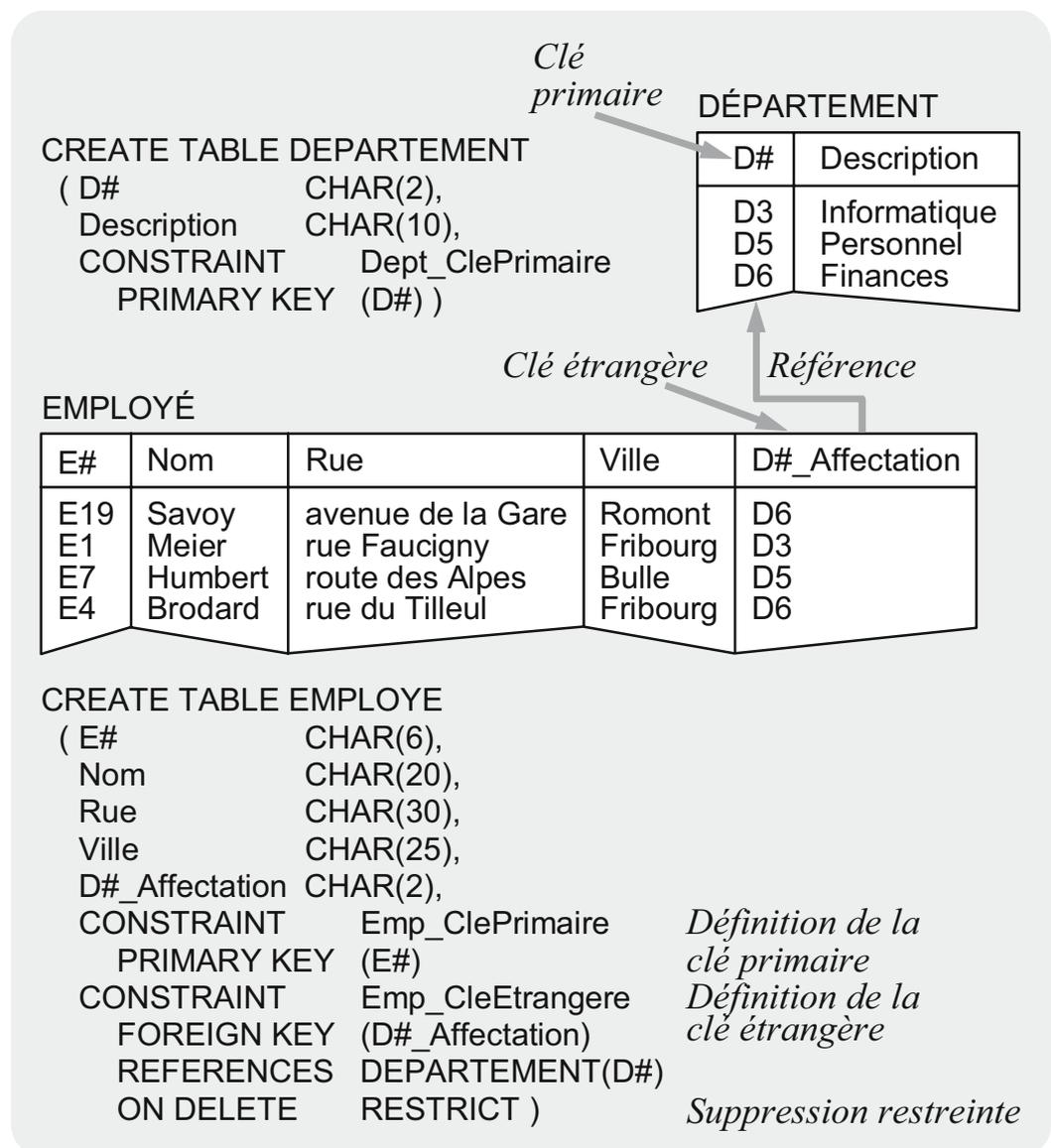
*Responsabilité du
chargé de la
protection des
données*

3.8 La formulation des contraintes d'intégrité

Un système de bases de données doit assurer l'intégrité des données

L'intégrité d'une base de données est une propriété vitale qui doit être garantie par tout système de gestion de bases de données. Les règles qui doivent être respectées lors de chaque opération d'insertion ou de mise à jour sont appelées *contraintes d'intégrité* (*integrity constraints*, en anglais). Il est logique de définir ces contraintes une fois pour toutes au niveau de la base de données, mais pas dans chaque programme. Nous distinguons les contraintes d'intégrité déclaratives et les contraintes d'intégrité procédurales.

Figure 3-13
Définition des contraintes d'intégrité référentielle



Les *contraintes d'intégrité déclaratives* (*declarative integrity constraints*, en anglais) sont définies à l'aide du langage de définition de données. Dans la commande de création d'une table (CREATE TABLE), on attribue un nom à chaque règle par la clause facultative CONSTRAINT. Les noms de contraintes sont gérés au niveau des tables système. Dans l'exemple de la figure 3-13 nous définissons la clé primaire de la table DÉPARTEMENT sous forme d'une contrainte d'intégrité nommée Dépt_CléPrimaire. De manière analogue nous spécifions la clé primaire et une clé étrangère pour la table EMPLOYÉ.

Contraintes
d'intégrité
déclaratives

Il existe plusieurs types de contraintes d'intégrité déclaratives :

- *Définition de la clé primaire* : la contrainte PRIMARY KEY définit une clé primaire unique dans une table. Par définition, cette clé n'admet jamais la valeur nulle.
- *Définition d'une clé étrangère* : la contrainte FOREIGN KEY définit une clé étrangère qui est liée à la table référencée par le mot-clé REFERENCES.
- *Unicité* : l'unicité des valeurs d'un attribut peut être imposée par la contrainte UNIQUE. Au contraire de la clé primaire, un attribut soumis à la contrainte UNIQUE peut contenir la valeur nulle.
- *Valeur nulle exclue* : la contrainte NOT NULL empêche un attribut d'être mis à la valeur nulle.
- *Règle de validation* : une règle de validation est déclarée dans une contrainte CHECK et doit être respectée par chaque tuple dans une table. Par exemple, pour la table PERSONNEL dans la figure 3-12, la règle suivante :

CONSTRAINT ValidationSalaire

CHECK (Salaire > 60'000 AND Affectation = 'D6')

impose un salaire annuel minimum de 60 000 francs à chaque employé du département D6.

- *Suppression avec mise à la valeur nulle* : avec la contrainte ON DELETE SET NULL déclarée dans une table dépendante, la suppression des tuples de la table de référence entraîne la mise à

la valeur nulle de la clé étrangère dans les tuples dépendants (voir 2.5).

- *Suppression restreinte* : avec la contrainte ON DELETE RESTRICT, la suppression d'un tuple de référence est refusée tant qu'il possède des tuples dépendants (voir 2.5).
- *Suppression en cascade* : avec la contrainte ON DELETE CASCADE, la suppression d'un tuple de référence entraîne celle des tuples dépendants (voir 2.5).

Dans la figure 3-13 une contrainte de suppression restreinte est définie dans la table EMPLOYÉ qui est liée à la table de référence DÉPARTEMENT par la clé étrangère D#_Affectation. En vertu de cette règle, on peut supprimer un département seulement s'il ne possède plus de tuples dépendants dans la table EMPLOYÉ.

*Violation de
l'intégrité
référentielle*

Par conséquent, la commande :

```
DELETE FROM DEPARTEMENT WHERE D# = 'D6'
```

provoquera une erreur, car les deux employés Savoy et Brodard sont encore enregistrés dans le département Finances.

Comme pour la suppression, les opérations d'insertion et de mise à jour peuvent aussi être soumises aux contraintes d'intégrité déclaratives. Ainsi, l'opération d'insertion suivante :

```
INSERT INTO EMPLOYE
VALUES ('E20', 'Morel', 'chemin du Cerisier', 'Marly', 'D7')
```

provoquera l'affichage d'un message d'erreur, car le département D7 n'existe pas encore dans la table référencée DÉPARTEMENT.

*Contraintes
d'intégrité
procédurales*

Dans les *contraintes d'intégrité procédurales (procedural integrity constraints, en anglais)*, les règles des *déclencheurs (trigger, en anglais)* jouent un rôle important. Le déclencheur est une alternative aux contraintes d'intégrité déclaratives : il entraîne l'exécution d'un groupe de commandes. Nous définissons un déclencheur en spécifiant essentiellement trois éléments : le nom du

déclencheur, une opération sur la base de données et un ensemble d'actions à déclencher. Par exemple :

CREATE TRIGGER EffectifPersonnel	Déclencheur	<i>Déclaration d'un déclencheur</i>
ON INSERTION OF EMPLOYE :	Opération sur la base de données	
UPDATE DEPARTEMENT	Action à déclencher	
SET Effectif = Effectif + 1		
WHERE D# = D#_Affectation		

Dans cet exemple, nous admettons que la table DÉPARTEMENT possède un attribut Effectif qui indique le nombre total des employés dans chaque département. À chaque nouvelle insertion dans la table EMPLOYÉ, le déclencheur provoque la mise à jour de l'attribut Effectif pour le département concerné.

Fonctionnement d'un déclencheur

La mise en œuvre des déclencheurs n'est pas toujours simple, car un déclencheur est généralement susceptible d'invoquer d'autres déclencheurs. Le problème consiste alors à déterminer la fin de l'ensemble des actions déclenchées. En général, dans les systèmes de bases de données commercialisés, l'activation simultanée des déclencheurs est impossible, ceci afin de garantir une séquence d'actions unique et pour que l'exécution d'un déclencheur se termine toujours proprement.

Le système de bases de données garantit l'absence de conflits

3.9 Notes bibliographiques

Les premiers travaux de Codd (1970) présentent à la fois le modèle relationnel et l'algèbre relationnelle. On trouve les développements ultérieurs sur l'algèbre relationnelle et le calcul relationnel dans les ouvrages de référence suivants : Date (2004), Elmasri et Navathe (2004) et Maier (1983). Ullman (1982) met en évidence l'équivalence de l'algèbre relationnelle et le calcul relationnel.

Les bases de l'algèbre relationnelle et du calcul relationnel

Le langage SQL est issu des travaux de recherche d'Astrahan et al. (1976) sur le système de bases de données relationnelles «System R». QUEL, défini par Stonebraker (1986), est le langage de requête et

Origine des langages relationnels

de manipulation des données du système de bases de données «Ingres». «System R» et «Ingres» sont les premiers systèmes relationnels. Également au milieu des années 1970, QBE fut défini par Zloof (1977).

Littérature sur les aspects du langage

Le manuel de Lockemann et Schmidt (1993) sur les bases de données donne un aperçu des langages de requête et de manipulation des données. Les aspects du langage sont traités dans les ouvrages de Celko (1999), Heuer et Saake (2000), Kemper et Eickler (2001), Lang et Lockemann (1995).

Livres de texte sur SQL

Le langage SQL est présenté dans un grand nombre de livres de texte dont voici une sélection non exhaustive : en anglais, Taylor (2003) ; en français, Akoka et Comyn-Wattiau (2001), Brouard et Soutou (2005), Celko (2000), Hernandez et Viescas (2001), Lentzner (2004) ; en allemand, Eilers et al. (1986), Kuhlmann et Müllmerstadt (2000), van der Lans (1988), Panny et Taudes (2000), Sauer (1992), Vossen et Witt (1988). Le livre de Jones (2005), en français, est consacré aux fonctions SQL, d'une part les fonctions communes, et d'autre part les fonctions spécifiques aux principaux éditeurs de systèmes de bases de données actuels.

Les normes SQL

Darwen et Date (1997) consacrent leurs travaux au standard SQL. L'article de Pistor (1993) présente les concepts orientés objet du standard SQL. L'ouvrage de Türker (2003) approfondit les normes SQL:1999 et SQL:2003.

4 Les composants de l'architecture d'un système de bases de données

4.1 Vue d'ensemble de l'architecture du système

Parmi les points forts d'un système de bases de données relationnelles, nous avons mis en relief le fait qu'aucune connaissance préalable de la structure interne d'un tel système n'est requise pour utiliser les langages de requête et de manipulation de données qui constituent l'interface entre le système et l'utilisateur. Quel est alors l'intérêt de ce chapitre entièrement consacré à l'architecture des systèmes de bases de données relationnelles ?

Les langages relationnels d'interrogation et de manipulation de données sont des langages descriptifs qui définissent des commandes expressives du style «Donnez-moi la liste de tous les employés domiciliés à Fribourg !» L'intérêt consiste maintenant à comprendre le mécanisme par lequel *un système de bases de données traduit et exécute une requête ensembliste*. Au commencement de la théorie des bases de données relationnelles, la principale préoccupation était de savoir s'il est possible de rendre opérationnel un système de bases de données ensembliste et comment le faire fonctionner. Ainsi, par exemple, les opérations de jointure sont longues et coûteuses dans des applications de bases de données relationnelles. Nous avons déjà appris qu'une jointure est équivalente à un produit cartésien combiné avec une sélection. Or, le produit cartésien de deux tables est un résultat intermédiaire qui peut être fort volumineux. Tôt ou tard, même l'utilisateur occasionnel en ressentira les conséquences : *le temps d'attente d'une réponse à sa requête lui paraîtra excessivement long*. En outre, il sera surpris de constater qu'une condition de sélection peut sensiblement réduire la taille d'une table résultat. Cela

*Avènement de
l'approche
ensembliste*

nous montre l'utilité de comprendre *la signification et le but de l'optimisation des requêtes* et d'explorer les principales stratégies de jointure des tables.

Garantie du bon fonctionnement dans un environnement multi-utilisateur

En général, l'utilisateur ne dispose pas d'une base de données à lui tout seul. Il la partage au contraire avec plusieurs personnes simultanément. Naturellement, il appartient au système de bases de données d'assurer son bon fonctionnement dans un environnement multi-utilisateur. Le concept de *transaction* est ici primordial : par transaction nous entendons un programme qui effectue des *opérations de lecture et d'écriture dans une base de données selon le «principe du tout ou rien»*. En d'autres termes, il faut qu'une transaction soit complètement traitée par le système de bases de données, ou bien qu'elle ne s'exécute pas du tout. Le principe du «tout ou rien» garantit la cohérence des données : le traitement partiel d'une transaction est exclu, de même qu'une mise à jour incomplète de la base de données. La compréhension du principe du «tout ou rien» et d'autres concepts clés dans la gestion des transactions permet aux développeurs de maîtriser la programmation des applications et aux utilisateurs d'exploiter correctement une base de données.

Augmentation de la performance grâce au parallélisme

Une autre préoccupation porte sur la *parallélisation des transactions*, car les exigences de performance ne permettent pas toujours le traitement séquentiel des transactions individuelles. Certes, l'utilisateur n'a pas besoin d'approfondir toutes ces questions. Il est néanmoins important d'expliquer comment un système de bases de données exécute correctement les transactions *«concurrentes»* (*concurrent*, en anglais) à l'aide de protocoles ou de techniques de validation. Nous étudierons des scénarios où les transactions se bloquent mutuellement et des méthodes de résolution de tels conflits par le système de bases de données.

Structures de données et d'accès internes

Enfin, il est important que l'administrateur et l'expert en base de données comprennent les *mécanismes internes de gestion et de maintenance des tables*. Particulièrement dans la mise en œuvre des grandes bases de données, ils doivent définir correctement les paramètres internes pour l'indexation, l'allocation de la mémoire, la taille de la mémoire-tampon, etc., car la performance du système en dépend directement en termes de débit, de temps de réponse, de

sécurité de fonctionnement, etc. Avec les *structures de stockage des données et les structures d'accès*, nous aborderons les méthodes de calcul des adresses (hachage), les structures de données hiérarchiques (arbres) et, apparues récemment, les structures de données multidimensionnelles. Les méthodes de calcul des adresses permettent l'organisation hachée du stockage des données, les structures arborescentes introduisent la gestion hiérarchique des enregistrements de données et des clés d'accès, et les structures de données multidimensionnelles permettent de découper l'espace de données de manière à garantir l'efficacité des accès multi-attributs.

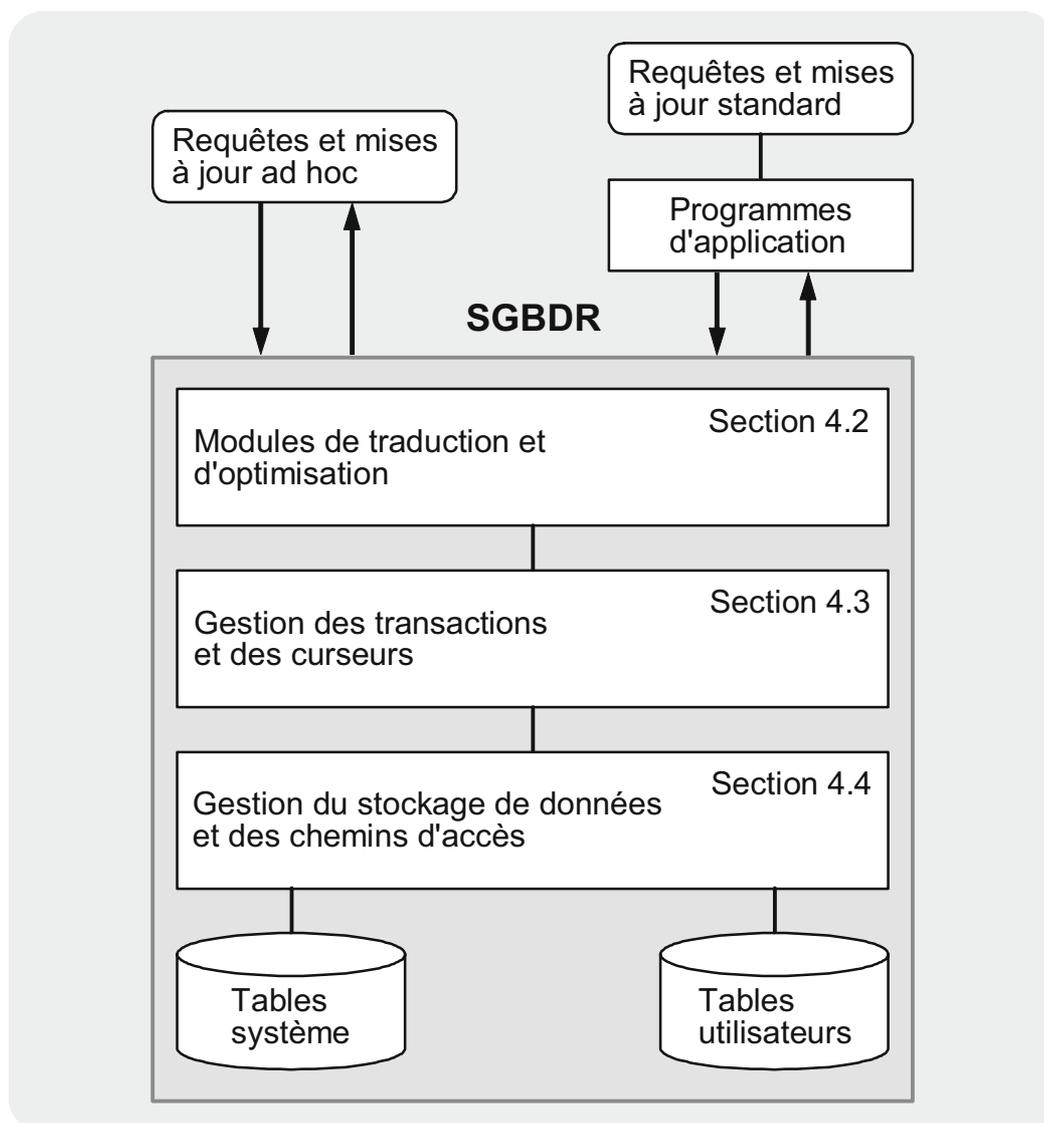


Figure 4-1
Présentation
sommaire de
l'architecture du
système de bases
de données

La figure 4-1 présente une vue d'ensemble des composants de l'architecture du système qui ont chacun un rôle déterminé :

Un système de bases de données est composé de plusieurs couches

- La traduction des requêtes a pour but de transformer une interrogation descriptive en expressions équivalentes de l'algèbre relationnelle en faisant appel aux techniques d'optimisation et aux stratégies de jointure (4.2).
- La gestion des transactions assure le fonctionnement du système sans conflit entre elles dans un environnement multi-utilisateur : plusieurs sessions de travail simultanées sont actives dans une même base de données sans compromettre son intégrité (4.3).
- Les structures de stockage et d'accès permettent la gestion efficace des tables dans le système de bases de données. Les structures de données doivent supporter la mise à jour dynamique des entrées dans les tables (4.4).
- En cas de panne ou d'erreur pendant le traitement des transactions, le système de bases de données permet de rétablir l'état originel de la base incomplètement mise à jour (4.5).

Différentes interfaces utilisateurs

Selon la figure 4-1, différentes interfaces utilisateurs sont disponibles dans un système de bases de données relationnelles. Ainsi, l'utilisateur occasionnel peut lancer des requêtes ad hoc et analyser les informations extraites de la base de données. Pour les grandes applications, des programmes sont mis au point pour exécuter des requêtes standard et manipuler la base de données.

4.2 Traduction et optimisation des requêtes

4.2.1 Construction d'un arbre d'interrogation

Interface utilisateur ensembliste

Dans un système de bases de données relationnelles, l'interface utilisateur est de nature ensembliste car elle permet aux utilisateurs de travailler avec des tables ou des vues disponibles. C'est pourquoi un système de bases de données doit être capable de traduire et d'optimiser les commandes formulées par l'utilisateur dans un langage relationnel d'interrogation et de manipulation de données. Il est important que la construction et l'optimisation d'un arbre d'interrogation se réalisent sans intervention humaine.

Arbre d'interrogation

Un *arbre d'interrogation* (*query tree*, en anglais) visualise graphiquement une interrogation relationnelle traduite en une *expression équivalente de l'algèbre relationnelle*. L'arbre est composé de feuilles qui correspondent aux tables utilisées dans la requête, d'un nœud racine et des nœuds intermédiaires qui désignent les opérations algébriques.

Un arbre d'interrogation est l'abstraction d'une commande SQL

À titre d'illustration, nous développons un exemple en SQL avec les deux tables définies précédemment, EMPLOYÉ et DÉPARTEMENT (Figure 4-2). Pour obtenir une liste des villes où habitent les employés du département Informatique, nous formulons la requête suivante :

Exemple d'interrogation

```
SELECT      Ville
FROM        EMPLOYE, DEPARTEMENT
WHERE       Affectation=D# AND Description='Informatique'
```

La suite d'opérations algébriques équivalentes s'écrit :

```
TABLE :=  $\pi_{\text{Ville}}$ 
        ( $\sigma_{\text{Description=Informatique}}$ 
        (EMPLOYÉ |  $\times$  |  $\sigma_{\text{Affectation=D\#}}$  DÉPARTEMENT))
```

Cette expression algébrique commence par la jointure des tables EMPLOYÉ et DÉPARTEMENT, basée sur le numéro de département commun. Puis, à partir du résultat intermédiaire, les employés qui travaillent au département Informatique sont sélectionnés. Enfin, une projection permet d'établir la liste des villes recherchées. Dans la figure 4-2 ces opérations algébriques sont représentées sous la forme d'un arbre d'interrogation qui se lit de bas en haut de la manière suivante :

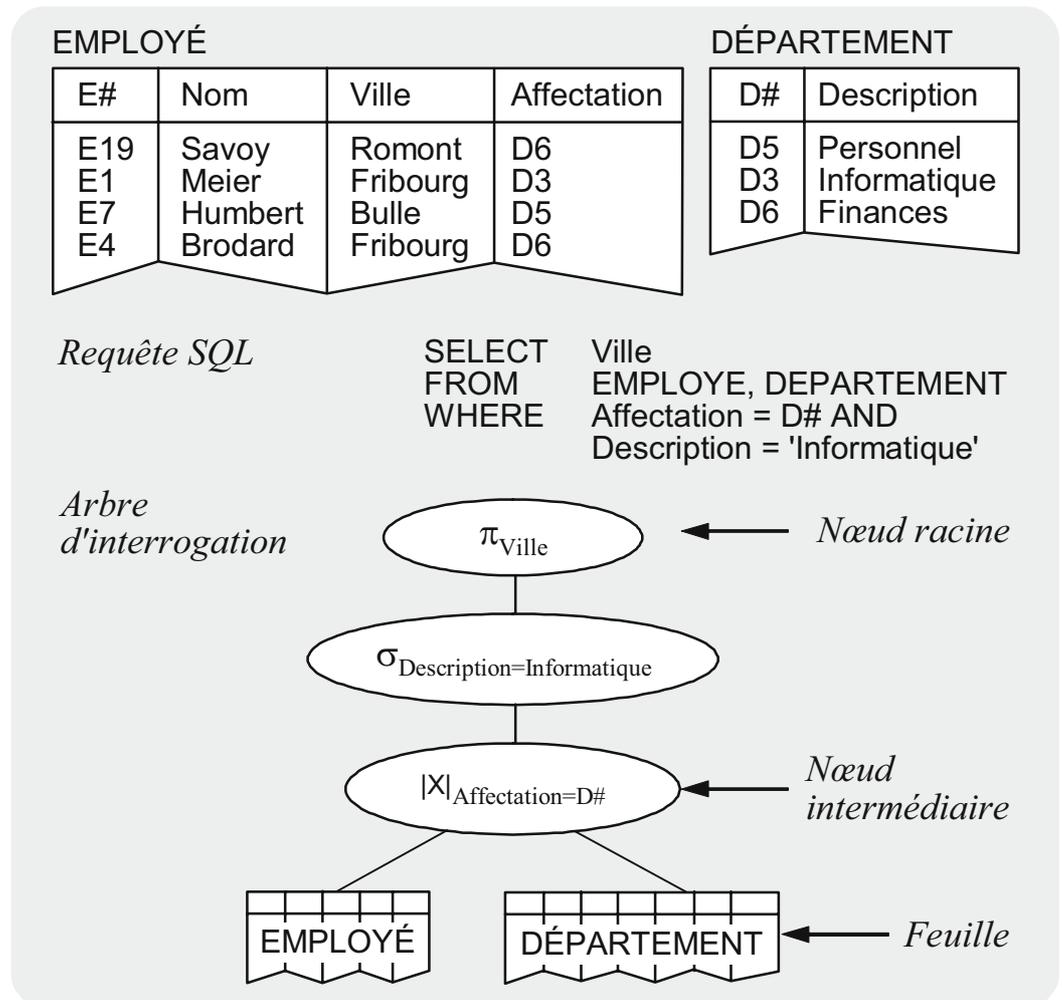
Un arbre d'interrogation est évalué de bas en haut

Les feuilles représentent les tables EMPLOYÉ et DÉPARTEMENT utilisées dans l'interrogation. Le premier nœud intermédiaire (opérateur de jointure) joint ces deux tables en une table résultat temporaire. Le second nœud intermédiaire (opérateur de sélection) réduit ce résultat aux seules entrées dont l'attribut Description a la

Interprétation d'un arbre d'interrogation

valeur 'Informatique'. Enfin, le nœud racine est la projection qui génère une table résultat contenant les villes recherchées.

Figure 4-2
Arbre d'interrogation d'une requête conditionnelle impliquant deux tables



Opérateurs unaires et binaires

Le nœud racine et les nœuds intermédiaires dans un arbre d'interrogation mènent à un ou deux sous-arbres. On parle d'opérateur unaire ou binaire selon que l'opérateur associé à un nœud porte sur une ou deux tables intermédiaires (sous-arbres). Les *opérateurs binaires* admettent deux tables comme opérandes et comprennent l'union, l'intersection, la différence, le produit cartésien, la jointure et la division. Les *opérateurs unaires* n'ont qu'une seule table comme opérande et comprennent la projection et la sélection (voir les figures 3-2 et 3-3 du chapitre précédent).

Les tables système contribuent à la vérification syntaxique

La construction d'un arbre d'interrogation est la première étape du processus de traduction et d'exécution d'une interrogation de la base de données relationnelle. Les noms des tables et des attributs fournis par l'utilisateur doivent exister dans les tables système afin

que les étapes de traitement suivantes puissent avoir lieu. C'est donc l'arbre d'interrogation qui permet de vérifier à la fois la syntaxe de la requête et les droits d'accès de l'utilisateur. Il existe d'autres contraintes de sécurité qui ne peuvent être vérifiées qu'au moment de l'exécution d'une requête. Tel est le cas, par exemple, d'une protection des données basée sur des valeurs ou des codes que l'utilisateur spécifie au moment de l'interrogation et qui autorisent l'accès à certaines catégories précises de données.

Après le contrôle d'accès et la vérification de l'intégrité des données, la seconde étape consiste à choisir et optimiser les chemins d'accès. Enfin, dans la troisième étape, a lieu soit la génération du code proprement dite, soit l'exécution de la requête en mode interprété. Dans le premier cas, la génération du code produit un module d'accès stocké dans une bibliothèque en vue de son exécution ultérieure. Dans le second cas, c'est un interpréteur qui assure le contrôle dynamique de l'exécution de la requête.

De l'optimisation à la génération du code

4.2.2 Optimisation des requêtes par transformation algébrique

Ainsi qu'il a déjà été vu au chapitre 3, il est possible de combiner les opérateurs de l'algèbre relationnelle. Nous parlons *d'expressions algébriques équivalentes* lorsqu'elles produisent un même résultat malgré les différentes suites d'opérateurs. Les expressions équivalentes nous intéressent dans la mesure où nous pouvons optimiser une interrogation de la base de données par des transformations algébriques sans en modifier le résultat. Elles permettent donc de réduire le coût du traitement des requêtes et constituent ainsi une partie importante du module d'optimisation dans un système de bases de données relationnelles.

Avantage des expressions algébriques équivalentes

La séquence des opérateurs dans une expression algébrique exerce une influence déterminante sur le coût de traitement d'une interrogation. Pour le comprendre, considérons de nouveau notre dernier exemple de requête : nous pouvons substituer à l'expression

$$\begin{aligned} \text{TABLE} &:= \pi_{\text{Ville}} \\ & \left(\sigma_{\text{Description}=\text{Informatique}} \right. \\ & \quad \left. \left(\text{EMPLOYÉ} \mid \times \mid_{\text{Affectation}=\text{D}\#} \text{DÉPARTEMENT} \right) \right) \end{aligned}$$

la forme équivalente suivante, représentée par un arbre d'interrogation dans la figure 4-3 :

$$\begin{aligned} \text{TABLE} &:= \pi_{\text{Ville}} \\ & \left(\pi_{\text{Affectation, Ville}} \left(\text{EMPLOYÉ} \right) \right. \\ & \quad \mid \times \mid_{\text{Affectation}=\text{D}\#} \\ & \quad \pi_{\text{D}\#} \\ & \quad \left. \left(\sigma_{\text{Description}=\text{Informatique}} \left(\text{DÉPARTEMENT} \right) \right) \right) \end{aligned}$$

*Appliquer
tardivement
l'opérateur de
jointure*

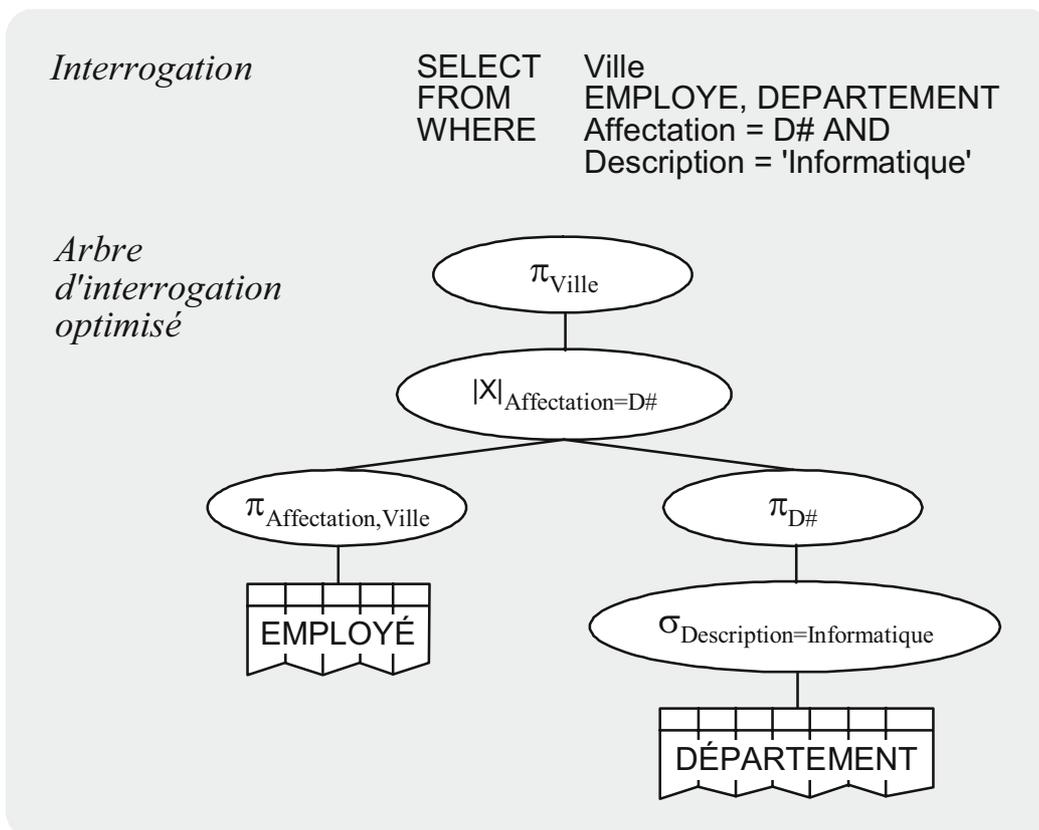
Sous cette forme, primo, la sélection ($\sigma_{\text{Description}=\text{Informatique}}$) s'opère sur la table DÉPARTEMENT, car seul le département Informatique nous intéresse dans le résultat. Secundo, deux projections sont effectuées : la projection ($\pi_{\text{Affectation, Ville}}$) de la table EMPLOYÉ et la projection ($\pi_{\text{D}\#}$) de la table intermédiaire du département Informatique, générée dans la première étape. Tertio, c'est seulement maintenant que se réalise la jointure ($\mid \times \mid_{\text{Affectation}=\text{D}\#}$) d'après le numéro de département, suivie finalement d'une projection sur l'attribut Ville (π_{Ville}). Quoique nous aboutissions au même résultat, le coût du traitement de la deuxième expression équivalente est sensiblement inférieur au coût de l'expression initiale.

*Exécuter les
projections et les
sélections le plus
tôt possible*

En règle générale, nous gagnons en performance en plaçant *les opérateurs de projection et de sélection le plus près possible des «feuilles» dans un arbre d'interrogation*. Cela permet de réduire la taille des tables résultat intermédiaires avant d'exécuter les jointures qui sont des opérations longues, donc onéreuses. Par *optimisation algébrique* nous entendons la mise en place d'une telle stratégie de calcul par transformation d'un arbre d'interrogation. En voici les principes :

- On fusionne en une seule plusieurs sélections sur une même table afin que le prédicat de sélection ne soit testé qu'une fois.
- Les sélections doivent s'effectuer le plus tôt possible afin de réduire la taille des tables résultat intermédiaires. Dans ce but, on place les opérateurs de sélection le plus près possible des feuilles (c'est-à-dire des tables initiales) dans un arbre d'interrogation.
- Les projections doivent aussi s'exécuter le plus tôt possible, mais jamais avant les sélections. Elles réduisent le nombre de colonnes et, la plupart du temps, également le nombre de tuples.
- Les opérateurs de jointure doivent être placés le plus près possible du nœud racine de l'arbre d'interrogation à cause du coût élevé de leur évaluation.

Les principes importants de l'optimisation



*Figure 4-3
Arbre d'interrogation optimisé par transformation algébrique*

Outre l'optimisation algébrique, l'organisation efficace des structures de stockage et d'accès (voir 4.4) permet de réaliser des gains de performance substantiels dans le traitement des requêtes relationnelles. Ainsi, dans un système de bases de données, les opérateurs de sélection et de jointure peuvent être optimisés en

Avantage des structures de données et d'accès efficaces

fonction de la taille des tables, des séquences de tri, des structures d'index, etc. En même temps, il est indispensable de développer un modèle approprié de calcul des coûts d'accès aux données car il faut souvent faire un choix parmi plusieurs variantes de traitement possibles.

Évaluation des stratégies de traitement par des formules de calcul des coûts

Pour évaluer le coût de traitement d'une interrogation de la base de données, nous avons besoin de formules qui prennent en compte des opérations telles que la recherche séquentielle dans une table, la traversée d'un index, le tri des données dans une table ou sous-table, l'utilisation des index sur les attributs de jointure, l'évaluation des opérateurs d'équi-jointure de plusieurs tables. Une telle formule considère le nombre d'accès aux *pages physiques* (*physical pages*, en anglais) et produit un coût pondéré des opérations d'entrée et de sortie, et de la charge du CPU (CPU = Central Processing Unit). L'estimation du coût de traitement d'une requête dépend étroitement de la configuration d'un ordinateur, plus précisément de la puissance du processeur et du temps d'accès à la mémoire centrale, à la mémoire-tampon et aux unités de stockage externes.

4.2.3 Évaluation de l'opérateur de jointure

Différentes stratégies de jointure

Un système de bases de données relationnelles doit comporter les algorithmes nécessaires à l'exécution des opérations de l'algèbre relationnelle ou du calcul relationnel. Contrairement à la sélection des tuples dans une seule table, une sélection effectuée sur plusieurs tables s'avère très coûteuse. C'est pourquoi, dans cette section, nous examinerons de plus près les différentes stratégies de jointure, quoique l'utilisateur occasionnel n'a guère la possibilité d'influer sur les variantes d'exécution.

Stratégies de calcul d'une jointure

La jointure de deux tables consiste à comparer chaque tuple de la première table à tous les tuples de la seconde en vérifiant le prédicat de jointure, et à les coupler en un seul tuple dans la table résultat si le critère de jointure est satisfait. Considérons le calcul d'une équi-jointure pour étudier les différences fondamentales entre deux stratégies : la jointure par boucles imbriquées et la jointure par tri-fusion.

Jointure par boucles imbriquées

Dans la *jointure par boucles imbriquées* (*nested join*, en anglais) d'une table R avec l'attribut A et d'une table S avec l'attribut B, nous comparons *chaque tuple de R à chaque tuple de S* pour vérifier si le prédicat de jointure «R.A=S.B» est satisfait ou pas. Si les tables R et S contiennent respectivement n et m tuples, nous effectuons n fois m comparaisons.

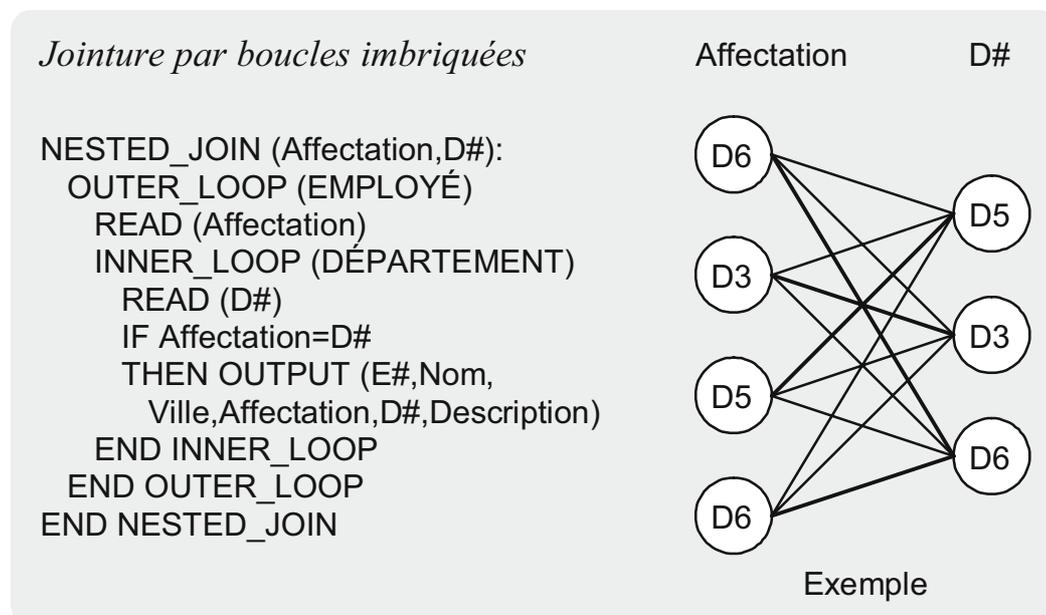
L'algorithme de jointure par boucles imbriquées calcule le produit cartésien de deux tables, et vérifie si le prédicat de jointure est satisfait ou pas. Le coût de l'opération est très élevé, car nous comparons tous les tuples de R dans une boucle extérieure OUTER_LOOP à tous les tuples de S dans la boucle intérieure INNER_LOOP.

La figure 4-4 présente une version simplifiée de l'algorithme de jointure par boucles imbriquées pour répondre à une demande d'informations sur les employés et leurs départements. Le code programmé dans les deux boucles OUTER_LOOP et INNER_LOOP montre à l'évidence cette situation extrême où chaque tuple de la table EMPLOYÉ doit être comparé à tous les tuples de la table DÉPARTEMENT.

Fonctionnement d'une jointure par boucles imbriquées

La jointure par boucles imbriquées s'effectue en deux boucles

Un exemple de jointure



*Figure 4-4
Calcul de la
jointure par
boucles imbriquées*

*Utilité du parcours
d'un index*

Si nous créons un index pour l'attribut A ou B, il est possible de réduire le coût de la jointure par boucles imbriquées. *L'index* (*index*, en anglais) d'un attribut est une structure d'accès qui, dans un ordre de rangement déterminé, associe à chaque valeur de l'attribut une adresse interne où l'enregistrement de données correspondant est stocké. L'index est comparable à celui d'un livre : chaque mot-clé - imprimé dans l'ordre alphabétique - est suivi du numéro de la page où il apparaît dans l'ouvrage.

*Les clés d'accès
sont définies par la
création d'index*

À titre d'exemple nous décidons de doter la table EMPLOYÉ d'un index sur l'attribut Nom. Le système de bases de données crée donc cette structure d'index (voir 4.4) qui reste cependant cachée à l'utilisateur. Chaque nom dans la table EMPLOYÉ, rangé par ordre alphabétique dans l'index, est associé soit à la clé d'identification E# soit à l'adresse interne du tuple de l'employé nommé. Le système de bases de données utilise cet index des noms pour traiter une interrogation ou une jointure impliquant la table EMPLOYÉ. Dans ce contexte, l'attribut Nom est appelé la *clé d'accès*.

*Les structures
d'index améliorent
la performance*

Dans le calcul de la jointure en figure 4-4, l'attribut D# possède aussi son propre index car le numéro de département est la clé primaire¹ de la table DÉPARTEMENT. À chaque itération de la boucle interne, le système de bases de données utilise la structure d'index du numéro de département pour atteindre directement un département recherché au lieu de parcourir toute la table DÉPARTEMENT de tuple à tuple séquentiellement. Dans le meilleur des cas, l'attribut Affectation de la table EMPLOYÉ comporte aussi un index que le système de bases de données peut utiliser à des fins d'optimisation. Cet exemple nous montre le rôle important de l'administrateur de bases de données dans le choix des structures d'index appropriées.

*Avantage des
attributs dont les
valeurs sont triées*

Nous pouvons développer un algorithme plus performant que la jointure par boucles imbriquées si les tuples contenus dans les tables R et S se présentent en ordre croissant ou décroissant des valeurs des

¹ Le système de bases de données crée automatiquement une structure d'index pour chaque clé primaire ; des structures d'index étendues sont définies pour les clés formées par la concaténation de plusieurs attributs.

attributs respectifs A et B formant le prédicat de jointure. Pour cela, avant l'opération de jointure proprement dite, il faut trier les tuples de R, ou de S, ou des deux tables si nécessaire. Ensuite, la jointure consiste simplement à parcourir les tables dans l'ordre croissant ou décroissant des valeurs des attributs A et B utilisés dans le prédicat de jointure, et à les comparer au fur et à mesure. Nous détaillons ci-après cette stratégie.

Jointure par tri-fusion

La *jointure par tri-fusion* (*sort-merge join*, en anglais), basée sur le prédicat de jointure $R.A=S.B$, présuppose que les deux tables R et S sont déjà triées d'après leurs attributs respectifs, A et B. L'algorithme calcule la jointure en *comparant les valeurs de A et B apparues dans leur ordre de tri*. Si les attributs A et B sont définis avec la contrainte d'unicité (par exemple, comme clés primaire et étrangère), le coût de traitement est linéaire.

Fonctionnement de la jointure par tri-fusion

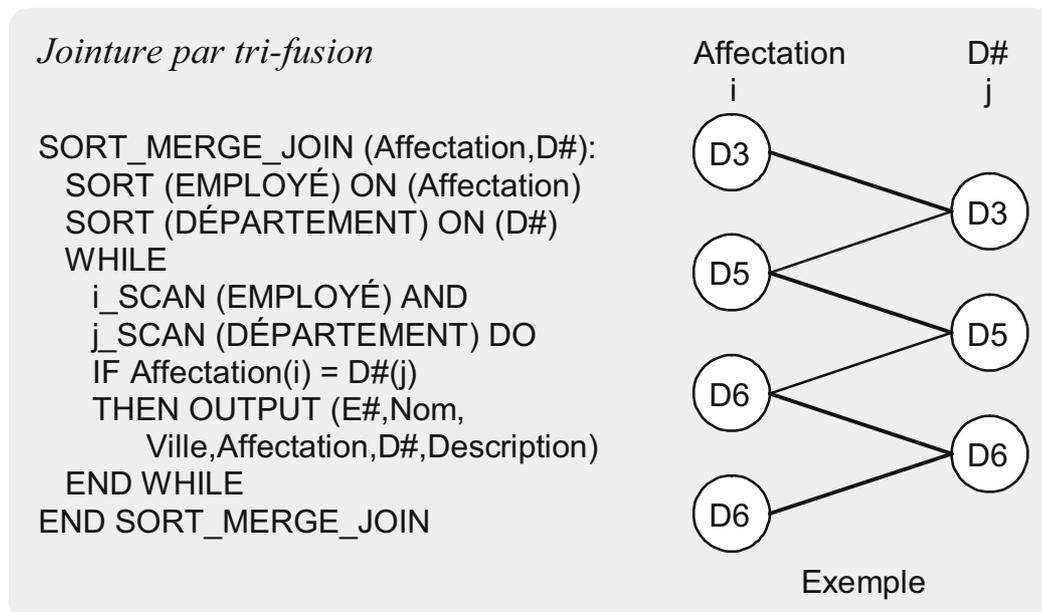


Figure 4-5
Balayage des tables dans l'ordre de tri des attributs définissant le prédicat de jointure

La figure 4-5 présente la forme générale de l'algorithme de jointure par tri-fusion. En premier lieu, les deux tables sont triées sur les attributs présents dans le prédicat de jointure. L'étape suivante consiste à parcourir les deux tables dans l'ordre de tri des attributs A et B en effectuant au fur et à mesure la comparaison $R.A=S.B$. En général, les deux attributs A et B peuvent avoir un lien réciproque complexe, en ce sens qu'une valeur identique peut se répéter plusieurs

Un exemple de jointure par tri-fusion

fois aussi bien dans la colonne R.A que dans la colonne S.B. Si le cas se présente, il est évident que plusieurs tuples de S peuvent être joints à un tuple précis de R, et vice versa. Par conséquent, l'algorithme doit prévoir une sous-jointure imbriquée afin de joindre les tuples correspondants dans R et S pour chaque valeur répétitive des attributs A et B.

Coût linéaire de la jointure par tri-fusion

Dans notre exemple d'interrogation des tables EMPLOYÉ et DÉPARTEMENT, nous constatons que le coût de l'étape tri-fusion est linéairement dépendant du nombre de tuples dans les tables grâce à l'unicité des valeurs de l'attribut clé D#. Les deux tables EMPLOYÉ et DÉPARTEMENT ne seront parcourues qu'une seule fois pour réaliser leur jointure.

La mise à jour des tables système est nécessaire

En principe, le choix d'une stratégie de jointure appropriée - tout comme celui d'une stratégie d'accès - n'est pas une décision a priori que peut prendre un système de bases de données. Contrairement à l'optimisation algébrique, ce choix dépend notamment de l'état courant du contenu d'une base de données. C'est pourquoi il est primordial que des données statistiques contenues dans les tables système soient régulièrement actualisées, de manière périodique ou par l'action de l'expert en bases de données.

4.3 Fonctionnement d'un système de bases de données multi-utilisateur

4.3.1 Le concept de transaction

Le travail sans conflit est assuré par un système multi-utilisateur

Assurer l'intégrité des données est une des exigences fondamentales du point de vue des utilisateurs d'une base de données. La *gestion des transactions* dans un système de bases de données doit *permettre à plusieurs utilisateurs de travailler sans conflit entre eux*. Pour cela, tout changement dans la base de données doit respecter les contraintes d'intégrité définies par les utilisateurs avant qu'il ne leur soit rendu visible.

Définition d'une transaction

Une *transaction* (*transaction*, en anglais) est un ensemble d'opérations soumises aux contraintes d'intégrité, qui met à jour une

base de données en garantissant la cohérence de ses états successifs. En termes précis, une transaction est une suite d'opérations caractérisée par les propriétés d'atomicité, de cohérence, d'isolation et de durabilité.

Atomicité

Une transaction doit être complètement exécutée. Sinon elle ne doit laisser aucune trace de son exécution dans la base de données. Les états intermédiaires générés par les opérations d'une transaction déterminée ne sont pas visibles aux autres transactions concurrentes. Dans ce sens, une transaction constitue une *unité d'exécution dont les opérations doivent être annulées* lorsque la transaction est interrompue (voir 4.5).

L'atomicité obéit au principe du «tout ou rien»

Cohérence

Il est possible qu'une transaction en cours d'exécution transgresse temporairement certaines contraintes de cohérence. Mais quand la transaction est terminée, le résultat final doit respecter de nouveau toutes les contraintes. Par conséquent, une transaction fait passer une base de données d'un état cohérent à un autre état cohérent et garantit l'absence de données contradictoires. La transaction représente donc une *unité d'exécution qui doit maintenir la cohérence d'une base de données*.

La cohérence signifie l'absence de données contradictoires

Isolation

Le principe de l'isolation exige que les résultats générés par des transactions simultanées soient identiques à ceux qu'on aurait pu obtenir dans un environnement à un seul utilisateur. Isolées les unes des autres, les transactions qui s'exécutent en parallèle sont protégées contre des interférences accidentelles. C'est pourquoi la transaction est considérée comme une *unité dont les opérations peuvent être sérialisées* (voir 4.3.2).

L'isolation protège la base de données contre les effets de bord

Durabilité

La durabilité présume la possibilité de reconstruire une base de données

Une base de données doit être maintenue dans un état cohérent jusqu'à la validation des modifications effectuées par une transaction. Lors d'incidents (erreurs de programmation, interruptions du système ou pannes d'unités de stockage externes), la durabilité garantit que seule une transaction correctement achevée permet de valider les changements dans une base de données. Dans le cadre des procédures de redémarrage et de restauration des bases de données, chaque transaction représente une *unité de reprise* (voir 4.5).

Le principe ACID

Les quatre concepts d'*atomicité* (Atomicity), de *cohérence* (Consistency), d'*isolation* (Isolation) et de *durabilité* (Durability) définissent le *principe ACID d'une transaction*. Ce principe fondamental des systèmes de bases de données garantit que chaque utilisateur transforme une base de données d'un état cohérent à un autre état cohérent. Les états intermédiaires incohérents restent invisibles depuis l'extérieur et sont annulés en cas d'incident.

Déclaration des transactions

L'utilisateur déclare une transaction en encadrant les opérations qui la composent par un `BEGIN_OF_TRANSACTION` et un `END_OF_TRANSACTION` respectivement au début et à la fin de la séquence. Ces délimiteurs de début et de fin d'une transaction permettent au système de bases de données d'identifier les opérations qui définissent une unité d'exécution, et de leur appliquer le principe ACID.

4.3.2 La sérialisabilité

Synchronisation des processus concurrents

La coordination (synchronisation) des processus actifs et le conflit des processus concurrents sont des préoccupations majeures dans la conception des systèmes d'exploitation et des langages de programmation. Il en est de même pour les systèmes de bases de données. Les accès simultanés aux mêmes objets d'une base de données doivent être sérialisés pour que ses utilisateurs puissent travailler indépendamment les uns des autres.

Principe de la sérialisabilité

On dit qu'un système traitant des transactions concurrentes est *correctement synchronisé*, si leur exécution séquentielle génère un état identique de la base de données.

Le principe de sérialisabilité garantit que les transactions fournissent des résultats identiques dans la base de données, quel que soit leur mode d'exécution, strictement séquentielle ou en parallèle. Pour définir les conditions de la sérialisabilité, nous devons examiner de près les opérations READ et WRITE de chaque transaction. Ces deux opérations lisent et écrivent des tuples ou enregistrements dans une base de données.

Pour illustrer les transactions concurrentes, considérons un exemple classique tiré du domaine bancaire. Les transactions comptables doivent respecter une contrainte d'intégrité fondamentale qui impose l'égalité des débits et des crédits. La figure 4-6 présente dans l'ordre chronologique les opérations READ et WRITE de deux transactions comptables qui s'exécutent en parallèle. Considérée individuellement, aucune transaction ne modifie le total des comptes a, b et c. Ainsi, la transaction TRX_1 crédite le compte a de 100 unités monétaires et débite en contrepartie le compte b du même montant. De même, la transaction TRX_2 porte au crédit du compte b et au débit du compte c un montant de 200 unités monétaires. Par conséquent, ces deux transactions vérifient la contrainte d'intégrité comptable, car les soldes s'annulent.

En revanche, l'exécution simultanée des mêmes transactions comptables soulève un *conflit* : la transaction TRX_1 ignore le nouveau crédit² $b:=b+200$ calculé par TRX_2 car cette mise à jour n'est pas immédiatement enregistrée. Ceci l'amène à lire une «fausse» valeur du compte b. Quand les deux transactions se terminent avec succès, le compte a contient sa valeur initiale augmentée de 100 unités monétaires ($a+100$), le compte b diminue de 100 unités monétaires ($b-100$) et c de 200 ($c-200$). Le total des débits n'est plus égal à celui des

*Synchronisation
correcte*

*Analyse des
opérations de
lecture et d'écriture*

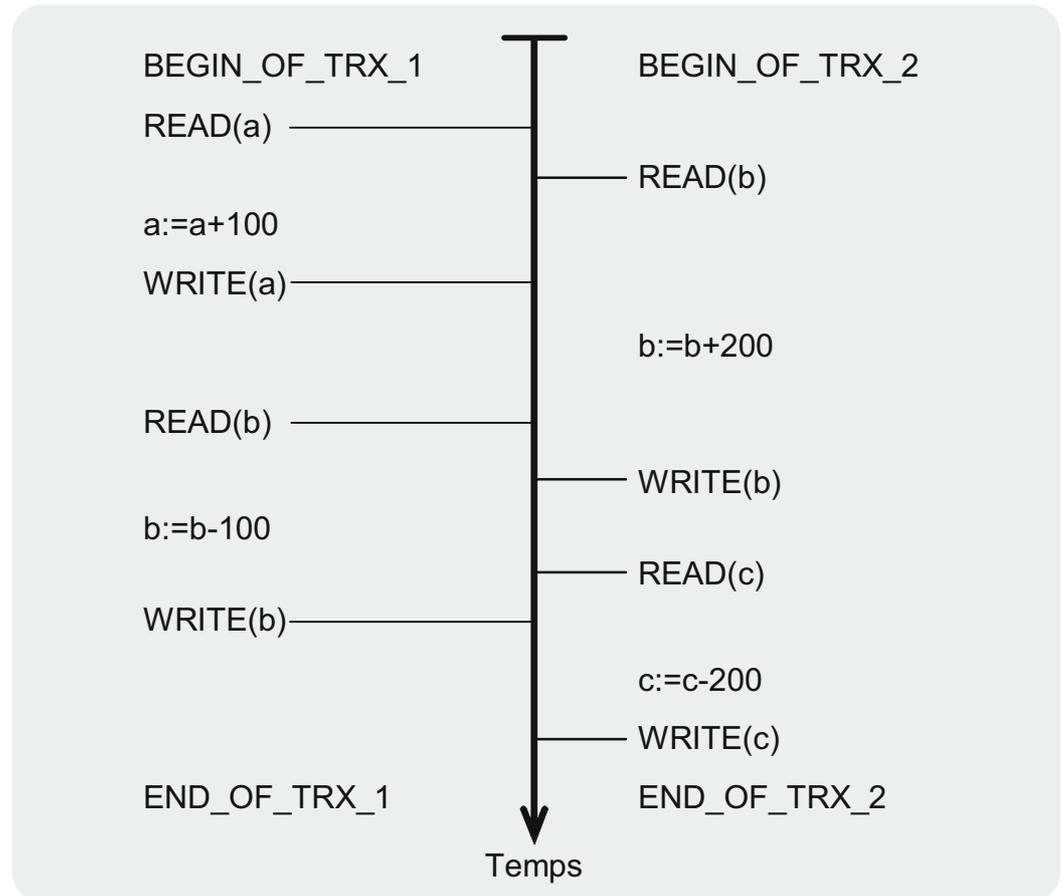
*Les transactions
comptables doivent
maintenir la
cohérence des
comptes*

*L'exécution
parallèle des
transactions donne
lieu à des conflits
potentiels*

² La notation $b:=b+200$ signifie que l'avoir de b augmente de 200 unités monétaires à partir de sa valeur actuelle.

crédits. La contrainte d'intégrité est donc violée car la valeur $b+200$ n'a pas été prise en compte par la transaction TRX_1 à la lecture du compte b .

Figure 4-6
Transactions
comptables
susceptibles de
provoquer des
conflits



Le journal garde la
trace des
opérations de
lecture et d'écriture

Comment détecter les situations de conflit ? L'analyse des transactions révèle que des conflits ont lieu lorsque plusieurs opérations concurrentes READ et WRITE portent sur un objet spécifique qui peut être une valeur de donnée, un enregistrement, une table, voire une base de données toute entière dans le cas extrême. La *granularité* (la taille relative) de cet objet détermine dans quelle mesure il est possible de paralléliser l'exécution des transactions concernées. Plus la granularité d'un objet est grande, plus le degré de parallélisation des transactions diminue, et vice versa. C'est pourquoi toutes les opérations READ et WRITE des transactions portant sur un objet déterminé x sont rapportées dans un *journal* (*log*, en anglais) de cet objet, LOG(x). En d'autres termes, le journal LOG(x) contient la chronologie des accès à un objet x en lecture et en écriture.

Dans l'exemple des transactions comptables simultanées TRX_1 et TRX_2, nous définissons comme granules les comptes a, b et c. A la figure 4-7, le journal de l'objet b contient quatre entrées (voir aussi la figure 4-6). Tout d'abord la transaction TRX_2 lit la valeur de b, TRX_1 lit ensuite la même valeur avant que la transaction TRX_2 enregistre la valeur modifiée de b. Enfin, selon la quatrième et dernière entrée dans le journal, la transaction TRX_1 enregistre la valeur de b qu'elle vient de modifier, écrasant celle enregistrée auparavant par la transaction TRX_2.

Tenue du journal

L'examen du journal permet d'analyser des conflits entre les transactions concurrentes de manière simple. Dans un *graphe de précedence* (*precedence graph*, en anglais) nous représentons les transactions par des nœuds et les conflits potentiels READ_WRITE ou WRITE_WRITE par des arcs orientés (flèches curvilignes). Pour un objet donné, un WRITE qui succède à un READ ou à un WRITE peut être source de conflit. En revanche, plusieurs lectures successives ne sont généralement pas conflictuelles. C'est pourquoi un graphe de précedence ne contient aucun arc orienté READ_READ.

Visualisation par un graphe de précedence

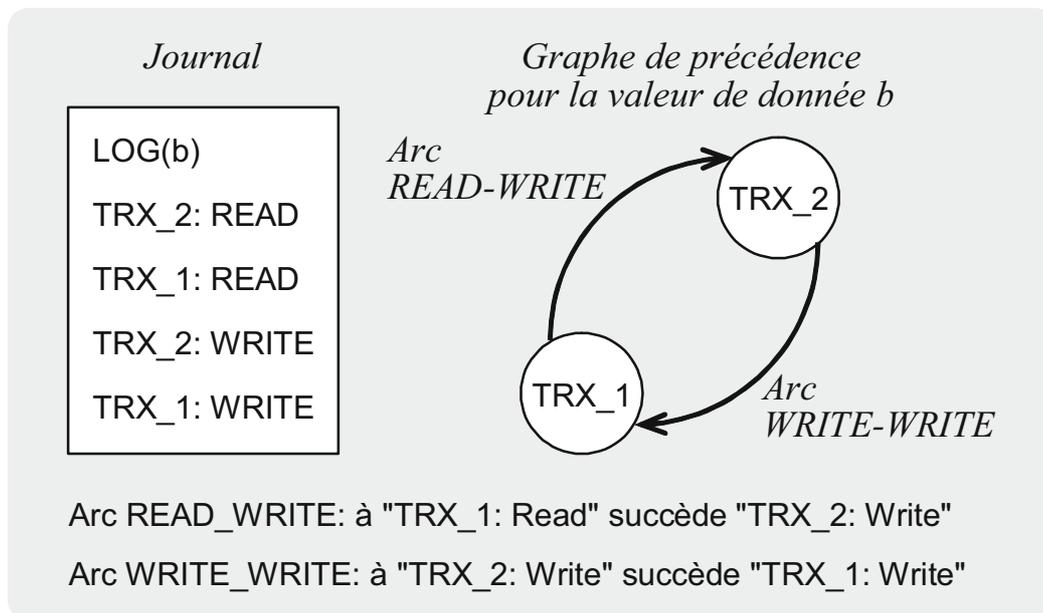


Figure 4-7
Analyse du journal à l'aide d'un graphe de précedence

Dans la figure 4-7, un graphe de précedence a été dessiné à partir du journal de l'objet b pour les transactions comptables TRX_1 et TRX_2. En partant du nœud TRX_1, un READ de l'objet b est suivi d'un WRITE du même objet par la transaction TRX_2. Ces deux opérations successives sont représentées par un arc orienté reliant le nœud

Les arcs READ_WRITE et WRITE_WRITE peuvent générer des conflits

TRX_1 au nœud TRX_2. Dessinons un deuxième arc orienté WRITE_WRITE qui relie TRX_2 au nœud TRX_1, car selon le journal un WRITE de la part de TRX_2 est suivi d'un autre WRITE (du même objet b) provenant de TRX_1. Nous constatons que ce graphe de précedence est cyclique (ou circulaire) : en partant d'un nœud quelconque il existe un chemin orienté qui nous ramène au point de départ. Cette dépendance cyclique entre les transactions TRX_1 et TRX_2 nous permet de conclure qu'elles ne sont pas sérialisables.

Critère de sérialisabilité

Sérialisabilité garantie

Un ensemble de transactions est *sérialisable* si le graphe de précedence correspondant ne contient *aucun cycle*.

Méthodes pessimistes et optimistes

La sérialisabilité d'un ensemble de transactions signifie que leur exécution simultanée dans un système de bases de données multi-utilisateur donne des résultats identiques à ceux obtenus dans un environnement à un seul utilisateur. Pour garantir la sérialisabilité, les *méthodes pessimistes* visent à empêcher le plus tôt possible des conflits entre les transactions simultanées. Les *méthodes optimistes* admettent l'existence des conflits entre transactions, mais cherchent à les résoudre le plus tard possible en annulant les transactions conflictuelles.

4.3.3 Approches pessimistes

Rôle des verrous exclusifs

En verrouillant les objets qu'elle doit lire ou modifier, une transaction peut en bloquer l'accès aux autres transactions. Les *verrous exclusifs* (*exclusive locks*, en anglais) permettent à une transaction de traiter exclusivement un objet déterminé en mettant les autres transactions simultanées en attente jusqu'à ce que l'objet en question soit libéré.

Emploi d'un protocole de verrouillage

Le mécanisme de verrouillage et de déverrouillage est défini par un *protocole de verrouillage* (*locking protocol*, en anglais). Si les verrous sont relâchés trop tôt ou à un moment inapproprié, cela peut déclencher des opérations non sérialisables. Il faut en outre éviter que les transactions se bloquent mutuellement, provoquant ainsi une situation d'interblocage dite «*verrou mortel*» (*deadlock*, en anglais).

Les opérations LOCK et UNLOCK permettent d'acquérir et de relâcher un verrou exclusif pour un objet. En principe, un objet doit être verrouillé avant d'être accédé par une transaction. Dès qu'un objet x acquiert un verrou LOCK(x), il est rendu indisponible aux autres transactions en lecture et en écriture. C'est seulement quand un UNLOCK(x) aura relâché le verrou courant de l'objet x que celui-ci pourra être verrouillé de nouveau par une autre transaction.

Verrouillage et déverrouillages des valeurs

Les verrous fonctionnent en principe selon un protocole bien défini, et ne peuvent donc être acquis ou relâchés de manière incontrôlée.

Protocole de verrouillage à deux phases

Règles pour la pose de verrous

Le *protocole de verrouillage à deux phases (two-phase locking protocol*, en anglais) empêche une transaction de *demander un autre LOCK (verrouiller) après un premier UNLOCK (déverrouiller)*.

Avec ce protocole de verrouillage, une transaction s'exécute toujours en deux phases. Dans la *phase d'acquisition*, la transaction demande tous les verrous nécessaires et les applique aux objets. Le nombre de verrous acquis augmente au fur et à mesure des besoins. Dans la *phase de relâchement*, les verrous sont libérés progressivement. Le nombre de verrous utilisés diminue donc au fur et à mesure. Par conséquent, pour une transaction donnée, la première phase consiste à acquérir des verrous (LOCK) soit en une fois soit au fur et à mesure sans en libérer aucun. C'est seulement dans la phase de relâchement que les verrous sont libérés (UNLOCK) progressivement ou en une fois à la fin de la transaction. Le protocole de verrouillage à deux phases interdit donc de mixer la pose et la levée de verrous.

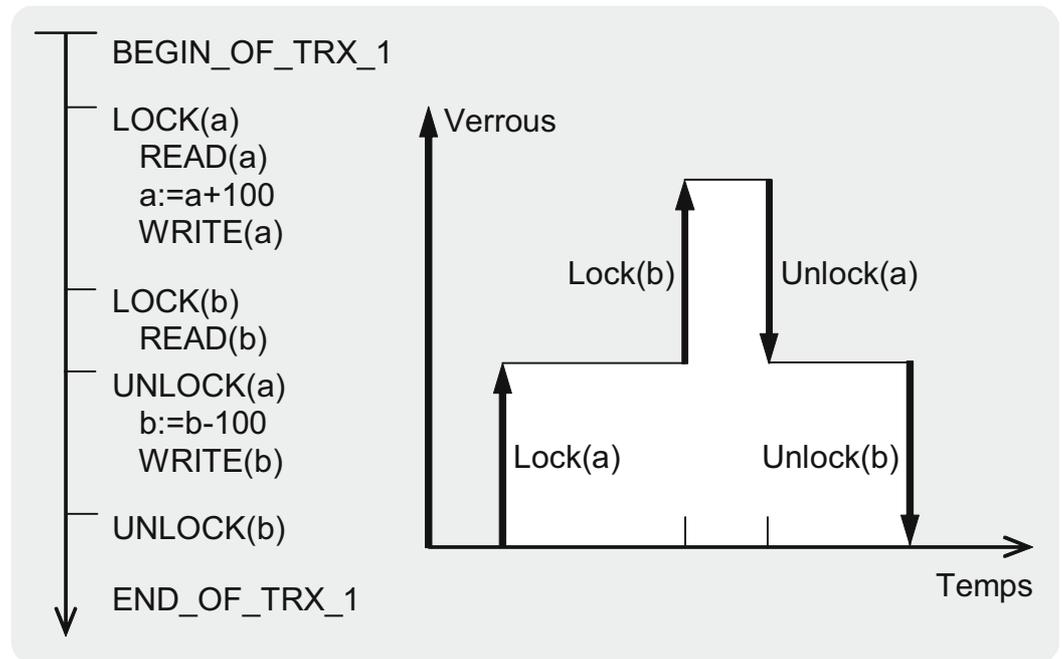
Les phases d'acquisition et de relâchement des verrous

La figure 4-8 illustre une application du protocole de verrouillage à deux phases à la transaction comptable TRX_1. Dans la phase d'acquisition, le compte a et sa contrepartie, le compte b, sont successivement verrouillés. Les verrous sont ensuite libérés l'un après l'autre dans la phase de relâchement. Dans cet exemple, les deux verrous pourraient aussi être acquis tout au début de la transaction et non pas par étapes successives. De la même façon, la levée de ces

Gestion des verrous dans une transaction comptable

deux verrous pourrait se faire en une fois à la fin de la transaction TRX_1 au lieu d'être échelonnée dans le temps.

Figure 4-8
Le protocole de verrouillage à deux phases appliqué à la transaction TRX_1



Augmentation du degré de parallélisation

Grâce au verrouillage progressif des objets a et b durant la phase d'acquisition et à la levée échelonnée des verrous dans la phase de relâchement, le degré de parallélisation de la transaction TRX_1 augmente. En revanche, si les deux verrous étaient acquis au début de la transaction et libérés seulement tout à sa fin, les transactions concurrentes devraient attendre le relâchement des objets a et b pendant toute la durée de traitement de TRX_1.

En général, le protocole de verrouillage à deux phases garantit la sérialisabilité des transactions concurrentes.

Sérialisabilité grâce au protocole de verrouillage

Synchronisation pessimiste (pessimistic concurrency control, en anglais)

Par l'application du protocole de verrouillage à deux phases, tout ensemble de transactions concurrentes est sérialisable.

Les cycles dans les graphes de précedence sont évités

Grâce à la nette séparation des deux phases d'acquisition et de relâchement des verrous, le protocole de verrouillage à deux phases prévient le plus tôt possible les dépendances cycliques dans les graphes de précedence ; les transactions concurrentes s'exécutent sans conflit entre elles. Dans l'exemple de la mise à jour des comptes a, b

et c, cela signifie qu'une gestion correcte de l'obtention et de la levée des verrous permet l'exécution simultanée des transactions TRX_1 et TRX_2 sans violer les contraintes d'intégrité.

La figure 4-9 confirme notre certitude que les transactions concurrentes TRX_1 et TRX_2 s'exécutent sans interférer entre elles grâce au placement des LOCKs et UNLOCKs conformément aux règles du protocole de verrouillage à deux phases. Ainsi par exemple, la transaction TRX_2 est la première à verrouiller le compte b qui ne sera libéré que dans sa phase de relâchement. TRX_1, qui a aussi besoin de verrouiller le compte b, doit donc attendre la levée des verrous. Aussitôt que TRX_2 exécute un UNLOCK(b), le compte b est rendu disponible à TRX_1. À ce moment-là, la transaction TRX_1 lit la valeur «correcte» de b, soit $b+200$. C'est ainsi que les transactions concurrentes TRX_1 et TRX_2 s'exécutent en parfaite synchronisation.

Transactions comptables non conflictuelles grâce aux verrous

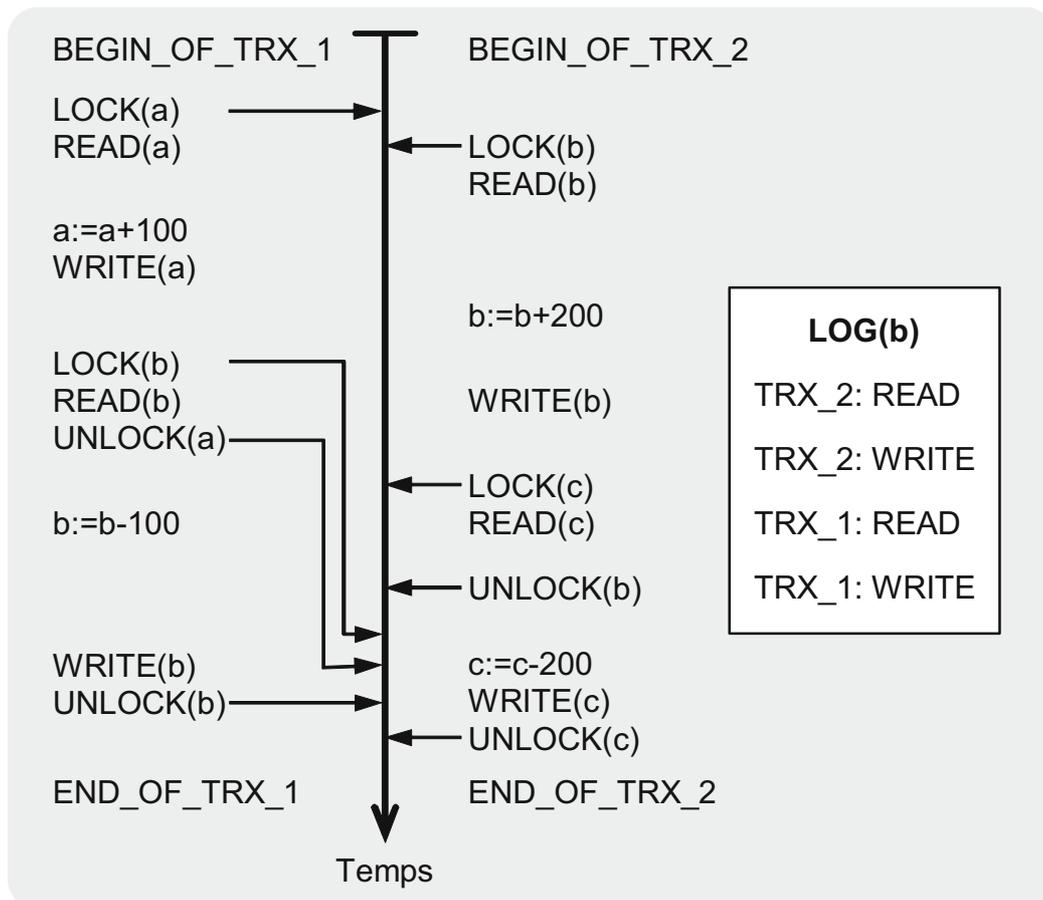


Figure 4-9
Transactions comptables non conflictuelles

Le protocole de verrouillage à deux phases retarde sans doute certaines opérations de TRX_1, mais cet inconvénient est le prix à payer pour qu'à la fin des deux transactions, les contraintes d'intégrité

Un léger retardement est accepté

soient respectées. Le compte a augmente de 100 unités ($a+100$), il en est de même pour b ($b+100$), et le compte c diminue de 200 unités ($c-200$). Par conséquent, le total des débits et crédits reste inchangé.

La comparaison des deux versions du journal LOG(b) du compte b (Figures 4-7 et 4-9) révèle une différence fondamentale : selon la figure 4-9, les deux opérations de lecture (TRX_2: READ) et d'écriture (TRX_2: WRITE) doivent être exécutées en premier par TRX_2, avant que TRX_1 prenne à son tour le contrôle du compte b pour le lire (TRX_1: READ) et le mettre à jour (TRX_1: WRITE). Le second journal LOG(b) donne lieu à un nouveau graphe de précedence qui ne contient aucun arc orienté READ_WRITE ou WRITE_WRITE reliant les nœuds TRX_2 et TRX_1. C'est donc un graphe acyclique. À leur terme, les deux transactions comptables satisfont par conséquent les contraintes d'intégrité.

Granularité des objets à verrouiller

Dans de nombreuses applications de bases de données caractérisées par un haut degré de concurrence des transactions, le verrouillage des tables entières ou de toute la base de données est inadmissible. C'est pourquoi nous devons définir des unités de verrouillage de taille plus petite qui désignent par exemple une section d'une base de données, une sous-table, un tuple ou simplement une valeur de donnée. Dans cette optique, il est avantageux que la *taille des unités de verrouillage* soit déterminée de manière à pouvoir exploiter des *dépendances hiérarchiques* dans la gestion des verrous. Par exemple, si une transaction verrouille un ensemble de tuples, alors, pendant toute la durée du verrouillage, aucune autre transaction ne peut bloquer entièrement les unités de verrouillage parentes telles que les tables ou la base de données correspondante. Lorsqu'un objet acquiert un verrou exclusif, la hiérarchie de verrouillage permet d'évaluer automatiquement les objets parents et de les identifier.

Importance des modes de verrouillage

Outre le verrouillage hiérarchique, il existe d'autres modes de verrouillage importants. La classification la plus simple consiste à distinguer des verrous en lecture et en écriture. Un *verrou en lecture* (*shared lock*, en anglais) accorde à une transaction le droit d'accès à un objet en lecture seule. En revanche, si une transaction demande un *verrou en écriture* (*exclusive lock*, en anglais), elle peut accéder à l'objet verrouillé en lecture et en écriture.

Une autre méthode pessimiste qui garantit la sérialisabilité est celle de l'estampillage : la gestion des accès aux objets repose sur l'ordonnancement des transactions d'après leur ancienneté. La technique d'estampillage permet de respecter l'ordre chronologique des opérations qui composent une transaction et d'éviter ainsi des conflits entre les transactions concurrentes.

*Technique
d'estampillage*

4.3.4 Approches optimistes

Les *approches optimistes* partent de l'idée que les conflits entre transactions concurrentes se produisent rarement. Pour augmenter le niveau de concurrence des transactions et diminuer la durée d'attente, nous renonçons a priori à l'emploi des verrous. Chaque transaction subit un contrôle de validation rétroactif avant de se terminer avec succès.

*Validation
rétroactive*

Avec la *synchronisation optimiste* une transaction s'exécute en trois phases : la phase de *lecture*, la phase de *validation* et la phase d'*écriture*. Aucun verrou n'est appliqué pour prévenir des conflits dans la première phase où tous les objets nécessaires sont lus et traités dans une zone de mémoire de travail privée de la transaction. À la fin du traitement, la transaction entre dans la phase de validation où les objets sont vérifiés pour détecter des mises à jour conflictuelles par rapport aux autres transactions. Cette phase vise à garantir que les transactions actives en ce moment n'ont aucune interférence entre elles. En cas de conflit, la transaction en phase de validation est annulée. Si elle passe avec succès le contrôle de validation, la transaction entre dans la phase d'écriture où les mises à jour stockées dans la zone de mémoire de travail sont appliquées à la base de données.

*Les trois phases de
la synchronisation
optimiste*

Les méthodes optimistes offrent un haut niveau de concurrence grâce aux zones de mémoire de travail privées des transactions. Les accès en lecture ne sont pas conflictuels. C'est au moment où les transactions appliquent des modifications à la base de données que la prudence s'impose. Par conséquent, plusieurs transactions en phase de lecture peuvent s'exécuter simultanément sans avoir besoin de verrouiller les objets. C'est la phase de validation qui vérifie si les

*Accroissement du
degré de
parallélisation*

objets mémorisés dans la zone de travail sont valides, c'est-à-dire s'ils respectent l'intégrité de la base de données.

La date du début de la phase de validation est importante

Admettons pour simplifier que les phases de validation des différentes transactions ne se chevauchent jamais. Nous relevons la date à laquelle une transaction entre en phase de validation. Ce relevé nous permet d'établir une liste des transactions classées par ordre chronologique des dates de début de cette phase. Dès qu'une transaction entre dans la phase de validation, nous vérifions si elle est sérialisable.

Analyser les ensembles d'objets lus et écrits

En mode de synchronisation optimiste, la détermination de la sérialisabilité procède comme suit : soient TRX_t la transaction à vérifier, et TRX_1 à TRX_k les transactions qui s'exécutent en parallèle à TRX_t et qui sont déjà validées pendant la phase de lecture de TRX_t . Le reste n'intervient pas dans le test de sérialisabilité car toutes les transactions sont strictement classées d'après leur date d'entrée dans la phase de validation. En revanche, les objets lus par TRX_t doivent être vérifiés, car dans le même intervalle de temps, ils risquent d'avoir été modifiés par les transactions critiques TRX_1 à TRX_k . Soient $READ_SET(TRX_t)$ l'ensemble des objets lus par TRX_t , et $WRITE_SET(TRX_1, \dots, TRX_k)$ l'ensemble des objets modifiés définitivement par les autres transactions. Le critère de sérialisabilité s'énonce comme suit :

Condition de disjonction des ensembles d'objets lus et écrits

Synchronisation optimiste (optimistic concurrency control, en anglais)

En mode de synchronisation optimiste, les ensembles $READ_SET(TRX_t)$ et $WRITE_SET(TRX_1, \dots, TRX_k)$ doivent être *disjoints* pour que la transaction TRX_t soit sérialisable.

Synchronisation optimiste des transactions comptables

À titre d'exemple, considérons de nouveau les deux transactions comptables TRX_1 et TRX_2 de la figure 4-6 en admettant que TRX_2 a été validée avant TRX_1 . La transaction TRX_1 est-elle sérialisable dans ce cas ? Avant d'y répondre, nous constatons (Figure 4-10) que l'objet b lu par TRX_1 a été modifié par la transaction TRX_2 et appartient à l'ensemble des objets mis à jour $WRITE_SET(TRX_2)$. L'intersection de $WRITE_SET(TRX_2)$ avec l'ensemble des objets lus

READ_SET(TRX_1) est non vide. Par conséquent, le critère de sérialisabilité n'est pas satisfait. Il faut donc annuler la transaction comptable TRX_1 puis la relancer.

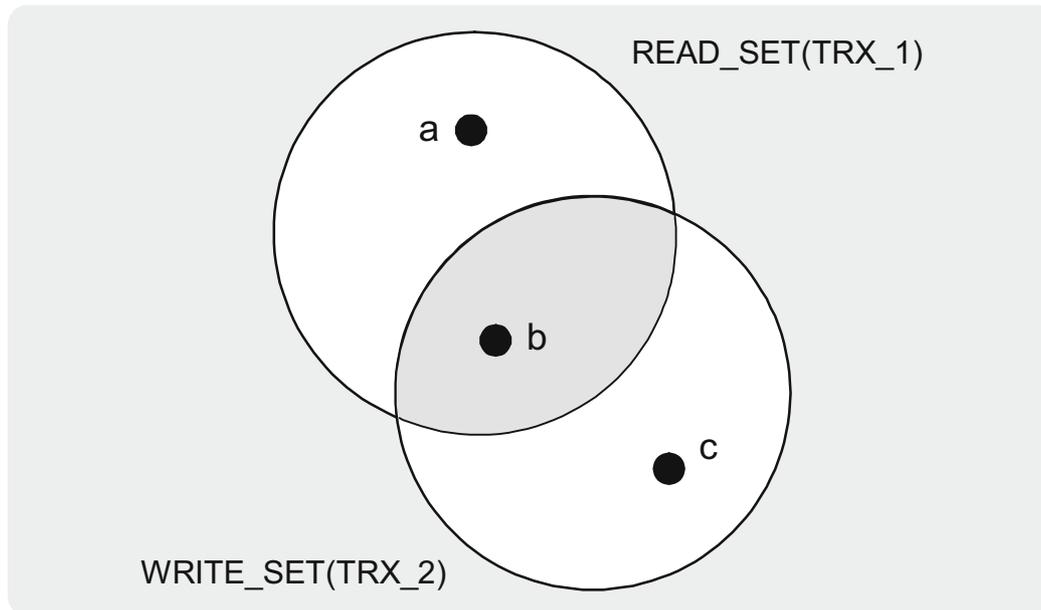


Figure 4-10
La transaction TRX_1 ne satisfait pas la condition de sérialisabilité

Nous pouvons améliorer la méthode optimiste en anticipant la détection des ensembles non disjoints READ_SET et WRITE_SET. À cette fin, dans la phase de validation de la transaction TRX_t, il faut vérifier si celle-ci a modifié éventuellement des objets lus par d'autres transactions concurrentes. Avec cette variante, le coût de validation est déterminé essentiellement par des transactions de mise à jour.

Extension de la méthode

4.4 Structures de stockage et d'accès

4.4.1 Structures arborescentes

Les structures de stockage dans les systèmes de bases de données relationnelles doivent être conçues de manière à gérer efficacement les données stockées sur des unités de mémoire secondaire. Avec des bases de données de grande taille, l'application pure et simple des structures de mémorisation des données résidentes au stockage externe de données sur des unités de mémoire auxiliaire devient problématique. En d'autres termes, nous devons modifier ces structures de mémorisation ou en concevoir de nouvelles afin de

Une gestion efficace de la mémoire auxiliaire est exigée

minimiser le nombre d'accès aux unités de mémoire externe lors des opérations de lecture ou d'écriture de données dans les tables.

Choix de la structure arborescente

Les structures arborescentes conviennent au stockage des clés d'accès ou des enregistrements de données. Lorsque le volume de données devient important, nous associons les nœuds (le nœud racine et les nœuds intermédiaires) et les feuilles d'un arbre non plus à des clés ou des enregistrements de données, mais à des *pages de données* entières (*data pages*, en anglais). Il faut ensuite parcourir l'arbre en question pour localiser un enregistrement de données recherché.

Choix de l'arbre binaire

Pour la gestion des données en mémoire centrale, nous construisons normalement des *arbres binaires* dans lesquels *le nœud racine et chaque nœud intermédiaire se scinde en deux sous-arbres*. Ce type d'arbres ne peut pas être appliqué tel quel aux grandes bases de données pour stocker des clés d'accès ou des enregistrements de données. La profondeur de l'arbre binaire augmente particulièrement vite s'il faut stocker des tables volumineuses. Or, les arbres de grande taille sont inefficaces pour chercher et lire des tables stockées sur des unités de mémoire externe, car ils demandent un nombre d'accès aux pages important.

Comment réduire les accès ?

*La hauteur (ou la profondeur) d'un arbre qui mesure la distance entre le nœud racine et les feuilles détermine le nombre d'accès aux unités de mémoire externe. Pour réduire le plus possible la fréquence des accès externes dans un système de bases de données, nous cherchons à construire des structures de stockage arborescentes qui vont croître en largeur plutôt qu'en profondeur. Une importante structure arborescente de cette catégorie s'appelle *arbre multiple*, dit aussi *arbre B (arbre équilibré)* (Figure 4-11).*

Un arbre multiple comporte plus de deux sous-arbres

Dans un arbre multiple, *le nombre de sous-arbres partant du nœud racine ou d'un nœud intermédiaire est généralement supérieur à deux*. Les pages de données associées aux nœuds intermédiaires ou aux feuilles ne doivent pas rester vides. Il faut autant que possible les remplir de valeurs d'une clé d'accès ou d'enregistrements de données. C'est pourquoi le taux de remplissage des pages, requis la plupart du temps, est d'au moins 50% (à l'exception de la page associée au nœud racine).

*Arbre multiple (B-tree, en anglais)**Définition de l'arbre B*

Les deux propriétés suivantes caractérisent un *arbre multiple*, appelé aussi *arbre B d'ordre n* :

- il est entièrement équilibré (chaque chemin connectant la racine à une feuille quelconque a une même longueur fixe) ;
- chaque nœud (excepté le nœud racine) et chaque feuille de l'arbre possède au moins n mais au plus $2*n$ entrées dans la page de données associée.

La deuxième propriété de l'arbre multiple peut encore s'interpréter comme suit : d'une part, puisque chaque nœud, sauf la racine, contient au moins n entrées, il possède au moins n sous-arbres. D'autre part, chaque nœud admet au plus $2*n$ sous-arbres, car il contient au maximum $2*n$ entrées.

Considérons à titre d'exemple notre table EMPLOYÉ dont la clé est le numéro d'employé E#. Nous utilisons cette clé pour construire une structure d'accès qui sera un arbre multiple d'ordre $n = 2$, représenté dans la figure 4-11. Par conséquent, les nœuds et les feuilles de l'arbre ne doivent pas contenir plus de quatre entrées. Nous admettons implicitement que les pages associées aux nœuds et aux feuilles contiennent non seulement les valeurs de la clé choisie, mais aussi des pointeurs qui renvoient aux pages de données où sont stockés les enregistrements de données proprement dits. L'arbre construit à la figure 4-11 représente donc un arbre d'accès, et non point une technique de gestion des tuples ou enregistrements de données dans la table EMPLOYÉ.

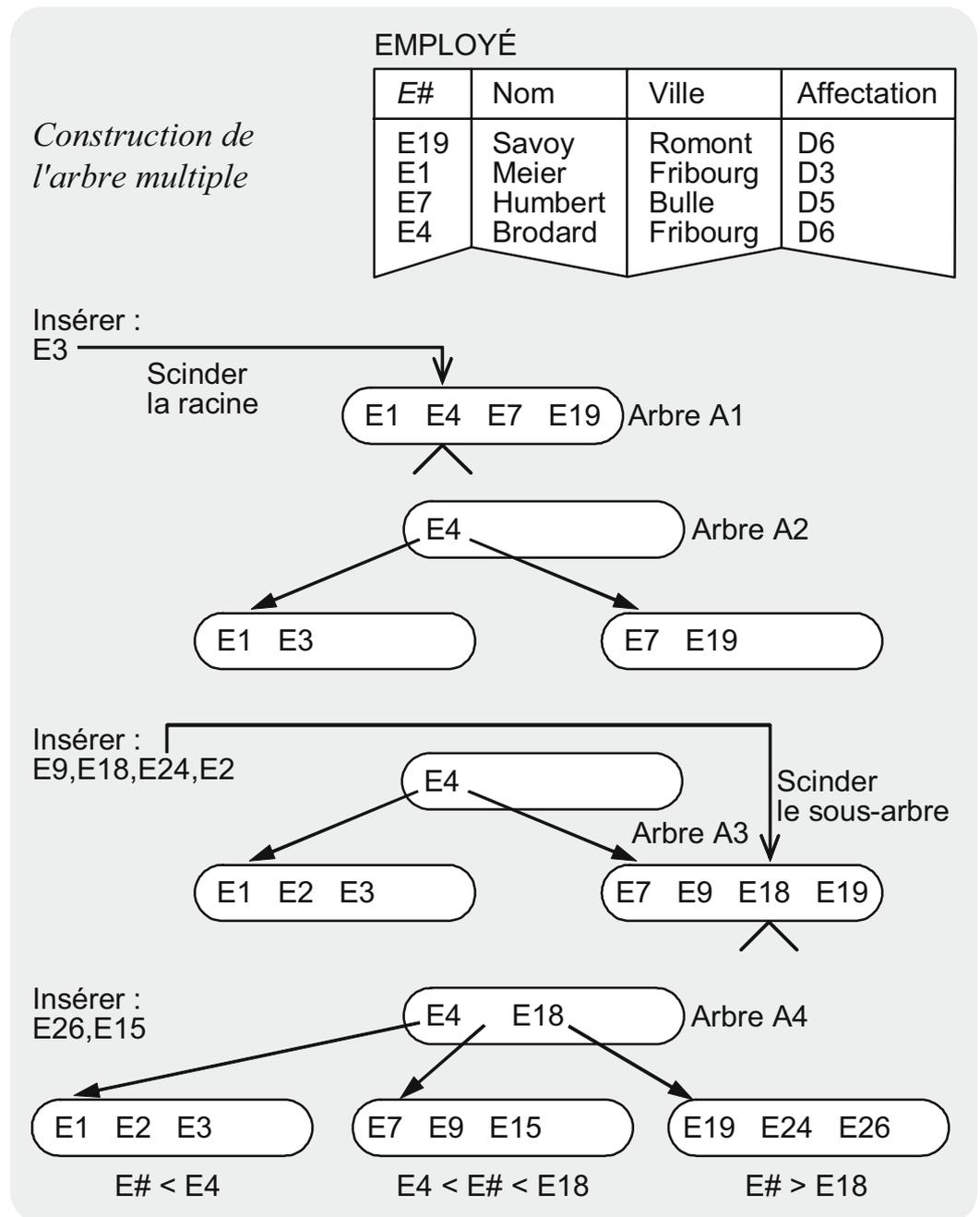
Exemple d'un arbre B d'ordre 2

Dans notre exemple, le nœud racine de l'arbre multiple A1 contient les quatre valeurs de la clé E# en ordre croissant, E1, E4, E7 et E19. Pour insérer une nouvelle valeur de clé E3, nous devons scinder le nœud racine parce qu'il n'admet plus d'entrées additionnelles. L'éclatement doit s'effectuer de manière à produire un arbre équilibré. La valeur de clé E4 reste associée au nœud racine, car elle partage l'ensemble des valeurs restantes en deux parties égales. Le sous-arbre de gauche contient des valeurs de clé telles que «E# est inférieur à E4» (c'est-à-dire E1 et E3 dans notre cas) tandis que le

Construction d'un arbre équilibré

sous-arbre de droite contient des valeurs satisfaisant la condition «E# est supérieur à E4» (c'est-à-dire E7 et E19). Nous procédons de la même manière pour insérer d'autres valeurs de clé sans changer la hauteur de l'arbre.

Figure 4-11
Mise à jour
dynamique d'un
arbre multiple



Processus de recherche

Pour rechercher une valeur de clé particulière, le système de bases de données procède comme suit : soit E15 la valeur à rechercher dans l'arbre multiple A4 à la figure 4-11. Le système la compare tout d'abord aux entrées du nœud racine. E15 est comprise entre les valeurs de clé E4 et E18, le système choisit donc le sous-arbre

approprié (qui est ici une feuille) et continue sa recherche. Finalement, il trouve l'entrée désirée dans la feuille choisie. Dans cet exemple simple, le coût de recherche de la valeur de clé E15 équivaut seulement à deux accès pour atteindre d'abord la page associée au nœud racine et ensuite la page associée à une feuille.

La hauteur de l'arbre multiple détermine le temps d'accès à une valeur de clé ainsi qu'aux données identifiées par cette valeur. Il est possible de réduire le temps d'accès en augmentant le nombre de valeurs de clé (degré de branchement) par nœud dans un arbre multiple.

Une autre méthode consiste à construire un *arbre multiple orienté feuilles* (aussi connu sous le nom d'*arbre B**). Dans cette structure, les enregistrements de données ne sont jamais stockés dans des nœuds intérieurs mais toujours dans les feuilles de l'arbre. Tous les nœuds contiennent uniquement des valeurs de clé afin de réduire le plus possible la profondeur de l'arbre.

La hauteur de l'arbre détermine le temps d'accès

L'arbre B est orienté feuilles*

4.4.2 Méthodes de hachage

Les méthodes de transformation de clés ou de calcul des adresses (*key hashing*, ou simplement *hashing*, en anglais) constituent le fondement des structures d'accès et de stockage aléatoire. Une *fonction de hachage* (*hash function*, en anglais) permet de transformer un ensemble de clés en un ensemble d'adresses qui forme un espace d'adressage contigu.

Stockage utilisant la technique d'adressage dispersé

Une méthode de transformation simple consiste à associer une adresse, représentée par un nombre naturel de 1 à n, à chaque clé dans une table. Cette adresse est interprétée comme un numéro de page relatif. Chaque page contient un nombre fixe de valeurs de clé, avec ou sans les enregistrements de données correspondants.

Une fonction de hachage simple

La transformation des clés doit satisfaire les critères suivants :

- la méthode de transformation doit comporter des opérations simples et non onéreuses ;

Conditions imposées au calcul des adresses

- les adresses réservées doivent être uniformément distribuées dans l'espace d'adressage ;
- la probabilité d'associer des mêmes adresses à plusieurs clés doit être uniforme pour toutes les valeurs de clé.

Il existe une grande variété de fonctions de hachage avec leurs avantages et leurs inconvénients. La méthode la plus connue et la plus simple est le hachage utilisant le reste d'une division.

Hachage par la méthode de la division

Fonction de hachage par la méthode de la division

Chaque clé est interprétée comme un nombre naturel. *Étant donné une clé k et un nombre premier p , la fonction de hachage H qui transforme la clé s'exprime par la formule suivante :*

$$H(k) := k \bmod p.$$

La division de la valeur de clé k par le nombre premier p donne un reste entier « $k \bmod p$ » qui désigne une adresse relative ou un numéro de page relatif. Avec cette méthode, le choix du nombre premier p détermine l'occupation de la mémoire et le degré d'uniformité de la distribution des adresses à l'intérieur de l'espace d'adressage.

Exemple de création des classes par le reste de la division

Dans la figure 4-12, les valeurs de la clé $E\#$ dans la table EMPLOYÉ sont insérées dans les différentes pages par la méthode de la division. Pour cet exemple simple, nous supposons que chaque page peut recevoir quatre valeurs de clé. Nous choisissons le nombre premier 5. Chaque valeur de clé est ensuite divisée par 5, donnant un reste entier qui détermine le numéro de page.

Au moment d'insérer la valeur de clé $E14$, une collision a lieu car l'adresse calculée pointe vers une page qui est déjà saturée. C'est pourquoi nous plaçons $E14$ dans une *zone de débordement*. Le chaînage de la page 4 à la zone de débordement garantit l'appartenance de la valeur de clé $E14$ à la classe 4 (reste de la division de 14 par 5).

Traitement des débordements

Il existe plusieurs techniques pour traiter le problème de débordement. Au lieu de créer une zone de débordement, nous

pouvons appliquer des transformations répétées à la clé en question. Le traitement du débordement se complique lorsque le domaine des valeurs d'une clé croît rapidement ou qu'il faut exécuter un grand nombre de suppressions. Pour surmonter ces problèmes, des méthodes de transformation de clés par hachage dynamique ont été développées.

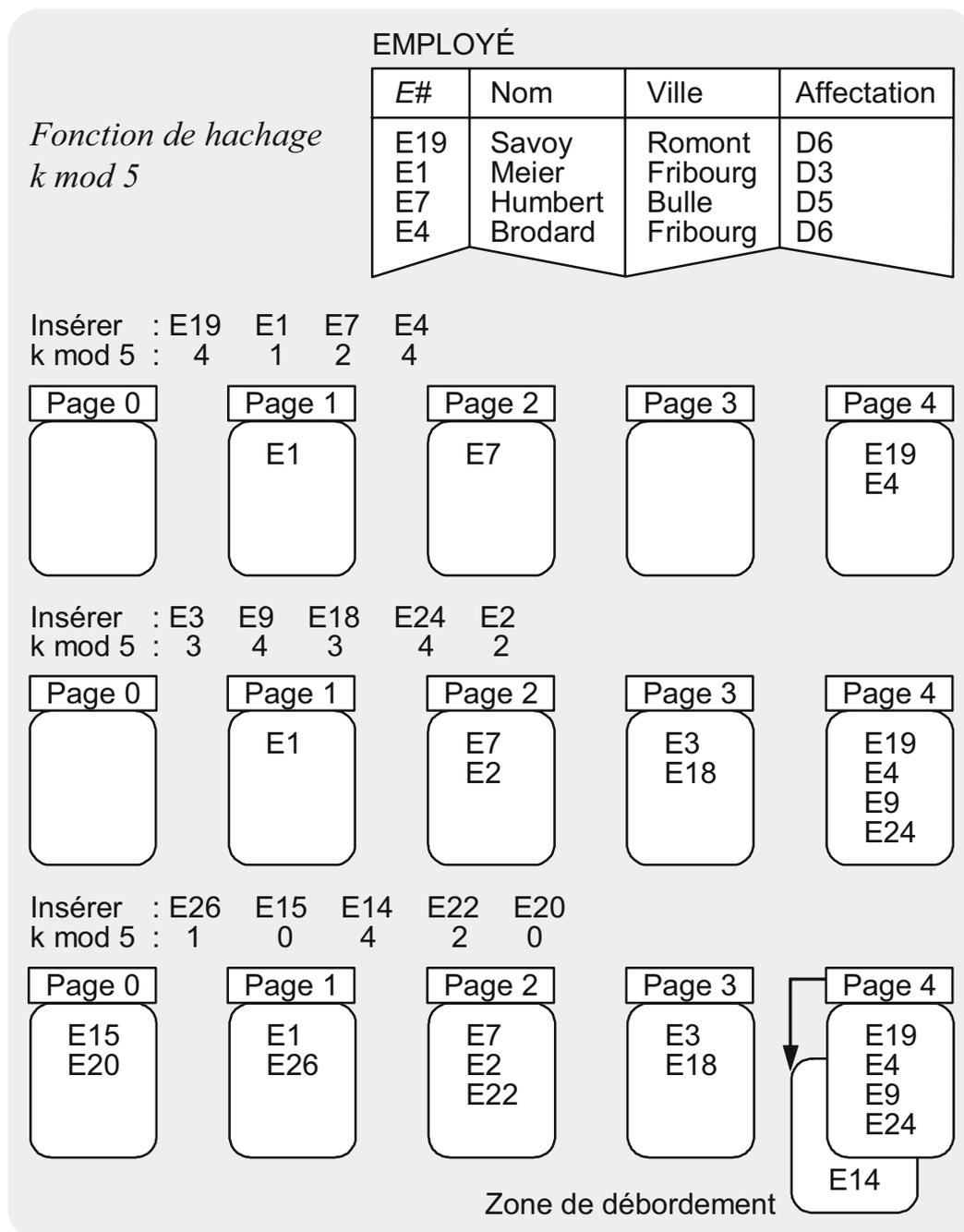


Figure 4-12
Hachage par la création de classes selon le reste de la division

Dans les *méthodes de hachage dynamique*, nous cherchons à rendre l'occupation de la mémoire indépendante de la croissance du

Hachage dynamique

nombre de valeurs d'une clé. Selon la méthode choisie, nous pouvons abandonner la gestion des zones de débordement ou la réorganisation globale des adresses. Avec les méthodes de hachage dynamique, un espace d'adressage existant peut être élargi soit grâce au choix judicieux d'une technique de transformation des clés, soit par l'emploi d'une table d'allocation de pages qui réside dans la mémoire centrale, sans avoir besoin de recharger toutes les valeurs de clé ou tous les enregistrements de données déjà stockés.

4.4.3 Structures de données multidimensionnelles

Structures de données avec clés multidimensionnelles

Les structures de données multidimensionnelles permettent l'accès aux enregistrements de données basé sur plusieurs valeurs de clés d'accès. Nous appelons *clé multidimensionnelle* l'ensemble de ces clés d'accès. Elle doit être unique, mais n'a pas besoin d'être toujours minimale. Par *structure de données multidimensionnelle* (*multi-dimensional data structure*, en anglais) nous entendons une structure de données qui admet une clé multidimensionnelle. Par exemple, à partir de la table EMPLOYÉ nous pouvons définir une structure de données bidimensionnelle dont la clé d'accès se compose de deux attributs : le numéro d'employé et l'année d'entrée en service. Le numéro d'employé qui forme la première partie de la clé bidimensionnelle est toujours unique. L'attribut «Année d'entrée en service» qui en constitue la deuxième partie sert de clé d'accès additionnelle dont les valeurs ne sont pas nécessairement uniques.

Condition de symétrie

À l'inverse des structures arborescentes, dans les structures de données multidimensionnelles aucune partie de clé ne doit imposer la séquence de stockage des enregistrements de données physiques. Une structure de données multidimensionnelle est *symétrique* si elle permet l'accès à partir de plusieurs clés sans donner la préférence à une clé spécifique ou une combinaison de clés particulière. À titre d'exemple, pour notre table EMPLOYÉ il faudrait que les deux composants de la clé bidimensionnelle (le numéro d'employé et l'année d'entrée en service) aient le même poids et garantissent l'efficacité des accès lors d'une requête d'interrogation.

Une importante structure de données multidimensionnelle s'appelle *fichier grille* (*grid file*, en anglais).

Fichier grille

Un fichier grille est une structure de données multidimensionnelle qui possède les propriétés suivantes :

Définition du fichier grille

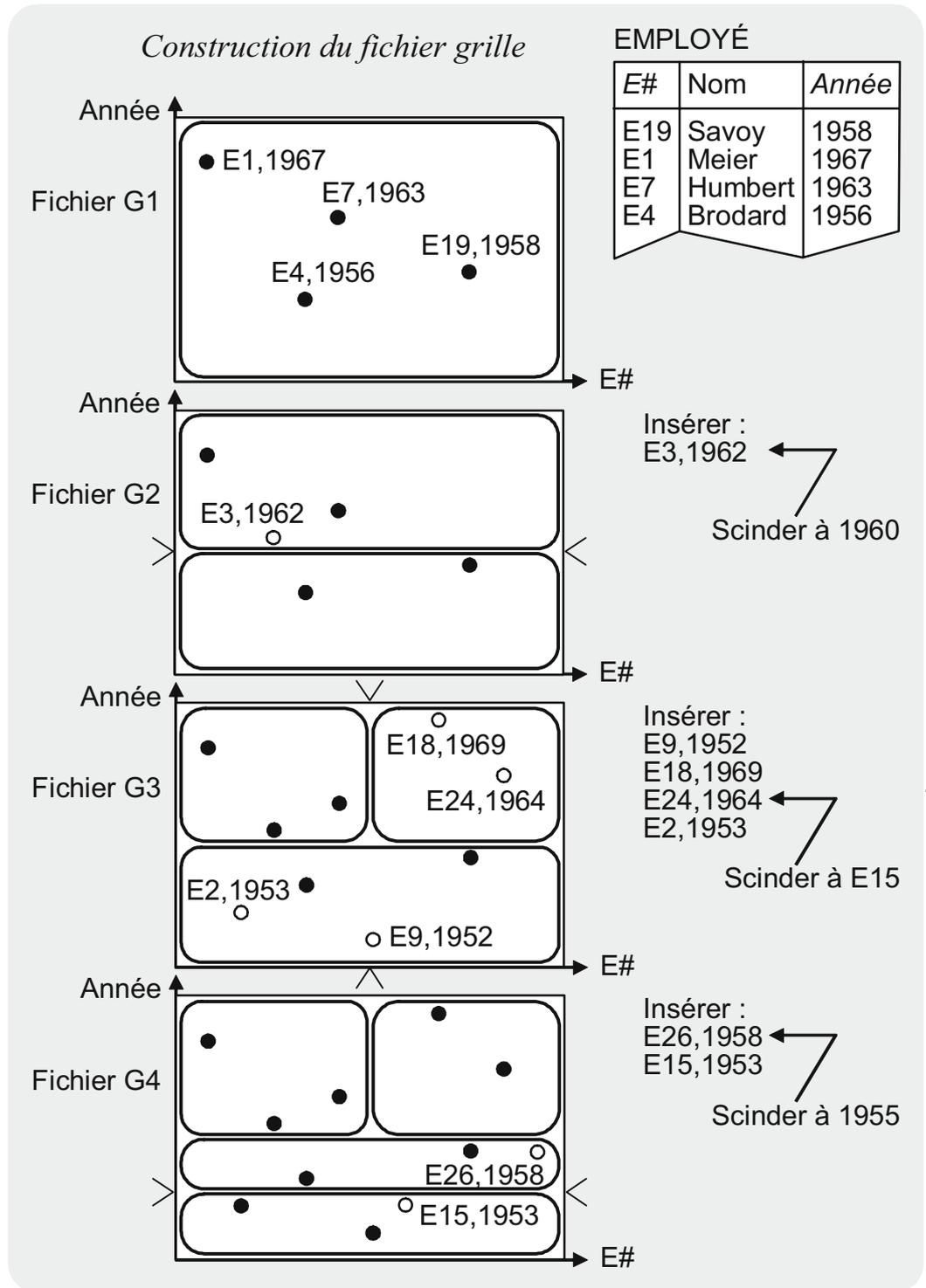
- elle permet l'accès à partir d'une *clé multidimensionnelle* de manière symétrique, en ce sens qu'aucune dimension n'est dominante ;
- elle permet de lire un enregistrement de données quelconque *après deux accès*, le premier pour atteindre un index grille, le deuxième pour atteindre la page de données recherchée.

Un fichier grille consiste en un index grille et un fichier contenant des pages de données. L'index grille représente un espace multidimensionnel où chaque dimension correspond à une partie de la clé d'accès multidimensionnelle. Pour insérer des enregistrements de données, l'index est subdivisé en cellules dans chaque dimension en alternance. À la figure 4-13, dans l'exemple avec une clé d'accès bidimensionnelle, l'éclatement des cellules se fait alternativement selon le numéro d'employé et l'année d'entrée en service. Les points de subdivision ainsi obtenus dans chaque dimension constituent la graduation de l'index grille.

Index grille

Chaque *cellule de l'index grille* correspond à une page de données et contient au moins n mais au plus $2*n$ entrées. Les cellules vides de l'index grille doivent être fusionnées en cellules plus grandes afin que les pages de données associées contiennent le minimum d'entrées requis. Comme dans le précédent exemple illustrant la méthode de la division, nous admettons de nouveau que chaque page de données contient au maximum quatre entrées ($n = 2$).

Figure 4-13
Partitionnement
dynamique d'un
index grille



Emploi de la
graduation dans
l'index grille

Comme l'index grille est généralement très volumineux, il doit être stocké dans une unité de mémoire auxiliaire comme les enregistrements de données. En revanche, l'ensemble des points de subdivision est de petite taille et peut donc résider en mémoire centrale. Chaque accès à un enregistrement de données spécifique procède comme suit : tout d'abord, à partir de k valeurs de clés dans

un fichier grille à k dimensions, le système parcourt la graduation pour déterminer l'intervalle où se trouve la valeur de chaque attribut constituant la clé de recherche. Les intervalles ainsi connus permettent d'accéder directement à une section précise de l'index grille. Chaque cellule de l'index contient le numéro de la page où sont stockés des enregistrements de données. Un deuxième accès à la page de données indiquée permet finalement de savoir si la page contient l'enregistrement de données recherché ou non.

Le principe des deux accès aux pages est garanti pour rechercher un enregistrement de données quelconque dans un fichier grille. Plus précisément, l'opération de recherche requiert au maximum deux accès aux pages stockées dans la mémoire secondaire : le premier pour atteindre la bonne section dans l'index grille, le second pour accéder à la bonne page de données.

Deux accès garantis pour rechercher un enregistrement

Par exemple, la recherche de l'employé N° E18 engagé en 1969, se déroule de la manière suivante dans le fichier grille G4 à la figure 4-13 : le numéro d'employé E18 est compris dans l'intervalle E15 à E30, plus exactement dans la moitié droite du fichier grille. L'année d'entrée en service 1969 est comprise entre 1960 et 1970 et se trouve donc dans la moitié supérieure. En utilisant ces deux indications sur la graduation, le système de bases de données effectue un premier accès à l'index grille pour obtenir l'adresse de la page de données. Un deuxième accès atteint cette page qui contient les enregistrements de données recherchés, identifiés par les clés d'accès (E18,1969) et (E24,1964).

Un exemple de recherche

Un fichier grille à k dimensions permet de traiter des interrogations pour extraire soit un enregistrement de données particulier, soit un ensemble d'enregistrements appartenant à un domaine spécifique. Dans une *requête singulière* (*point query*, en anglais), le résultat obtenu est un enregistrement de données identifié par k clés d'accès. Il est possible de formuler une requête singulière partielle en spécifiant seulement une partie de la clé. Une *requête d'intervalle* (*range query*, en anglais) permet d'effectuer la recherche sur un domaine particulier de chacune des k parties de la clé. Le résultat comprend tous les enregistrements de données pour lesquels les valeurs des attributs constituant la clé appartiennent aux domaines

Soutien efficace aux requêtes d'intervalle

désirés. Il est aussi possible de spécifier des domaines seulement pour une partie de la clé, nous parlons alors de requête d'intervalle partielle.

Réduction de l'espace de recherche

La recherche de l'enregistrement de données (E18,1969), expliquée précédemment, est un exemple de requête singulière. Si nous connaissons seulement l'année d'entrée en service, nous recherchons l'employé par une requête singulière partielle en spécifiant l'année 1969. Nous formulons une requête d'intervalle (partielle) lorsque nous désirons connaître par exemple tous les employés engagés entre 1960 et 1969. En retournant à la figure 4-13, nous nous trouvons alors dans la moitié supérieure de l'index grille G4, et la recherche se limite donc aux deux pages de données dans cette partie de l'index. Ainsi, grâce au fichier grille multidimensionnel, nous obtenons des résultats d'une requête d'intervalle, partielle ou non, sans devoir parcourir tout le fichier.

Soutien aux systèmes d'information géographique

Au cours des dernières années, des recherches sont menées pour étudier et proposer différentes structures de données multidimensionnelles, capables d'implanter efficacement des clés d'accès multi-attributs de manière symétrique. Malgré l'application encore limitée, à l'heure actuelle, des structures de données multidimensionnelles dans les systèmes de bases de données relationnelles, elles jouent un rôle de plus en plus important dans les processus de recherche d'information orientés web. Les systèmes d'information géographique, topologique et géométrique ont particulièrement besoin de telles structures pour le traitement efficace des requêtes.

4.5 Traitement des erreurs

Variété des sources d'erreurs

Pendant l'exploitation d'une base de données, des erreurs de tous genres peuvent survenir, et normalement un système de gestion de bases de données est capable de les corriger. Dans le cadre de notre exposé sur la gestion des transactions concurrentes, nous avons déjà parlé des anomalies qui entraînent la violation des contraintes d'intégrité ou l'interblocage des transactions. Un système d'exploitation ou un matériel informatique peut aussi provoquer

d'autres types d'incidents. Par exemple, une unité de stockage externe qui tombe en panne peut rendre illisible les données dans une base.

Le concept de *reprise* (*recovery*, en anglais) vise à *restaurer une base de données dans un état cohérent après une panne*. Pour y parvenir, il est crucial de savoir où s'est produite l'erreur. Est-elle causée par un programme d'application, par le logiciel de base de données ou par un matériel défectueux ? En cas de violation des contraintes d'intégrité ou de «crash» d'un programme d'application, il suffit de défaire une, voire plusieurs transactions, et de les refaire ensuite après avoir apporté des corrections nécessaires. En cas d'incident grave, la corruption des données peut être telle que nous devons retrouver, dans un cas extrême, une copie de sauvegarde et, à partir de là, reconstruire la base de données en rejouant les transactions depuis la dernière sauvegarde.

Restauration d'une base de données

Pour défaire les transactions, le système de bases de données s'appuie sur un certain nombre d'informations de reprise. En principe, avant toute modification d'un objet, le système en crée une copie dite *image avant* (*before image*, en anglais) dans un *journal de transaction*³ (*log file*, en anglais). Outre la sauvegarde des anciennes valeurs de l'objet, le système génère dans le journal des marques qui signalent le début et la fin d'une transaction.

Mécanisme de journalisation

Pour que le journal de transaction puisse être exploité efficacement lors de la reprise après panne, nous définissons des *points de reprise* (*checkpoints*, en anglais), appelés aussi *points de sauvegarde*, ou encore *points de cohérence*, soit par le biais d'instructions appropriées dans un programme d'application, soit en les associant à des événements particuliers dans le système. Un point de reprise du système comporte une liste des transactions actives jusqu'à ce moment précis. À chaque *redémarrage* (*restart*, en anglais), le système de bases de données se fonde sur le dernier point de reprise pour défaire les transactions inachevées (en exécutant par exemple la commande SQL ROLLBACK).

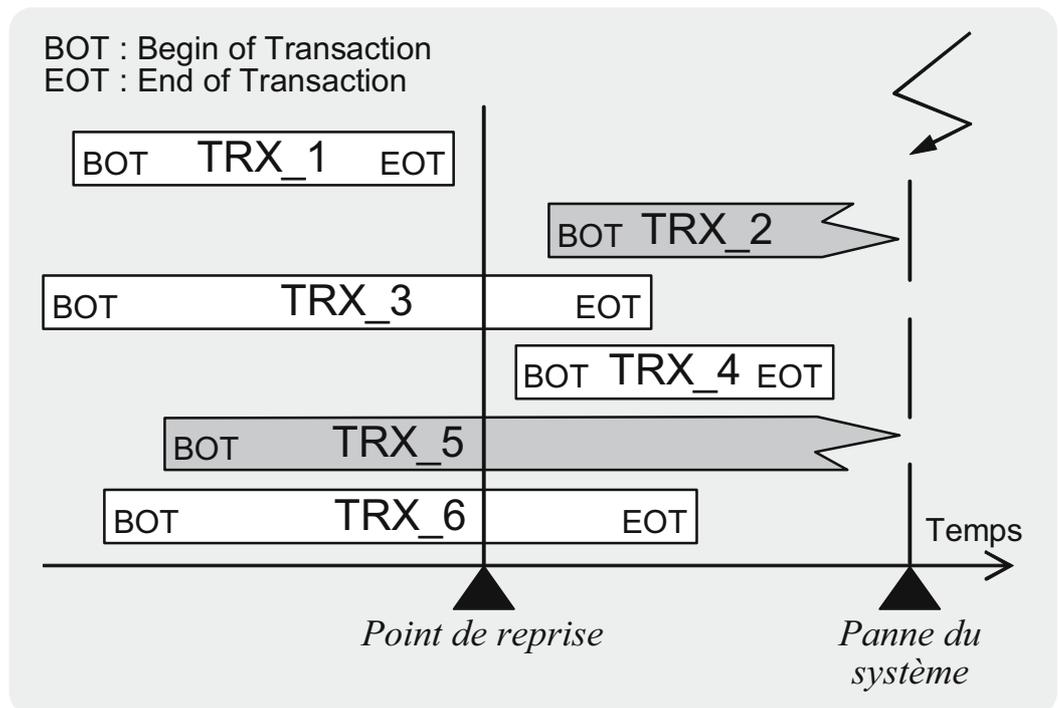
Pose des points de reprise

³ Dans le présent contexte, ce journal ne doit pas être confondu avec un autre journal défini dans la section 4.3.

Technique d'annulation des transactions

Le mécanisme de l'annulation des transactions est illustré par la figure 4-14 : après une panne, le système parcourt le journal depuis la fin jusqu'au point de reprise le plus récent. Les transactions qui retiennent l'attention sont celles qui ne portent pas encore la marque EOT (EOT = End Of Transaction) signalant leur achèvement normal. Il s'agit des deux transactions TRX_2 et TRX_5 qu'il faut maintenant *défaire* (*undo*, en anglais) à l'aide du journal afin de rétablir l'ancien état de la base de données. À cette fin, le système part du point de reprise et remonte jusqu'à la marque BOT (BOT = Begin Of Transaction) de la transaction TRX_5 pour retrouver l'image avant de TRX_5. Indépendamment de la nature du point de reprise, le système doit aussi régénérer (*redo*, en anglais) l'état le plus récent (*after image*, en anglais) produit au moins par la transaction TRX_4.

Figure 4-14
Reprise d'un
système de bases
de données après
une panne



Organisation de la sauvegarde

Après une panne, pour reconstruire une base de données sur des unités de mémoire externe, nous devons disposer d'une copie d'archive de la base de données et de l'ensemble des modifications exécutées depuis la date de sauvegarde. Les copies d'archive sont normalement effectuées avant et après les travaux de traitement exécutés en fin de journée, car l'opération est très gourmande en temps. Durant la journée, c'est l'enregistrement des modifications

dans le journal de transaction qui détermine pour chaque objet son état le plus récent.

L'expert en bases de données est chargé de mettre en œuvre un *plan de reprise après sinistre* qui définit clairement des procédures en vue de garantir la sécurité des bases de données. En principe, il faut maintenir un certain degré de redondance en créant plusieurs générations de copies de sauvegarde physiquement séparées. La disponibilité des copies d'archive et l'élimination des versions obsolètes doivent être journalisées de manière systématique. En cas de panne ou lors des simulations de sinistre, cela permet de restaurer l'ensemble des données courantes dans un délai convenable à partir des copies de sauvegarde d'une part, et des modifications archivées de la base de données d'autre part.

*Importance de la
prévention des
sinistres*

4.6 Architecture détaillée du système

Le principe de localité des modifications et des extensions futures est fondamental à l'architecture interne des systèmes de bases de données. Comme dans l'implantation des systèmes d'exploitation ou d'autres composants logiciels, les systèmes de bases de données relationnelles sont conçus d'après une *architecture en plusieurs couches indépendantes* qui communiquent entre elles au travers d'interfaces prédéfinies.

*Nécessaire
indépendance des
différentes couches
du système*

La figure 4-15 donne une vue d'ensemble de l'architecture du système, décomposée en cinq niveaux. Nous énumérons ci-après les fonctions essentielles de chaque couche, qui ont été traitées dans les sections précédentes.

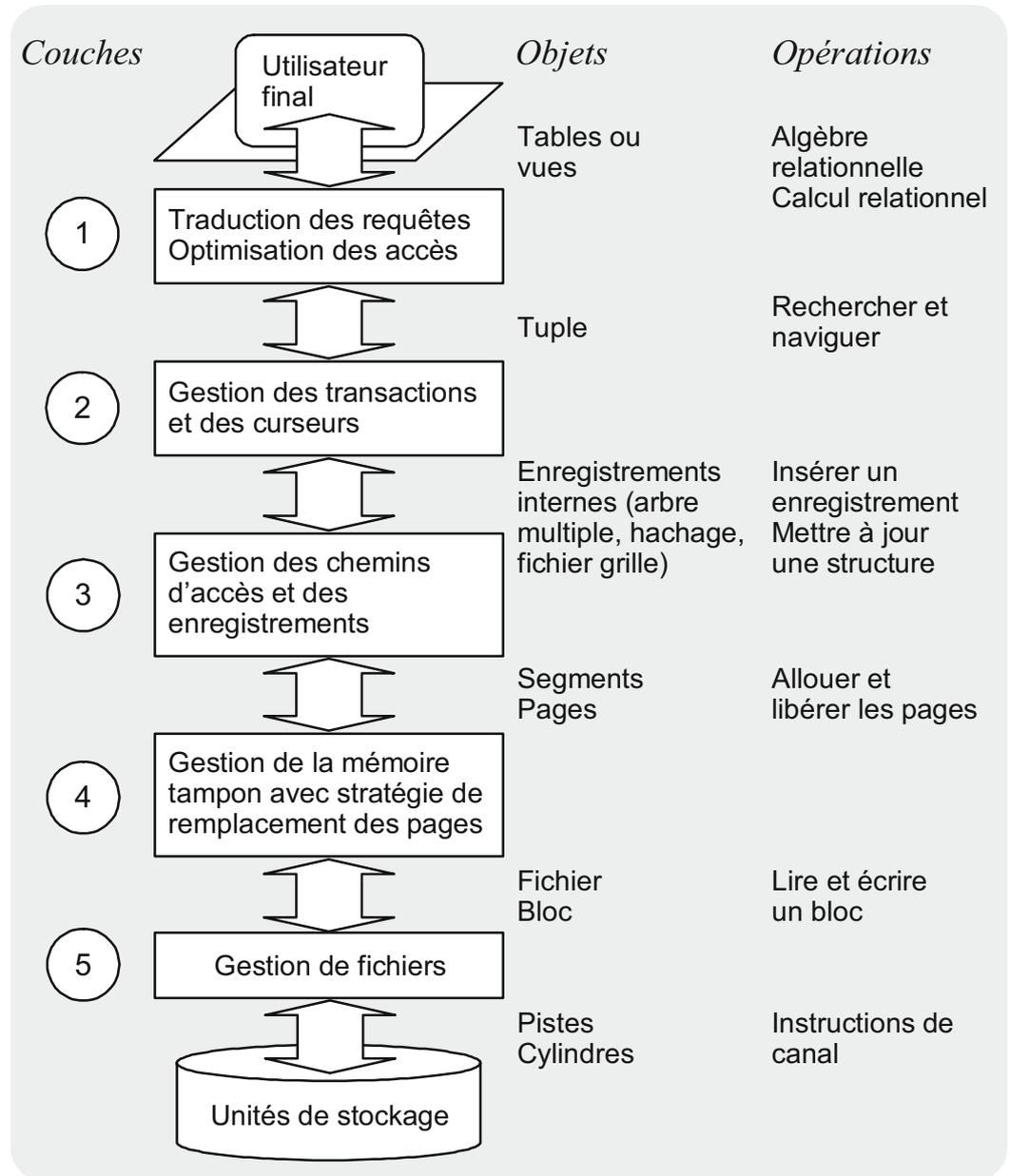
Couche 1 : Interface orientée ensemble

La première couche gère la définition des structures de données, la mise au point des opérations sur les ensembles, la spécification des conditions d'accès et la vérification des contraintes de cohérence (voir 4.2). La correction syntaxique, la résolution des noms et la détermination des chemins d'accès doivent s'effectuer, soit par anticipation lors de la traduction et de la génération des modules

*Couche de
traduction et
d'optimisation des
requêtes*

d'accès, soit plus tard à l'exécution des requêtes. Le choix des chemins d'accès est dicté par le souci majeur d'optimiser les requêtes.

Figure 4-15
Modèle à cinq couches des systèmes de bases de données relationnelles



Couche chargée du maintien de la cohérence

Couche 2 : Interface orientée enregistrement

La deuxième couche a pour fonction de transformer les enregistrements et les chemins d'accès logiques en structures physiques. Le concept de curseur permet de naviguer parmi les enregistrements de données ou de les parcourir dans leur ordre de stockage physique, de localiser un enregistrement particulier dans une table ou d'extraire des tuples de données triés sur des critères précis. Les techniques de gestion des transactions, présentées à la section 4.3,

garantissent la cohérence de la base de données et résolvent les conflits entre les besoins des utilisateurs simultanés.

Couche 3 : Structures de stockage et d'accès

La troisième couche se charge de l'implantation physique des enregistrements et des chemins d'accès sous forme de pages. Les formats de pages possibles sont certes limités. Mais dans le futur, outre les structures arborescentes et les techniques de hachage, les structures de données multidimensionnelles devront aussi être prises en charge. Ces principales structures de stockage visent l'efficacité de l'accès aux unités de mémoire externes. En outre, le groupage physique (création de clusters) et les chemins d'accès multidimensionnels contribuent aussi à l'optimisation dans la gestion des enregistrements et des chemins d'accès.

*Couche assurant
l'efficacité des
accès*

Couche 4 : Allocation de pages

Dans la quatrième couche, par souci d'efficacité et pour mettre en œuvre le mécanisme de reprise sur panne, l'espace d'adresses linéaire est découpé en segments constitués de pages de même taille. Au fur et à mesure des besoins, le gestionnaire de fichiers transfère les pages demandées vers une zone de mémoire tampon. Inversement, une stratégie de remplacement spécifique détermine les pages qu'il faut réécrire dans la mémoire externe avant qu'elles soient remplacées dans la mémoire tampon. Outre l'allocation directe des pages aux blocs, il existe des techniques d'allocation indirecte, telles que celles utilisant la mémoire cache, permettant d'écrire en une seule opération plusieurs pages dans la mémoire tampon de la base de données.

*Couche de gestion
de la mémoire
tampon*

Couche 5 : Allocation de la mémoire

La cinquième couche assure l'allocation de la mémoire externe et communique avec la couche supérieure grâce à une gestion de fichiers orientée bloc. Les spécificités du matériel sont indépendantes des opérations orientées bloc sur les fichiers. En principe, le gestionnaire de fichiers est capable de gérer l'extension dynamique des fichiers avec la possibilité de paramétrer la taille des blocs. En outre, le système doit permettre le groupage des blocs (création de clusters)

*Couche du
système de fichiers*

ainsi que la lecture et l'écriture de plusieurs blocs en une seule opération.

4.7 Notes bibliographiques

Littérature sur l'architecture du système

Certains travaux se consacrent exclusivement à l'architecture des systèmes de bases de données, ou à quelques niveaux particuliers de cette architecture multi-couches. Härder (1978), Härder et Rahm (1999) présentent les concepts fondamentaux de la mise en œuvre des systèmes de bases de données relationnelles (modèle multi-couches). Riordan (2005) et le manuel des bases de données de Lockemann et Schmidt (1993) abordent une sélection de thèmes sur l'architecture des données. Les problèmes d'optimisation sont traités par Ramakrishnan et Gehrke (2003), Ullman (1982) et Maier (1983) sous leur aspect théorique.

Littérature sur la gestion des transactions

Härder et Reuter (1983) proposent le principe ACID. Gray et Reuter (1993), Weikum (1988), Weikum et Vossen (2002) expliquent en détail les concepts relatifs aux transactions. Bernstein et al. (1987) travaillent sur l'environnement multi-utilisateurs et les techniques de restauration. Reuter (1981) présente les techniques de traitement des erreurs dans les systèmes de bases de données. Castano et al. (1995) exposent les différentes méthodes de sauvegarde des bases de données.

Structures de stockage et d'accès

Wiederhold (1983) étudie les structures de stockage dans les systèmes de bases de données. Bayer (1972) traite des arbres multiples, Maurer et Lewis (1975) passent en revue les fonctions de hachage, et le fichier grille est introduit par Nievergelt et al. (1984).

5 Intégration et migration des bases de données

5.1 Exploitation d'ensembles de données hétérogènes

Les ensembles de données subissent des changements structurels et des extensions graduelles au cours du temps. Dans bon nombre d'entreprises, cette évolution entraîne comme conséquence l'existence de données incohérentes et redondantes qui doivent être systématiquement réorganisées. Dans ce contexte, nous parlons de systèmes hérités du passé (*legacy systems*, en anglais) dont le remplacement ou une refonte complète s'impose.

La problématique des systèmes hérités

Une option consiste à entreprendre un développement orienté web de *sous-systèmes d'information*. Les utilisateurs internes et les clients importants de l'entreprise bénéficieront dès lors d'un accès convivial aux données et aux informations disponibles sur le Web. L'intégration des données provenant de sources hétérogènes sur le Web constitue un grand défi : d'une part, la majeure partie des données existantes ont déjà migré progressivement vers des bases de données relationnelles ; d'autre part, la technologie relationnelle n'offre pas toujours des solutions optimales dans la gestion des hyperdocuments.

Les bases de données sur le Web

Dans la section 5.2, des solutions seront proposées pour mettre en œuvre des bases de données sur le Web. Nous y aborderons les aspects architecturaux et présenterons brièvement l'évolution de XML (eXtensible Markup Language) et des bases de données XML. Nous pourrons ainsi évaluer ces technologies en les comparant à celle des bases de données relationnelles.

Avènement de XML

Nécessité d'un environnement de développement homogène

Des entreprises sont confrontées aux problèmes liés à la gestion d'une partie de leurs données qui est encore stockée sous forme de fichiers ou dans des bases de données traditionnelles, hiérarchiques ou en réseau par exemple. L'exploitation des systèmes de bases de données hétérogènes nécessite en effet l'engagement de spécialistes qui maîtrisent les diverses technologies de bases de données installées et de développeurs d'applications aux compétences polyvalentes. Ce constat pousse l'entreprise à *cesser progressivement d'exploiter ses systèmes hérités* pour les migrer vers des bases de données relationnelles ou post-relationnelles.

Nécessité de réduire la complexité des systèmes hétérogènes

La coexistence des ensembles de données hétérogènes représente une hypothèque sur le plan économique. En effet, au coût des ressources humaines s'ajoute un prix à payer pour garantir *la sécurité et la disponibilité d'un ensemble complexe de systèmes*. En outre, la compétitivité de l'entreprise et sa survie même risquent d'être remises en question par l'exploitation parallèle des systèmes d'information hétérogènes, partiellement obsolètes.

Les règles de conversion du système source au système cible sont à définir

Avant de passer en revue les variantes de migration dans ce chapitre, nous traiterons d'abord de *l'intégration des schémas de bases de données* en présentant dans la section 5.3 un ensemble de règles de transformation qui s'appliquent aussi bien à l'intégration qu'à la migration des ensembles de données. Nous y définirons notamment les modalités de conversion des ensembles de données d'un système source (*systèmes hérités ; legacy systems*) en ensembles de données dans un système cible. La migration des systèmes d'information hétérogènes est un projet dont la réalisation demande plusieurs mois, voire plusieurs années. C'est pour cette raison que les règles de transformation doivent être clairement définies de manière à pouvoir continuer encore un certain temps la mise à jour des ensembles de données hérités en cas de besoin.

Avantage de la solution de coexistence

La section 5.4 est consacrée à l'analyse détaillée des variantes de migration. Nous y proposons également une solution de *coexistence des ensembles de données hétérogènes dans le temps*. Cette dernière variante a l'avantage de rendre possible une refonte des systèmes applicatifs en évitant la forte pression du temps et les risques qui y sont liés. En d'autres termes, la variante permet d'évoluer d'après les

besoins du marché et de supprimer graduellement les systèmes hérités pendant le processus de transformation.

L'intégration des ensembles de données sur le Web et la migration des systèmes applicatifs requièrent une bonne planification et une réalisation par étapes. Dans la section 5.5 nous élaborerons les principes de base nécessaires au succès d'un *projet d'intégration et de migration*.

Planification et migration par étapes

5.2 Les bases de données sur le Web

Élément essentiel de la technologie Internet, le Web, appelé aussi la Toile et abrégé en WWW ou W3 (*World-Wide Web*, en anglais), a connu un développement fulgurant au cours des dernières années. Le nombre de systèmes d'information et de bases de données disponibles sur le Web ne cesse de croître, ouvrant ainsi l'accès à une masse considérable d'informations destinées à diverses catégories d'utilisateurs, du grand public à des groupes d'utilisateurs restreints. Avec des serveurs de bases de données et d'applications, un objectif de première importance consiste à réaliser l'intégration des systèmes d'information existants.

L'offre d'informations sur Internet

5.2.1 Création d'un système d'information orienté web

Dans un *système d'information orienté web* (*web-based information system*, en anglais), d'importants documents et informations sont disponibles *en ligne*. Ces systèmes permettent non seulement l'échange d'informations, mais aussi la gestion de la relation clientèle (*customer relationship management*, en anglais) et la conduite des affaires dans le commerce électronique. En outre, l'établissement des offres, la négociation des contrats, la distribution et le trafic des paiements seront progressivement mis en ligne, particulièrement dans la gestion de la chaîne logistique (*supply chain management*, en anglais).

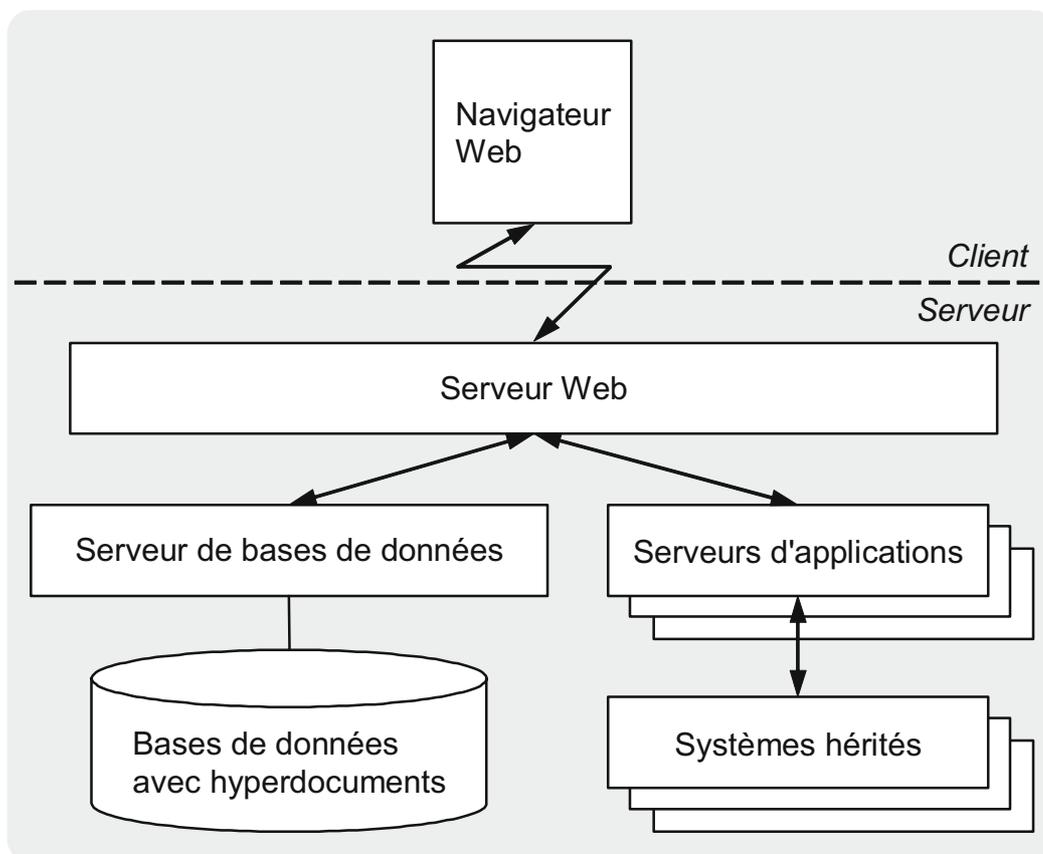
Les systèmes orientés web comme soutien aux processus d'affaires

La figure 5-1 présente l'architecture générale d'un système d'information orienté web. Au cœur du système est un serveur Web

Architecture générale

qui fournit les informations dans des documents hypertextes grâce au protocole de communication HTTP (HyperText Transfer Protocol). La plupart de ces documents sont créés à l'aide du langage HTML (HyperText Markup Language) ou d'un langage plus puissant, XML (eXtensible Markup Language). Nous étudierons plus en détail la gestion des données semi-structurées dans la section 5.2.2.

Figure 5-1
Composants d'un
système
d'information
orienté web



Importance
croissante des
systèmes mobiles

En général, on peut accéder aux informations disponibles sur un serveur Web à n'importe quel moment et depuis n'importe quel endroit. À cette fin, il suffit de disposer d'un ordinateur client (*client*, en anglais) muni d'un navigateur. Le client peut être un ordinateur fixe ou mobile comme les ordinateurs portables, les téléphones mobiles, les Palms, les livres électroniques et les assistants digitaux personnels.

Répartition des
tâches entre
serveur de bases
de données et
serveurs
d'applications

Les documents hypertextes peuvent être stockés de manière statique dans le système de fichiers d'un serveur Web, ou générés de manière dynamique lors de l'accès d'un utilisateur au serveur. Il existe plusieurs méthodes et techniques de génération dynamique des documents, dont la discussion dépasse le cadre de cet ouvrage. En principe, les informations nécessaires à la création de documents sont

stockées sur un serveur de bases de données. En outre, il est possible d'exploiter les données fournies par des systèmes d'information hérités (*legacy systems*, en anglais) dotés d'interfaces appropriées. Les serveurs d'applications ont pour fonction de traiter les travaux entrants et accèdent également au serveur de bases de données ou aux systèmes d'information existants.

L'administrateur d'un système d'information orienté web et du site Web associé se trouve confronté à un *gros volume de documents hypertextes*. C'est pourquoi, il a besoin de serveurs de bases de données pour assurer leur stockage à long terme. Le contenu des pages HTML dynamiques peut être créé à partir des bases de données relationnelles. Des documents entiers peuvent également être mémorisés dans des systèmes de bases de données relationnelles orientées objet (voir la section 6.4). Une autre solution de stockage consiste à gérer les données semi-structurées dans des bases de données XML. Nous étudierons cette approche de plus près dans la section suivante.

Mise en œuvre des bases de données relationnelles orientées objet et des bases de données XML

5.2.2 Documents et schémas XML

Le langage de balisage XML (eXtensible Markup Language) était conçu par le Consortium World Wide Web (W3C). Comme en HTML, le contenu d'un document hypertexte est marqué par des balises. Un document XML est auto-descriptif, car il contient à la fois des données proprement dites et des informations sur leur structure :

Le langage de balisage XML

```
<Adresse>
  <Rue>Rue Faucigny</Rue>
  <Numero>2</Numero>
  <Numero_postal>1700</Numero_postal>
  <Ville>Fribourg</Ville>
</Adresse>
```

Fragment d'un document XML

La brique de base d'un document XML se nomme l'élément. Le contenu d'un élément est délimité par une balise d'ouverture `<Nom>` (entourée d'une paire de crochets pointus) et une balise de fermeture `</Nom>` (entourée d'une paire de crochets pointus avec un trait

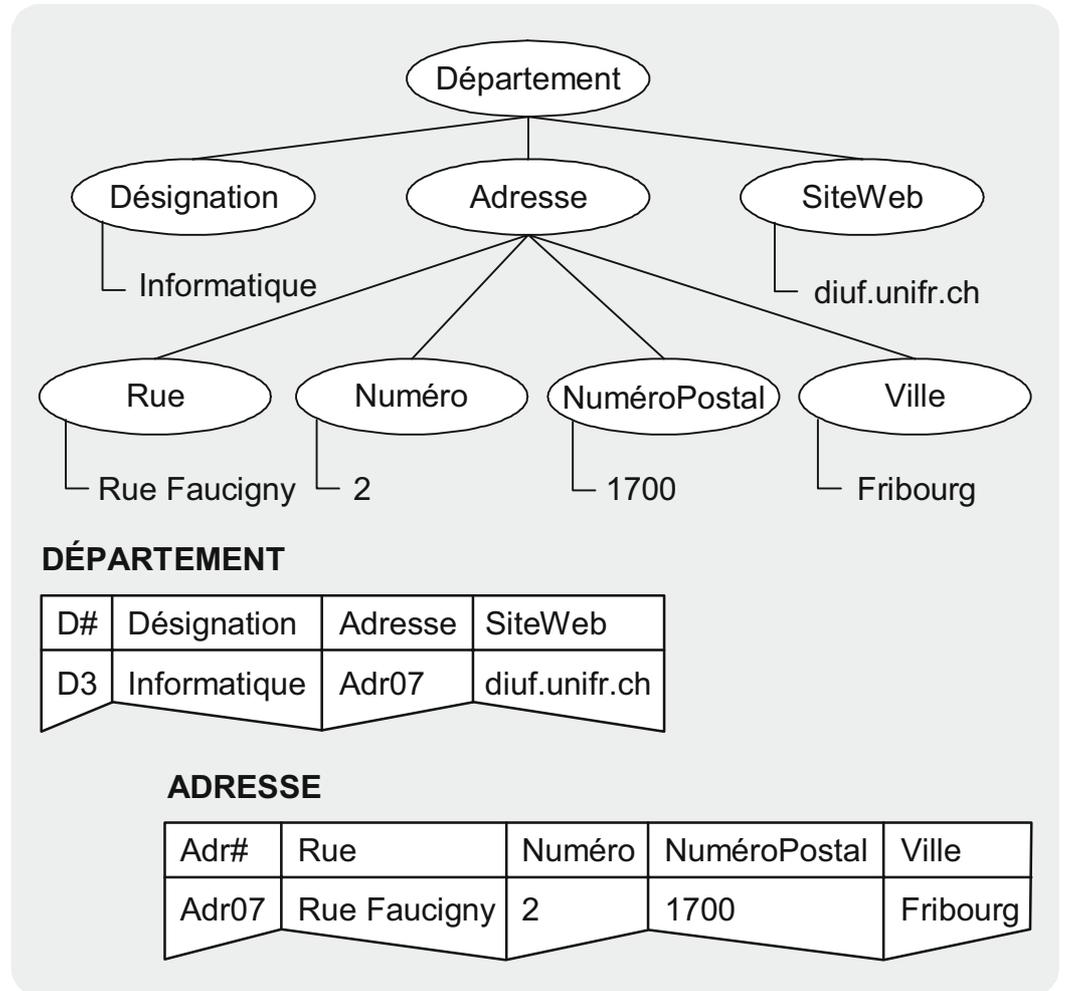
Les éléments sont marqués par des balises

oblique). Les identificateurs des balises d'ouverture et de fermeture doivent être identiques.

Les documents XML peuvent avoir une structure quelconque

Les balises expriment la sémantique des données, permettant ainsi de comprendre la signification de leurs valeurs. Les éléments dans un document XML peuvent être imbriqués aussi profondément que nécessaire. La *structure hiérarchique du document* qui en résulte peut être visualisée par un diagramme, comme le montre la figure 5-2.

Figure 5-2
Représentation d'un document XML dans un schéma de base de données relationnelle



Définition de la structure en XML

Comme nous l'avons dit plus haut, un document XML contient implicitement des *informations sur sa structure*. La connaissance de la structure d'un document XML est importante aux applications. C'est pourquoi le W3C a élaboré des recommandations pour la représenter explicitement (DTD = Document Type Definition ou Schéma XML). Un schéma explicite indique les balises présentes dans un document XML et la façon dont elles sont organisées. Il facilite ainsi la détection et la correction des erreurs dans le document. La suite de l'exposé

illustrera le schéma XML et ses avantages dans la construction des systèmes de bases de données.

Dans la figure 5-2 nous mettons en parallèle un document XML avec un schéma de base de données relationnelle. Les schémas de bases de données relationnelles se caractérisent généralement par trois éléments hiérarchiquement imbriqués : la désignation de la base de données, les noms de relations et les noms d'attributs. Le fragment d'un document XML représenté en figure 5-2 nous montre les deux relations DÉPARTEMENT et ADRESSE, leurs attributs et les valeurs associées. Nous parlerons brièvement de l'introduction des clés primaires et étrangères qui est possible dans un schéma XML.

Le typage des données et la déclaration des noms attribués aux types de données constituent le concept fondamental du schéma XML grâce auquel nous pouvons représenter explicitement tout document XML. Il est en outre possible de définir des règles d'intégrité pour contrôler la validité des documents XML.

Il existe une multitude de types de données prédéfinis tels que String, Boolean, Integer, Date, Time, etc., auxquels s'ajoutent des types de données définis par l'utilisateur. Les facettes permettent de déclarer les propriétés spécifiques à un type de donnée. Nous pouvons ainsi définir la propriété d'ordre d'un type de donnée, les limites inférieure et supérieure d'un intervalle de valeurs, les restrictions de longueur, ou énumérer les valeurs admissibles :

```
<xs:simpleType name=«Ville»>
  <xs:restriction base=«xs:string»>
    <xs:length value=«20»/>
  </xs:restriction>
</xs:simpleType>
```

Dans l'exemple ci-dessus, nous définissons un type de donnée simple, «Ville», basé sur le type prédéfini String. De plus, nous imposons la contrainte que le nom d'une ville ne doit pas dépasser 20 caractères.

Mise en correspondance d'un document XML et une base de données relationnelle

Le schéma XML définit les types de données

Déclaration des propriétés spécifiques

Un exemple de type de donnée simple dans un schéma XML

Les clés et les liens de référence

Dans un schéma XML, les clés des tables d'une base de données sont définies à l'aide de «key». La déclaration «keyref» permet de spécifier la référence à une clé préalablement définie. Ainsi, ces deux constructions permettent de définir les clés primaires et les clés étrangères.

Représentation de documents et schémas XML

Des éditeurs XML ont été développés pour représenter graphiquement les documents et schémas XML. Ces outils aident à la déclaration des propriétés structurales d'un document ainsi qu'à la saisie des données. La possibilité d'agrandir et de réduire les sous-structures facilite le travail avec des documents et schémas XML volumineux.

5.2.3 Le langage de requête XQuery

XQuery permet la sélection des valeurs et des structures

Il est important que nous puissions interroger et analyser les documents et les bases de données XML. À la différence des langages de requête relationnels, les critères de sélection sont définis non seulement par rapport aux valeurs (sélection de valeurs), mais aussi aux structures des éléments (sélection de structures). L'extraction et la mise à jour des sous-éléments dans un document XML figurent parmi les opérations de base d'une requête XML. Il est possible de combiner des éléments provenant de différentes structures à la source pour créer de nouvelles structures d'éléments. Enfin, dernier point mais non le moindre, un langage de requête approprié doit permettre de naviguer à l'aide d'hyperliens et de références, d'où l'importance des expressions de chemin.

FLWR est la structure de base de XQuery

XQuery, proposé par le W3C, emprunte des éléments au langage SQL, à divers langages XML (par exemple XPath, langage de navigation dans les documents XML), ainsi qu'aux langages de requête orientés objet. Les expressions FOR-LET-WHERE-RETURN constituent la base de XQuery : FOR et LET lient une ou plusieurs variables aux résultats de l'évaluation des expressions. Comme en SQL, la clause WHERE permet de définir des restrictions sur l'ensemble résultat. Le résultat d'une requête est généré par RETURN.

L'exemple simple suivant illustre les bases de XQuery de manière sommaire. Il s'agit d'interroger un document XML

«Département» (voir figure 5-2) pour en extraire les noms des rues de l'ensemble des départements :

```
<NomRue>
  {FOR $Département IN //Département
    RETURN $Département/Adresse/Rue }1
</NomRue>
```

*Exemple d'une
expression FLWR
simple*

Au cours du traitement de la requête ci-dessus, la variable \$Département est liée successivement à chacun des nœuds de type <Département>. Pour chaque lien, l'adresse sera évaluée et le nom de la rue fourni par l'expression RETURN. La requête formulée en XQuery produit le résultat suivant :

*Création de liens
pour les variables*

```
<NomRue>
  <Rue>Rue Faucigny</Rue>
  <Rue>..... </Rue>
  <Rue>..... </Rue>
</NomRue>
```

*Le résultat d'une
requête XQuery
forme une
séquence*

Dans XQuery chaque variable est identifiée de manière unique par un nom précédé du symbole du dollar (\$) pour la différencier des noms d'éléments. Contrairement à certains langages de programmation, une variable dans XQuery peut n'avoir aucune valeur. En fait, les expressions sont d'abord évaluées, puis le résultat est assigné aux variables. Cette liaison des variables au résultat se réalise par les expressions FOR et LET de XQuery.

*Le lien des
variables au
résultat est créé
par FOR et LET*

Dans la requête ci-dessus, nous n'avons pas utilisé l'expression LET. La clause WHERE permet de restreindre l'ensemble résultat. La clause RETURN est évaluée à chaque itération de la boucle FOR, mais ne renvoie pas de résultat. En revanche, les résultats successifs forment une séquence et constituent l'output de l'expression FOR-LET-WHERE-RETURN.

*RETURN renvoie le
résultat d'une
requête*

¹ Les accolades viennent du langage de navigation XPath. Les expressions mises entre accolades sont évaluées directement par le parseur.

XQuery, langage de requête pour documents XML et bases de données XML

XQuery est un langage de requête puissant pour hyperdocuments, et il sera disponible aussi bien pour les bases de données XML que pour les systèmes de bases de données post-relationnelles. Des extensions dans les composants de stockage doivent être mises en œuvre afin de rendre possible l'enregistrement des documents XML dans les systèmes de bases de données relationnelles. Ainsi par exemple, on peut travailler avec XML dans les systèmes de bases de données relationnelles orientées objet grâce à l'introduction du concept de structure hiérarchique (voir 6.4).

5.3 Règles de conversion pour l'intégration et la migration des données

Des règles de mise en correspondance sont nécessaires

La migration des bases de données et l'intégration des ensembles de données hétérogènes sur le Web exigent des règles de mise en correspondance (*mapping rules*, en anglais) uniques entre les schémas de bases de données du système source et ceux du système cible ; il est concevable que plusieurs systèmes sources soient convertis en un seul système cible. Pour éviter d'appliquer des règles de conversion aux différents modèles de données de manière dispersée, nous devons partir d'une abstraction du système source sous la forme d'un modèle entité-association et admettre que le système cible repose sur un modèle de données relationnel ; des règles de mise en correspondance analogues peuvent aussi être définies pour une base de données XML.

Lien unique entre le système source et le système cible

L'intégration et la migration des données reposent sur un principe fondamental : les ensembles d'entités du système source sont transformés de manière unique en tables dans le système cible, et vice versa. Dans un projet d'intégration ou de migration, l'unicité de cette conversion est vitale en raison du maintien des applications qui doivent continuer à fonctionner dans le système source. Même lorsque la conversion des programmes d'application existants en programmes du système cible est assistée par l'ordinateur, l'unicité est la condition primordiale pour réussir une migration des données méthodique et sans erreur.

5.3.1 Conversion des ensembles d'entités simples et des groupes répétitifs

Pour traduire un modèle entité-association en un schéma de base de données relationnelle et préparer l'intégration ou la migration subséquente des données, nous distinguons plusieurs types de conversion des ensembles d'entités et d'associations.

Le premier type de conversion, schématisé à la figure 5-3, s'applique exclusivement aux ensembles d'entités. La *conversion de type 1a* fait correspondre à chaque ensemble d'entités A une table distincte A dont la clé (possiblement artificielle) est formée par une combinaison unique et minimale d'attributs. Quel que soit le type de conversion, nous pouvons inclure *la totalité ou seulement une partie des attributs* des ensembles d'entités du système source dans les tables du système cible. Ainsi, pour le type de conversion 1a, il n'est pas obligatoire de traduire toutes les propriétés de l'ensemble d'entités A en attributs de la table A.

La conversion de type 1a transforme les ensembles d'entités en tables

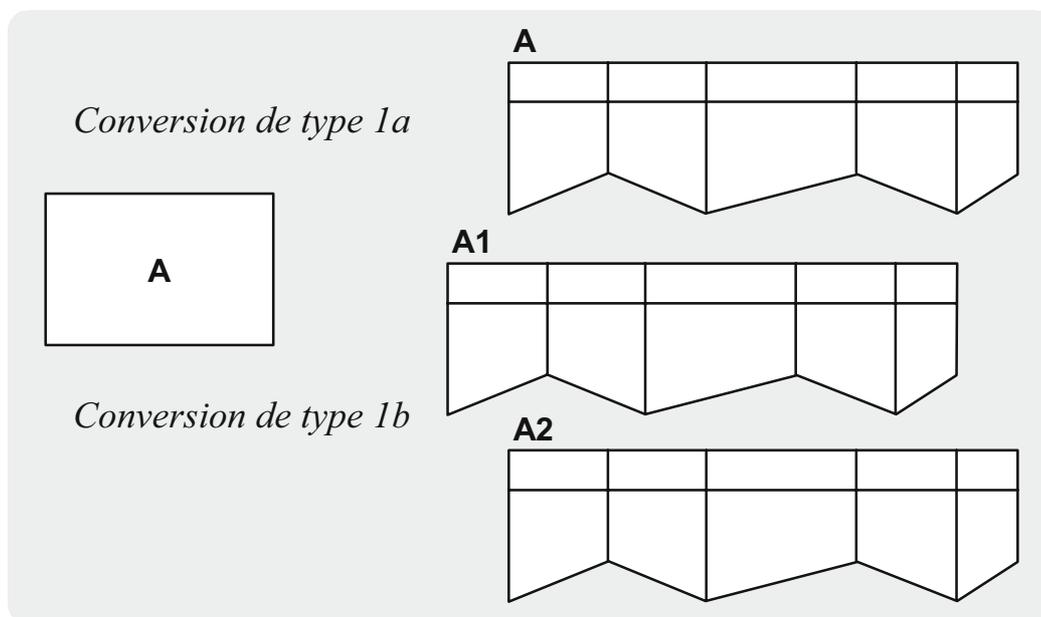


Figure 5-3
Règles de conversion des ensembles d'entités simples

Une extension du type de conversion 1a consiste à décomposer un ensemble d'entités en deux ou plusieurs tables par un prédicat de sélection. Ainsi, dans la *conversion de type 1b*, l'ensemble d'entités A devient deux tables A1 et A2. Cette règle de conversion se révèle utile en pratique lorsque nous avons besoin de créer des classes d'après un critère de sélection sur l'ensemble d'entités A en vue *d'obtenir deux*

Conversion de type 1b et classification des ensembles d'entités

nouveaux ensembles d'entités A1 et A2 lors d'une intégration ou d'une migration (rénovation des applications héritées).

Un exemple de classification

Dans la figure 5-3, une règle de décomposition précise permet donc de convertir les entités dans un ensemble d'entités A en deux tables A1 et A2. Admettons par exemple que A désigne l'ensemble d'entités EMPLOYÉ. Nous pouvons, par décomposition, créer les tables A1 et A2 comme deux sous-classes : la première contenant les employés domiciliés à Bulle et la seconde les employés habitant dans les autres villes. Dans ce cas, les deux tables A1 et A2 divisent la totalité des employés enregistrés en deux sous-classes exactement. Inversement, à partir des tables A1 et A2, la reconstruction de l'ancienne base de données des employés A est garantie.

Naturellement, dans la conversion de type 1b, il n'est pas impératif de traduire toutes les entités en autant d'entrées dans les sous-tables correspondantes. Par exemple, si la table A2 est prévue seulement pour les employés domiciliés à Fribourg, alors les deux tables A1 et A2 ne renferment que des employés habitant Bulle et Fribourg. La conversion ne porte pas sur les employés qui habitent dans les autres villes et qui sont donc maintenus dans l'ancienne base de données. Malgré cette création de classes incomplète, la reconstruction de l'ancienne base de données à partir de ces sous-ensembles est possible et unique.

La conversion de type 2a supprime les groupes répétitifs

Les groupes répétitifs, c'est-à-dire les ensembles d'entités qui incluent des sous-ensembles d'entités, deviennent en principe des tables distinctes, et inversement. Dans la figure 5-4, l'ensemble d'entités A contient deux groupes répétitifs A1 et A2. La *conversion de type 2a* le transforme en trois tables A, A1 et A2. L'éclatement des groupes répétitifs A1 et A2 en deux tables distinctes A1 et A2 s'impose en vertu de la première forme normale qui n'admet que des attributs à valeurs atomiques (voir la section 2.4.2).

La conversion de type 2b traduit les groupes répétitifs en tables imbriquées

Une extension du modèle relationnel consiste ici à admettre des groupes répétitifs comme valeurs d'attributs, ce qui donne lieu à des tables imbriquées. Ces tables non normalisées, c'est-à-dire qui ne sont pas en première forme normale, seront approfondies dans la section 6.4. Néanmoins, nous les utilisons déjà ici à des fins de conversion

dans le cadre de l'intégration et de la migration. La *conversion de type 2b* (figure 5-4) transforme l'ensemble d'entités A avec ses groupes répétitifs A1 et A2 en une table A qui inclut les sous-tables A1 et A2. Les tables ainsi imbriquées ne sont pas prises en charge par tous les systèmes de bases de données, mais leur présence est de plus en plus répandue.

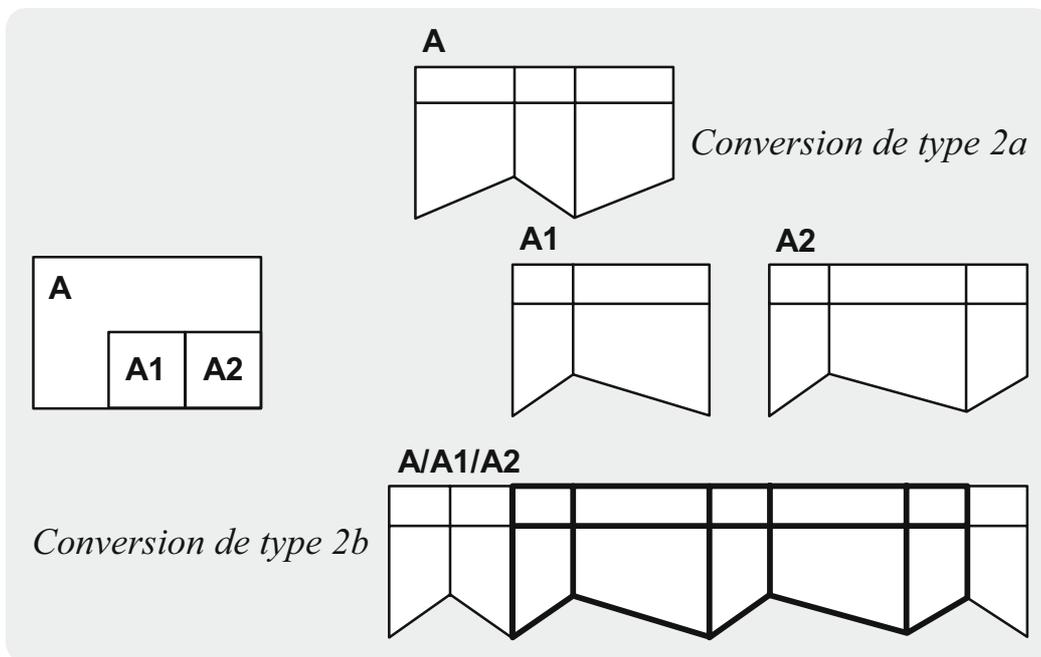


Figure 5-4
Règles de
conversion des
groupes répétitifs

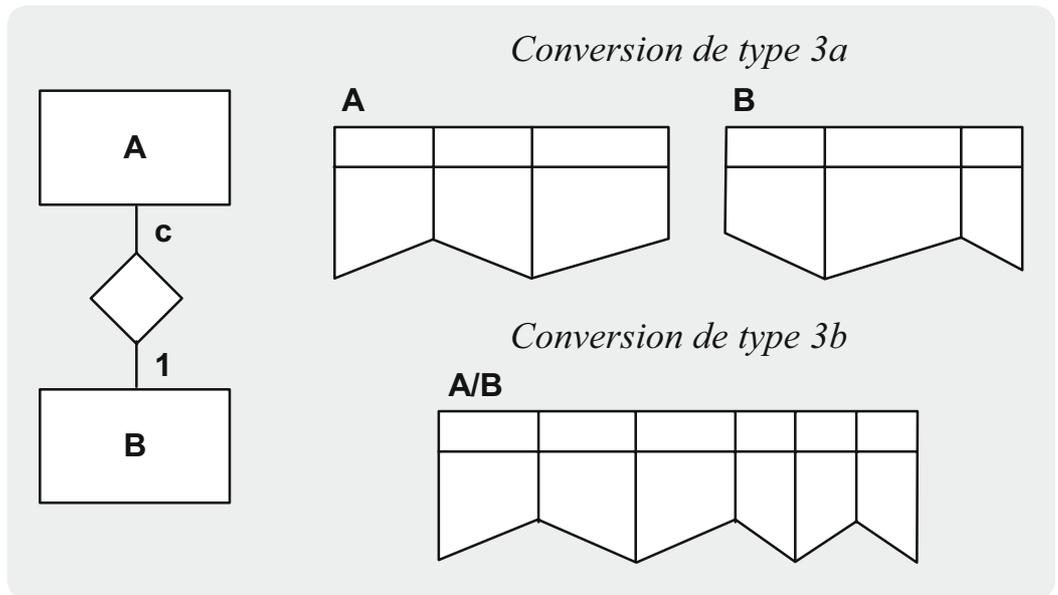
5.3.2 Conversion des ensembles d'entités dépendants

Nous abordons à présent les règles pour traduire des ensembles d'entités dépendants en tables dépendantes, et inversement. La première règle de conversion s'applique au cas où deux ensembles d'entités A et B ont une liaison de degré (c,1). En d'autres termes, à chaque entité dans l'ensemble d'entités A correspond «au plus une» entité dans l'ensemble d'entités B ($c=0$ ou $c=1$) ; inversement, à chaque entité de B correspond «exactement une» (1) entité de A.

Nous distinguons ici deux types de conversion. La *conversion de type 3a* (Figure 5-5) est le cas normal où une liaison de degré (c,1) entre deux ensembles d'entités A et B se traduit par la création de deux tables A et B. L'attribut A# est défini comme clé étrangère dans la table B et constitue une référence à la table dépendante A.

La conversion de
type 3a génère une
table pour chaque
liaison (c,1)

Figure 5-5
Règles de
conversion des
ensembles
d'entités simples
dépendants



La conversion de
type 3b traduit
chaque liaison (c,1)
en une seule table

La figure 5-5 contient une variante importante, la *conversion de type 3b* qui réunit les deux ensembles d'entités en une seule table A/B. Comme nous venons de le rappeler, le type d'association c fait correspondre à chaque entité dans A «au plus» une entité dans B. Par conséquent, puisque chaque entité dans A n'est pas nécessairement connectée à une entité dépendante dans B, la table A/B peut contenir des valeurs nulles. C'est pourquoi la conversion de type 3b n'a de sens que si la plupart des entités dans A admettent une entité dans B.

Rénovation des
systèmes hérités

La conversion de type 3b permet en outre de corriger les schémas de bases de données existants. Par souci d'efficacité, l'extension des bases de données traditionnelles se réalise souvent au mépris des principes de la modélisation de données : on a tendance à compléter un ensemble d'entités particulier par des ensembles d'entités additionnels ou de nouveaux segments de telle manière qu'il ne soit pas nécessaire d'adapter les applications existantes à cette extension du schéma de base de données. Au cours de l'intégration ou de la migration des données, la conversion de type 3b permet de fusionner deux, voire plusieurs ensembles d'entités conformément à la théorie des formes normales (rénovation des systèmes hérités).

Comment traduire
les liens
hiérarchiques ?

La figure 5-6 illustre la *conversion de type 4*. Deux ensembles d'entités connectés par une véritable liaison hiérarchique (m,1) peuvent se traduire en deux tables ou en une seule. À chaque entité dans l'ensemble d'entités C sont associées «plusieurs» (m) entités dans

l'ensemble d'entités D ; inversement, à chaque entité dans D correspond «exactement une» (1) entité dans C.

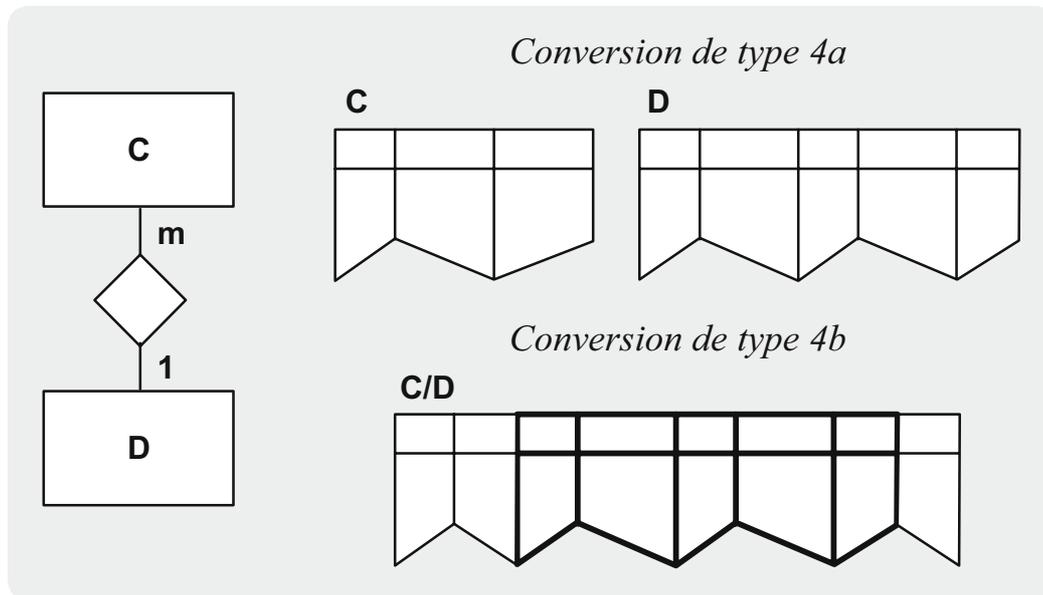


Figure 5-6
Règles de conversion des ensembles d'entités avec liaison de type simple-complexe

La *conversion de type 4a* transforme l'ensemble d'entités C en une table C et l'ensemble d'entités D en une table D. La liaison hiérarchique des deux tables s'exprime par la clé étrangère C# dans la table D. Si le système de bases de données relationnelles prend en charge des tables imbriquées, nous pouvons appliquer la *conversion de type 4b*. Les tables imbriquées ou récursives peuvent comporter des attributs qui sont à leur tour des tables. La table C/D n'est plus du tout en première forme normale, mais à tout moment nous pouvons la décomposer par des opérateurs de projection, en deux tables C et D comme dans la conversion de type 4a (et inversement).

La conversion de type 4 traduit une hiérarchie en une table ou deux

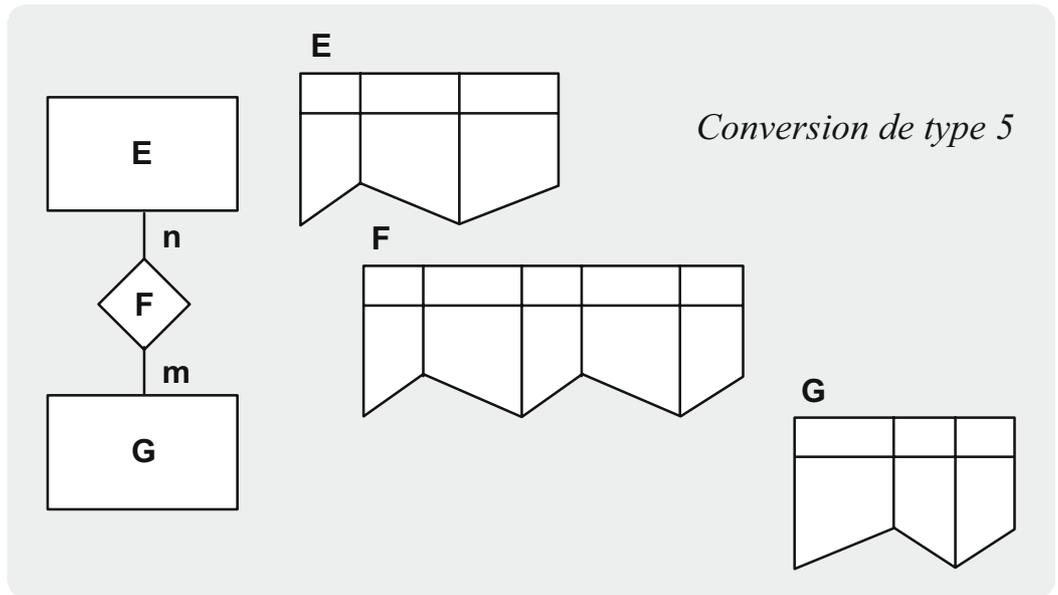
En vertu des propriétés connues des formes normales, les ensembles de liaisons de type plusieurs-à-plusieurs doivent être convertis en tables séparées qui ne contiendront pas de redondances. Ainsi, la *conversion de type 5* (Figure 5-7) traduit les ensembles d'entités E et G en deux tables E et G, et l'ensemble de liaisons F en une table F, dont on complète la définition par des contraintes d'intégrité référentielle appropriées.

La conversion de type 5 transforme une liaison complexe-complexe en trois tables

Outre les règles de conversion de type 1 à 5, nous pouvons concevoir d'autres règles qui produisent des tables reflétant par exemple une hiérarchie de généralisation ou une structure

d'agrégation. De nos jours, les systèmes de bases de données relationnelles qui supportent ces concepts d'abstraction sont encore relativement rares dans la pratique. C'est pourquoi nous n'aborderons pas ces thèmes en profondeur dans le présent ouvrage.

Figure 5-7
Règles de conversion des ensembles d'entités avec liaison de type complexe-complexe



Possibilité de combiner les différents types de conversion

Au niveau logique, les conversions de type 1 à 5 permettent de traduire en tables les structures de données en réseau ou hiérarchiques. Il va de soi qu'il est possible de combiner librement ces types de transformation. Ils sont en partie supportés par des outils de rétroingénierie disponibles sur le marché dans le but de dériver des schémas relationnels à partir des structures de données traditionnelles. Naturellement, à ce stade de la transition d'une base de données non relationnelle à un système relationnel, la migration ne touche pas encore à sa fin, car nous devons aussi développer de nouveaux programmes d'application ou adapter les programmes existants.

5.3.3 Les conversions indirectes pour l'intégration et la migration des données

À première vue les transformations de type 1 à 5 paraissent trop restrictives. Pour réaliser l'intégration ou la migration des données, pourquoi ne pouvons-nous pas transformer un ensemble d'entités particulier en un nombre quelconque de tables tout simplement ? Ou convertir en une seule table plusieurs ensembles d'entités dans une base de données non relationnelle, et vice versa ?

La réponse se trouve dans les principes de la modélisation des données que nous avons approfondis à la section 2.3. Les règles de modélisation 1 à 7 énoncées dans cette section permettent de traduire méthodiquement un modèle d'entité-association en un schéma de base de données relationnelle. Des règles similaires s'appliquent à l'intégration ou à la migration des ensembles de données. Par conséquent, dans la conversion des bases de données, il n'est pas toujours correct de décomposer chaque ensemble d'entités en un nombre quelconque de tables, ou de combiner en une seule table plusieurs ensembles d'entités arbitrairement choisis. Ce n'est qu'au moment d'exploiter une base de données relationnelle que nous sommes libres de sélectionner ou de combiner des informations à notre manière.

L'intégration doit respecter les principes de la modélisation des données

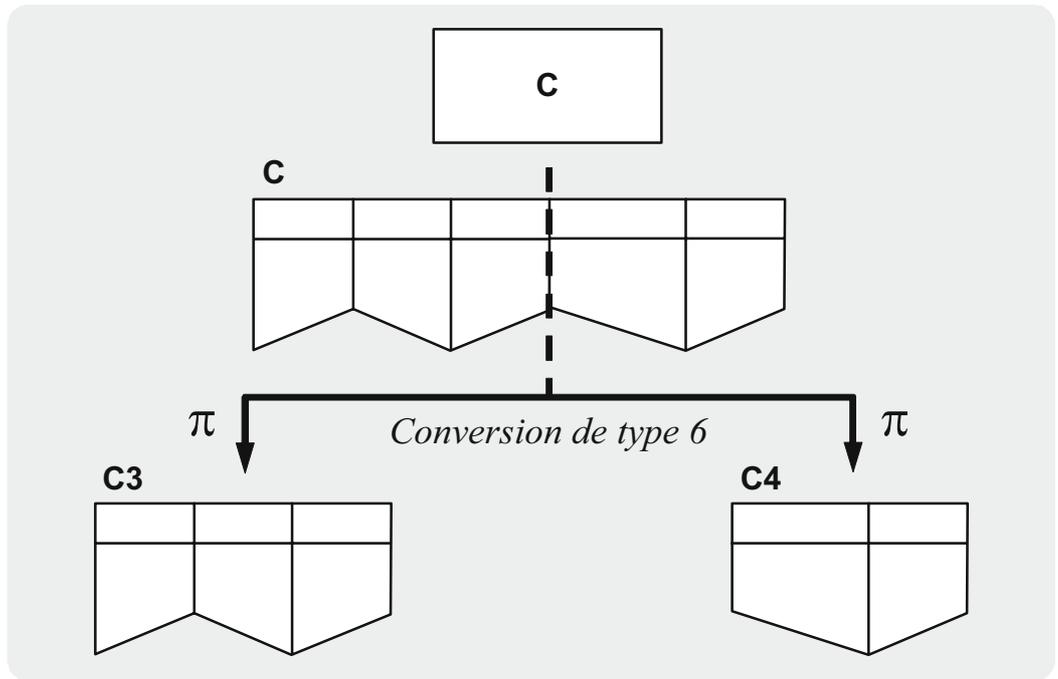
Si les systèmes applicatifs hérités du passé étaient correctement conçus et mis en œuvre avec leurs bases de données hétérogènes, les règles de conversion de type 1 à 5 permettraient leur transition au relationnel sans difficulté. Cependant, la réalité nous montre que les bases de données étaient souvent bâties de manière inadéquate et limitées à un domaine d'application particulier. Ainsi, même de nos jours, on constate qu'une architecture de données globale de l'entreprise est souvent absente ou, tout au plus, à l'état de projet en construction. Comment pouvons-nous contourner les lacunes d'un système si les règles de transformation 1 à 5 ne suffisent pas ?

Comment sortir du chaos des données ?

La *conversion de type 6* permet de décomposer indirectement les entrées d'un ensemble d'entités en plusieurs tables. En appliquant ce type de transformation illustrée par la figure 5-8, une partie des attributs de l'ensemble d'entités C est indirectement convertie en une table C3 et le reste en une table C4. Un problème se pose lorsqu'il faut reconstituer l'ensemble d'entités C : si à chaque entrée dans C3 ne correspond plus exactement une entrée dans C4, la reconstruction univoque de l'ensemble d'entités C à partir des tables C3 et C4 n'est plus garantie. Cela arrive, par exemple, après la suppression d'un certain nombre de tuples dans C4.

La conversion de type 6 décompose les ensembles d'entités

Figure 5-8
Règle de
conversion
indirecte par la
projection



La mise à jour des
vues est soumise à
des restrictions

La transformation de type 6 est *indirecte*, car elle s'effectue sans conflit en passant par un niveau intermédiaire. Tout d'abord, une conversion de type 1a (éventuellement de type 1b) transfère les entités de C vers la table correspondante C. Ensuite, c'est l'administrateur de la base de données relationnelle qui décide de décomposer la table C en deux sous-tables C3 et C4 par des opérateurs de projection par exemple. Il met ainsi à disposition de l'utilisateur deux vues C3 et C4. La base de données relationnelle demeure cohérente, car les opérations de mise à jour des vues C3 et C4 sont soumises à certaines restrictions. Par exemple, un système de bases de données relationnelles refuse d'exécuter une suppression dans la vue C4 si cette opération n'entraîne pas la suppression univoque d'un tuple correspondant dans la table de départ C.

La conversion de
type 7 fusionne
deux ensembles
d'entités
indépendants

La *conversion de type 7* (Figure 5-9) est également *indirecte*, car les deux ensembles d'entités B et T ne fusionnent pas directement pour former une table B/T. En fait, une conversion de type 1a traduit d'abord les ensembles d'entités B et T en deux tables distinctes B et T. Celles-ci sont ensuite combinées par un opérateur de jointure $|\times|$. Normalement, la jointure de deux tables n'a de sens que si elles sont liées par leurs attributs communs. La table désirée B/T peut donc être considérée comme une vue résultant de la jointure des tables B et T. Le processus inverse de la conversion de type 7 permet à tout moment

de reconstruire les ensembles d'entités B et T à partir des tables B et T.

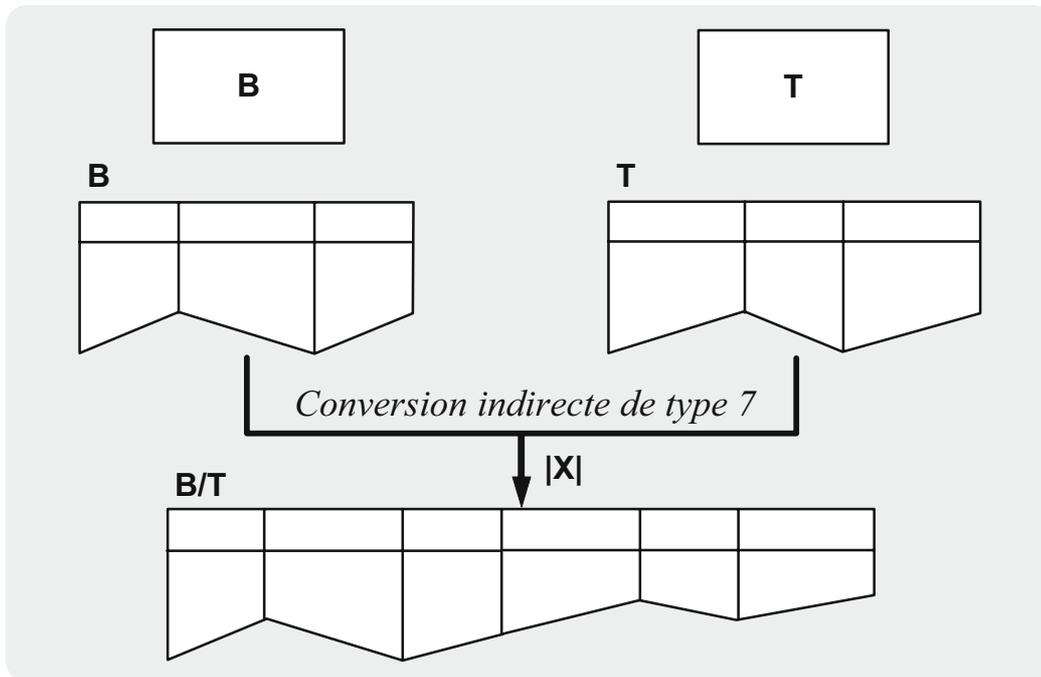


Figure 5-9
Règle de
conversion
indirecte par la
jointure

Dans l'étude des formes normales à la section 2.4, nous avons souligné qu'il n'est pas toujours possible de décomposer une table en sous-tables par des opérateurs de projection et de la reconstruire ensuite par des opérateurs de jointure (voir Dépendance de jointure, section 2.4.4). C'est donc en vertu de la cinquième forme normale que les règles de conversion indirecte ont été définies afin de permettre la décomposition et la reconstruction des ensembles d'entités. Elles viennent compléter ainsi les règles de conversion directe.

*Prudence avec la
dépendance de
jointure*

5.4 Variantes de migration des bases de données hétérogènes

Quoique, de nos jours, de nombreuses entreprises exploitent encore en parallèle des bases de données relationnelles et non relationnelles, une telle coexistence pose de sérieuses difficultés : *l'actualité et la cohérence des données deviennent problématiques dans un environnement de bases de données hétérogènes*. La situation se complique du fait que de gros investissements ont été consentis pour développer une multitude de programmes d'application dans les systèmes de bases de données non relationnelles. À long terme,

*Comment
préserver les
investissements ?*

comment les entreprises concernées parviendront-elles à gérer ce patrimoine d'applications héritées du passé ?

5.4.1 Caractérisation des variantes de migration

Dans le milieu académique et plusieurs centres de recherche, la compatibilité des générations successives de systèmes de bases de données lors d'une migration constitue rarement un thème de discussion. En revanche, plusieurs sociétés de logiciels et certaines entreprises dotées de systèmes d'information de grande envergure ont pris l'initiative de mettre en place le support logiciel nécessaire au «rajeunissement» d'une génération de bases de données ou à la coexistence des systèmes de bases de données hétérogènes.

Nous caractérisons ci-après de manière sommaire les plans d'action possibles, schématisés en figure 5-10.

Variante de migration : convertir les données et les programmes

- *Conversion des données et des programmes d'application (Figure 5-10, Variante 1) : les bases de données non relationnelles sont d'abord transformées en bases de données relationnelles d'après les règles générales de conversion (voir 5.3.1 et 5.3.2). Ensuite, les données sont transférées vers les nouvelles bases à l'aide d'utilitaires d'exportation et d'importation. Dans une étape séparée, on recourt dans une large mesure aux routines de conversion pour traduire automatiquement des requêtes procédurales dans un programme applicatif en requêtes dans le langage relationnel du système cible. Dans des cas particuliers ou pour des raisons de performance, on peut être amené à reformuler manuellement les requêtes après leur conversion automatique. Dans certaines situations, la migration des données et la conversion des programmes impliquent une refonte complète de l'ensemble des applications. Cela signifie, dans le meilleur des cas, l'arrêt de l'exploitation d'un système de bases de données non relationnelles.*

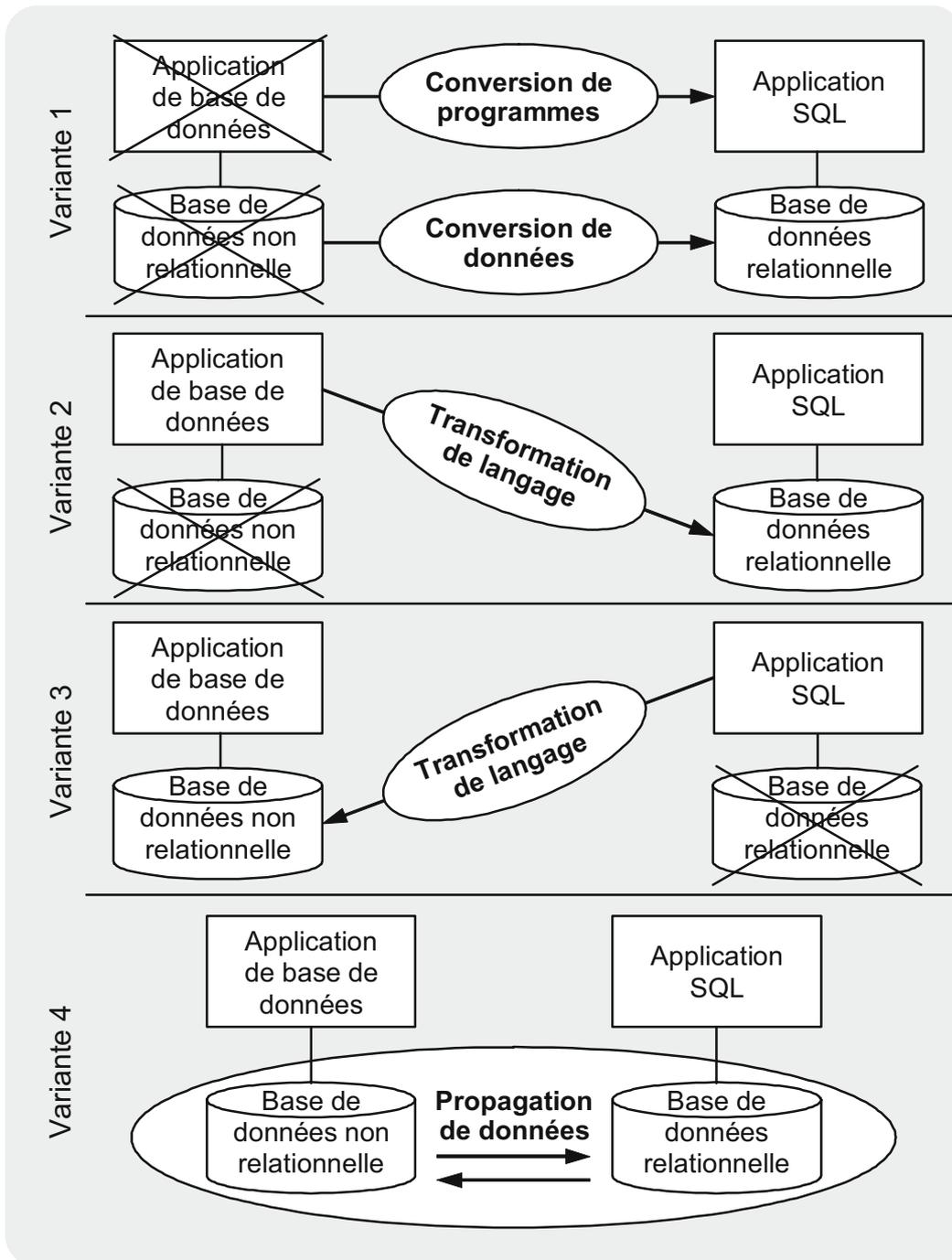


Figure 5-10
Vue d'ensemble
des principales
variantes de
migration

- *Passage du langage cible procédural à une interface descriptive (Variante 2) :* à cette fin, on doit ajouter au système de bases de données relationnelles une couche logicielle générale (*wrapper*, en anglais) qui traduit chaque requête procédurale dans un langage relationnel d'interrogation et de manipulation de données. Les programmes d'application existants ne sont donc pas affectés, car la conversion au niveau de la couche logicielle additionnelle porte uniquement sur des requêtes adressées au

Variante de migration :
transformer les requêtes de bases de données contenues dans les programmes existants

système cible relationnel. Des études empiriques sur les systèmes de bases de données relationnelles et non relationnelles révèlent des difficultés majeures pour traduire chaque requête procédurale avec toutes ses règles de traitement imaginables en une requête ensembliste équivalente, sans parler des problèmes de performance.

Variante de migration : Exécuter les applications en SQL sur des bases de données non relationnelles

- *Passage inverse du langage cible relationnel à une interface procédurale (Variante 3) : pour ce faire, on dote le système de bases de données non relationnelles d'une couche logicielle générale qui permet de formuler des requêtes dans un langage descriptif et de les adresser au système non relationnel. Les expressions ensemblistes seront prises en charge par l'interface procédurale du système de bases de données non relationnelles, la plupart du temps au prix d'une dégradation sensible de la performance. Un autre inconvénient vient du fait que la gestion des données reste comme auparavant sur le système de bases de données non relationnelles. La réécriture des applications existantes n'est pas nécessaire. Les nouvelles applications sont réalisées dans le langage cible relationnel à l'aide d'outils de développement appropriés.*

Variante de migration : coexistence temporaire

- *Maintenance cohérente des bases de données hétérogènes par des règles de duplication (Variante 4) : seules les modifications dans des bases de données non relationnelles sont dupliquées vers le système de bases de données relationnelles. Inversement, en cas de besoin, les changements dans des bases de données relationnelles peuvent être dupliqués vers le système de bases de données non relationnelles. Dans les deux cas il faut définir des règles de duplication qui mettent en correspondance les structures différentes des deux bases de données. Au cours du développement de nouvelles applications, la coexistence des bases de données relationnelles et non relationnelles pendant une durée limitée permet de convertir des ensembles de données courantes ou périodiquement mises à jour, ainsi que des applications existantes sans subir la pression du temps.*

Dans la suite nous traiterons en détail la solution de coexistence, car elle est souvent appliquée dans la pratique et son importance est

indéniable, par exemple, dans la construction d'un entrepôt de données (voir la section 6.5).

5.4.2 Duplication des bases de données sous contrôle du système

Dans la modélisation des données, notre maxime est d'éviter des informations redondantes qui sont la cause des anomalies et des problèmes d'incohérence. Cependant, en pratique nous ne cherchons pas de manière systématique à éliminer les redondances de données. Dans la conception physique des bases de données relationnelles, il existe même des situations où les tables non normalisées présentent des avantages. En effet, le système peut y gagner en performance car les informations redondantes dans les tables permettent d'opérer des sélections plus simples et plus efficaces que des jointures coûteuses. C'est pourquoi, en vue d'optimiser les requêtes, nous acceptons souvent des compromis dans la conception physique des bases de données.

Avantages de la redondance physique des données

Une autre forme de redondance des données se présente lorsque des systèmes de bases de données hétérogènes doivent cohabiter ou que la transition vers un système cible relationnel s'étend sur une longue période. *Pour garantir la cohérence des données, le contrôle de ce type de redondance par le système s'impose* lors de la duplication d'une base de données non relationnelle vers un système relationnel, et vice versa.

La coexistence garantit la redondance des données contrôlée par le système

La duplication d'une base de données non relationnelle vers une base de données relationnelle comprend trois phases représentées dans la figure 5-11 : la phase de définition, la phase d'initialisation et la phase de transfert.

Les trois phases de la solution de coexistence

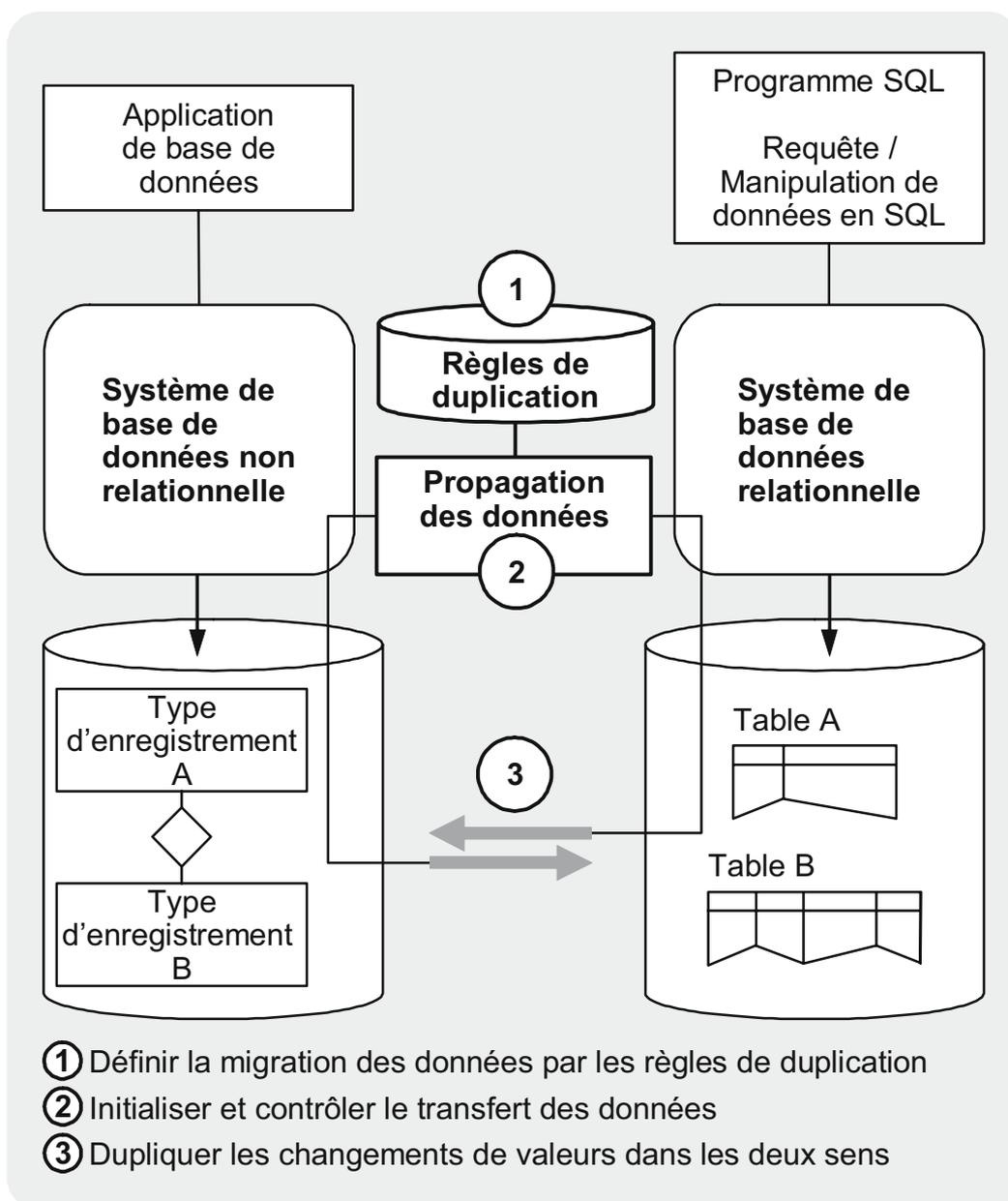
■ *Phase de définition* : des règles de duplication sont établies et stockées dans le catalogue des données. Elles définissent la conversion des types d'enregistrements en tables. Les enregistrements directement dépendants dans une base de données non relationnelle se traduisent par la création de tables dotées d'une règle d'intégrité référentielle appropriée.

Stocker les règles de duplication dans le catalogue de données

Préparer des ensembles de données identiques au départ

- *Phase d'initialisation* : nous n'avons pas besoin d'adapter ou d'élargir un programme applicatif qui interroge une base de données non relationnelle. C'est uniquement avant le premier usage des données dupliquées qu'il faut charger au préalable le contenu existant de la base de données non relationnelle dans des tables correspondantes. Ainsi, au départ, la base de données relationnelle dispose d'un *contenu identique* à celui de la base de données non relationnelle à dupliquer.

Figure 5-11
La redondance des données sous contrôle du système dans leur propagation



- *Phase de transfert* : en vertu des règles de duplication, toutes les opérations de mise à jour dans la base de données non relationnelle seront, au moment de leur exécution, automatiquement traduites en requêtes relationnelles (duplication synchrone) en vue d'effectuer *uniquement des modifications de valeurs dans le système relationnel*. Une autre approche consiste à regrouper les modifications de la base de données non relationnelle et à les reporter dans le système relationnel à des moments déterminés (duplication asynchrone). En outre, les règles de duplication déterminent les types d'enregistrements à dupliquer vers la base de données relationnelle ainsi que les champs d'un enregistrement à transférer vers les attributs correspondants d'une table.

Modifier les valeurs en mode synchrone ou asynchrone

La duplication des bases de données non relationnelles vers des systèmes relationnels permet de réaliser de nouvelles applications à l'aide de langages ou d'outils de développement issus de la technologie relationnelle. La variante de duplication inverse est nécessaire lorsque certaines portions d'un système applicatif doivent être rénovées par la technologie relationnelle et qu'en parallèle, les bases de données non relationnelles doivent continuer à fournir des données à d'autres fonctions du système. Naturellement, dupliquer les modifications d'une base de données relationnelle vers la base de données non relationnelle d'un système source se justifie seulement si les programmes d'application existants dépendent de ces ensembles de données. Ainsi, *la duplication inverse préserve les investissements* du passé consacrés au développement de nombreux programmes d'application que nous ne pouvons pas réécrire du jour au lendemain. En outre, dans la perspective des risques de la migration, l'intérêt de la coexistence des bases de données relationnelles et non relationnelles tient au fait qu'elle permet la transition par étapes vers la technologie relationnelle.

La duplication inverse réduit les risques et préserve les investissements

5.5 Principes de la planification de l'intégration et de la migration

L'intégration et la migration doivent être planifiées à long terme

L'usage des technologies du Web, le remplacement d'un système de bases de données et le choix d'une variante d'intégration et de migration adéquate exigent une planification rigoureuse car un tel projet engage des investissements considérables. Le retrait des systèmes hérités doit aussi être l'occasion de *corriger les faiblesses accumulées jadis dans les ensembles de données* et d'*aligner l'architecture de données sur les objectifs stratégiques de l'entreprise*. C'est pourquoi une architecture de données globale de l'entreprise constitue la base d'un véritable plan d'intégration et de migration. Elle représente une abstraction des secteurs d'excellence parmi les domaines d'activité de l'entreprise et se focalise sur ses besoins d'information à long terme (voir 2.6).

Le retrait des systèmes hérités relève de la responsabilité de l'entreprise

La planification de l'intégration et de la migration ne peut pas être uniquement l'affaire d'une équipe d'experts. En revanche, elle nécessite la coopération des différents secteurs d'activités de l'entreprise, car la rénovation des systèmes d'information existants et le passage à une nouvelle technologie de bases de données qui les accompagne signifient de gros investissements à consentir. Par conséquent, dans un projet d'intégration et de migration des données, il est primordial de fixer certains principes fondamentaux qui dictent les règles de conduite à suivre dans les activités de développement quotidiennes.

Définir les principes de la planification axés sur les secteurs d'excellence de l'entreprise

Les principes suivants sont définis dans le cadre des besoins d'une société internationale dans le secteur des services. Quoiqu'ils ne puissent pas se transposer tels quels à d'autres entreprises, c'est néanmoins un exemple qui montre comment concevoir et réaliser l'intégration et la migration des ensembles de données hétérogènes dans le cadre des secteurs d'activité d'excellence d'une entreprise.

Principe 1

Définir une architecture de données globale de l'entreprise
L'intégration et la migration des données doivent se réaliser dans le cadre d'une architecture de données globale de l'entreprise.

Ce principe qui paraît évident est néanmoins controversé dans la pratique. Il est frappant de constater qu'on hésite souvent à consentir les efforts nécessaires pour structurer les données à l'échelle de l'entreprise. Une telle attitude réduit l'intégration et la migration éventuelle des données aux aspects purement techniques. Or, si l'entreprise veut qu'un changement de la technologie des bases de données aille de pair avec la mise en place d'une architecture de données à long terme, elle doit appliquer ce premier principe dès le début.

Le seul changement de la technologie ne suffit pas

Adopter la technologie des bases de données relationnelles ou post-relationnelles

Principe 2

La mise en œuvre de nouvelles applications et fonctions d'affaires doit se baser sur la technologie des bases de données relationnelles et relationnelles orientées objet.

Devant la complexité des systèmes d'information existants et de l'usage du Web, la standardisation des différentes approches architecturales et la réduction des interfaces s'imposent. Dans le développement des applications, il est par conséquent logique que des principes obligatoires soient définis et appliqués en matière de technologie des bases de données et de l'usage du Web. Autrement, les unités de développement, constamment exposées à la pression des délais, sont trop souvent confrontées aux problèmes de décisions et continuent d'utiliser par la force de l'habitude des méthodes et techniques traditionnelles.

Réduire la complexité des systèmes et les interfaces

Interdire les extractions incontrôlées de données

Principe 3

Les ensembles de données créés par extraction à l'usage des départements doivent être considérés uniquement comme des exceptions dûment autorisées.

Dans une société de services, l'extraction régulière de données contenues dans des bases de données existantes est extrêmement problématique et ne devrait être envisagée que dans le cadre d'une stratégie d'entrepôt de données adéquate (voir aussi 6.5). L'extraction incontrôlée des données productives est incompatible avec l'objectif *d'assurer un service 24 heures sur 24*. En outre, la création périodique des ensembles de données soulève de graves problèmes de

L'extraction de données est admise seulement dans le cadre d'une stratégie d'entrepôt de données

coordination entre les secteurs d'activité. *L'intégrité des données n'est garantie que pour des informations actualisées ou relatives à des instants précis.* Par conséquent, les départements rencontreront des difficultés insurmontables quand il s'agit d'analyser des données actualisées combinées aux ensembles de données extraites périodiquement.

Principe 4 Respecter les normes ISO

Les normes internationales de l'ISO (International Organization for Standardization) doivent être prises en considération.

Saisir l'occasion d'introduire les normes ISO

Il est pratiquement inconcevable d'introduire dans des milliers de programmes d'application les codes internationaux de pays, de branches, de monnaies ou d'adresses, car l'effort à accomplir est considérable. C'est pourquoi le passage à une nouvelle technologie de base de données ou la mise en œuvre des technologies du Web offre l'unique occasion d'appliquer dès le début les normes ISO aux nouvelles structures de données.

Principe 5 Adopter les modèles de données par branches et les bases de données en ligne

Dans un plan d'intégration et de migration des données il faut envisager l'adoption des modèles de données par branche et l'accès aux bases de données en ligne disponibles sur le marché.

Considérer les fournisseurs de données externes et les courtiers en information

Au cours des dernières années, des modèles de données et des cadres de travail (*framework*, en anglais) par domaines d'application arrivent sur le marché. Cette offre commerciale est complétée par des bases de données en ligne stockant des informations que nous pouvons acheter, telles que données économiques, analyses de marché et bilans de sociétés. Ces fournisseurs de données externes doivent entrer en considération lorsque les besoins le justifient dans un projet de remplacement des systèmes d'information.

Nous élaborons à présent un plan d'intégration et de migration fondé sur les principes énoncés précédemment. Ce plan doit tenir compte des bases de données existantes, héritées du passé. Il comporte les étapes suivantes :

- *Affiner l'architecture de données globale de l'entreprise* : tous les ensembles d'entités et de liens qui font l'objet de l'intégration ou de la migration doivent être affinés par rapport à l'architecture de données globale de l'entreprise, puis transformés en un schéma de base de données relationnelle. *Assurer la conformité du système cible à l'architecture de données*
- *Fixer les règles de mise en correspondance des système source et cible* : le schéma de base de données relationnelle du système cible, dérivé de l'architecture de données globale de l'entreprise, est comparé aux bases de données existantes du système source. À chaque divergence, il faut déterminer parmi les types de transformation disponibles, celui qui mène au schéma de base de données relationnelle du système cible désiré. *Sélectionner les types de mise en correspondance*
- *Choisir les variantes d'intégration et de migration* : il faut décider d'une stratégie d'intégration et de migration adaptée à chaque situation particulière. Elle peut être une combinaison de plusieurs variantes. En outre, dans la solution de coexistence, il faut opter pour l'un des deux modes de duplication, synchrone ou asynchrone. Avec la duplication synchrone, les données seront toujours actualisées et cohérentes dans les deux bases de données, relationnelle et non relationnelle. La duplication asynchrone est concevable si la propagation périodique des modifications vers la base de données relationnelle du système cible s'avère suffisante. *Il est possible de combiner les variantes d'intégration et de migration*
- *Résoudre les problèmes de mise en correspondance* : si aucune règle de conversion ne peut s'appliquer à certains types d'enregistrements particuliers, deux solutions sont possibles. La première consiste à rénover la base de données non relationnelle (ce qui entraîne l'adaptation éventuelle de certaines applications). La seconde consiste à développer des règles de conversion spécialement destinées aux types d'enregistrements en question. Dans le système cible nous devons aussi spécifier des contraintes d'intégrité référentielle qui correspondent aux règles de traitement définies dans les bases de données du système source. *Corriger le système source en cas de besoin*

Aligner l'interface utilisateur sur l'architecture de données globale de l'entreprise

- *Définir les schémas externes* : l'application des règles de conversion peut conduire à la création de tables dans la base de données relationnelle du système cible, qui ne reflètent pas les recommandations dans l'architecture de données globale de l'entreprise. Dans ce cas, nous devons définir des schémas externes appropriés à l'aide de vues (*views*, en anglais). Ce puissant concept lié aux bases de données relationnelles permet de présenter, au niveau de l'interface utilisateur, une vue des données conforme à l'architecture de données globale de l'entreprise.

La persévérance est le maître mot dans un projet de conversion

Les étapes de la planification que nous venons de décrire sommairement illustrent les multiples facettes d'un projet d'intégration des données et de migration vers la technologie des bases de données relationnelles ou post-relationnelles. Nous pouvons combiner plusieurs variantes d'intégration et de migration. Pour certaines applications dont la performance est un facteur critique, il peut même s'avérer inopportun de changer de technologie. L'aspect temporel joue aussi un rôle important dans un projet de conversion, car les systèmes applicatifs ont normalement un cycle de vie pluriannuel avant qu'une révision de grande envergure ne s'impose.

5.6 Notes bibliographiques

Apparition notable de la récente littérature sur XML et les bases de données orientées web

Des livres traitant des bases de données en relation avec le Web et XML ont été récemment publiés. Loeser (2001) présente la mise en œuvre des bases de données relationnelles-objet dans les systèmes d'information orientés web. Plusieurs experts en matière de services Web et de bases de données ont contribué à l'édition d'un ouvrage collectif de Rahm et Vossen (2003). Meier et Wüst (2003) abordent les bases de données orientées objet et relationnelles-objet, et présentent différents systèmes de bases de données post-relationnelles. Dietrich et Urban (2005), Kazakos et al. (2002), Klettke et Meyer (2003), Schöning (2003), Williams et al. (2001) s'intéressent au thème XML et les bases de données. Leurs ouvrages, riches en exemples, traitent du langage de balisage XML, du schéma XML, des langages de requête tels que XQuery, et de l'échange de

données entre les documents XML et les systèmes de bases de données.

Au milieu des années 1970, l'avènement des systèmes de bases de données relationnelles a donné lieu à la publication de plusieurs travaux de recherche sur la mise en correspondance des schémas de bases de données. Chen (1976) développe au niveau logique le processus de transformation du modèle entité-association en un schéma de base de données relationnelle, hiérarchique ou en réseau. Dans son ouvrage contenant une sélection de thèmes sur les bases de données relationnelles, Date (1986) aborde la problématique de la mise en correspondance des bases de données hiérarchiques et relationnelles. Gillenson (1990) établit les règles de conversion pour les différents systèmes de bases de données physiques.

Brodie et Stonebraker (1995) présentent les stratégies de migration des systèmes d'information obsolètes. Meier et Dippold (1992) traitent de la migration et de la coexistence des bases de données hétérogènes. Meier et al. (1993) et Meier (1997) illustrent la préservation des investissements dans un projet de migration des bases de données. L'ouvrage de Dippold et al. (2001), consacré à la gestion des données, définit les éléments essentiels de la migration des bases de données. Hüsemann (2002) décrit dans son livre la migration des bases de données relationnelles vers la technologie orientée objet et présente une méthodologie de migration ainsi que des mesures de soutien logiciel.

Littérature de recherche sur la conversion des bases de données

Ouvrages spécifiques sur la migration des bases de données

6 Les systèmes de bases de données post-relationnelles

6.1 Évolution future : pourquoi et dans quelle direction ?

De nos jours, la technologie relationnelle domine largement le marché des bases de données. La réussite de ce développement se poursuivra sans doute encore dans les années à venir. Néanmoins, nous nous interrogeons sur l'évolution future dans ce domaine en pensant notamment aux systèmes de bases de données réparties, temporelles, déductives, sémantiques, aux systèmes orientés objet, aux bases de données floues, aux systèmes capables de gérer des versions, etc. Que dissimulent ces jargons ? Dans le présent chapitre, nous aborderons quelques-uns de ces concepts, et nous présenterons une sélection subjective de méthodes émergentes et de tendances futures du développement.

L'apparition de nouveaux besoins avec l'élargissement des domaines d'application révèle les limites du modèle relationnel classique et des systèmes de bases de données relationnelles. Les constructeurs de matériels informatiques et les éditeurs de logiciels n'ont manifestement pas pu anticiper un développement aussi rapide de la technologie relationnelle. Ainsi, par exemple dans les applications orientées web, ils doivent maintenant mettre au point des méthodes et techniques pour stocker et gérer dans une même base de données non seulement des données formatées, mais encore des documents textuels, des graphiques et des images. Une extension de la technologie des bases de données s'avère également indispensable pour des applications du domaine technique, par exemple dans la conception des circuits intégrés assistée par l'ordinateur, dans le contrôle des processus et de la fabrication des machines et des composants, dans le stockage et le traitement des photos retransmises par satellite ou dans la production de plans et de cartes géographiques.

La technologie des bases de données relationnelles occupe une place dominante sur le marché

Faiblesses des systèmes de bases de données relationnelles

Architecture d'applications unifiée

À l'heure actuelle, il existe sur le marché une vaste gamme de produits pour bases de données proposant de nouvelles fonctionnalités. La jungle des annonces met souvent le professionnel dans l'embarras du choix. En outre, il est généralement difficile d'évaluer le rapport du coût de mise en œuvre aux avantages économiques. C'est pourquoi bon nombre d'entreprises ont encore grand besoin de leur matière grise pour définir une architecture des applications orientée vers le futur et choisir leurs produits de manière pertinente. En d'autres termes, leur réussite dépend des concepts d'architecture et des stratégies de migration clairement définis pour une mise en œuvre optimale de la technologie post-relationnelle.

Avènement des systèmes de bases de données post-relationnelles

Dans les prochaines sections, nous présenterons une sélection de problèmes et proposerons des approches de solution en explorant les orientations de développement des futurs systèmes de bases de données. Parmi les besoins qui ne trouvent pas encore de solution à l'heure actuelle, certains peuvent être résolus par une extension ponctuelle des systèmes de bases de données relationnelles alors que d'autres doivent être satisfaits en introduisant des concepts et méthodes fondamentalement nouveaux. Ces deux tendances caractérisent l'évolution actuelle des *systèmes de bases de données post-relationnelles*.

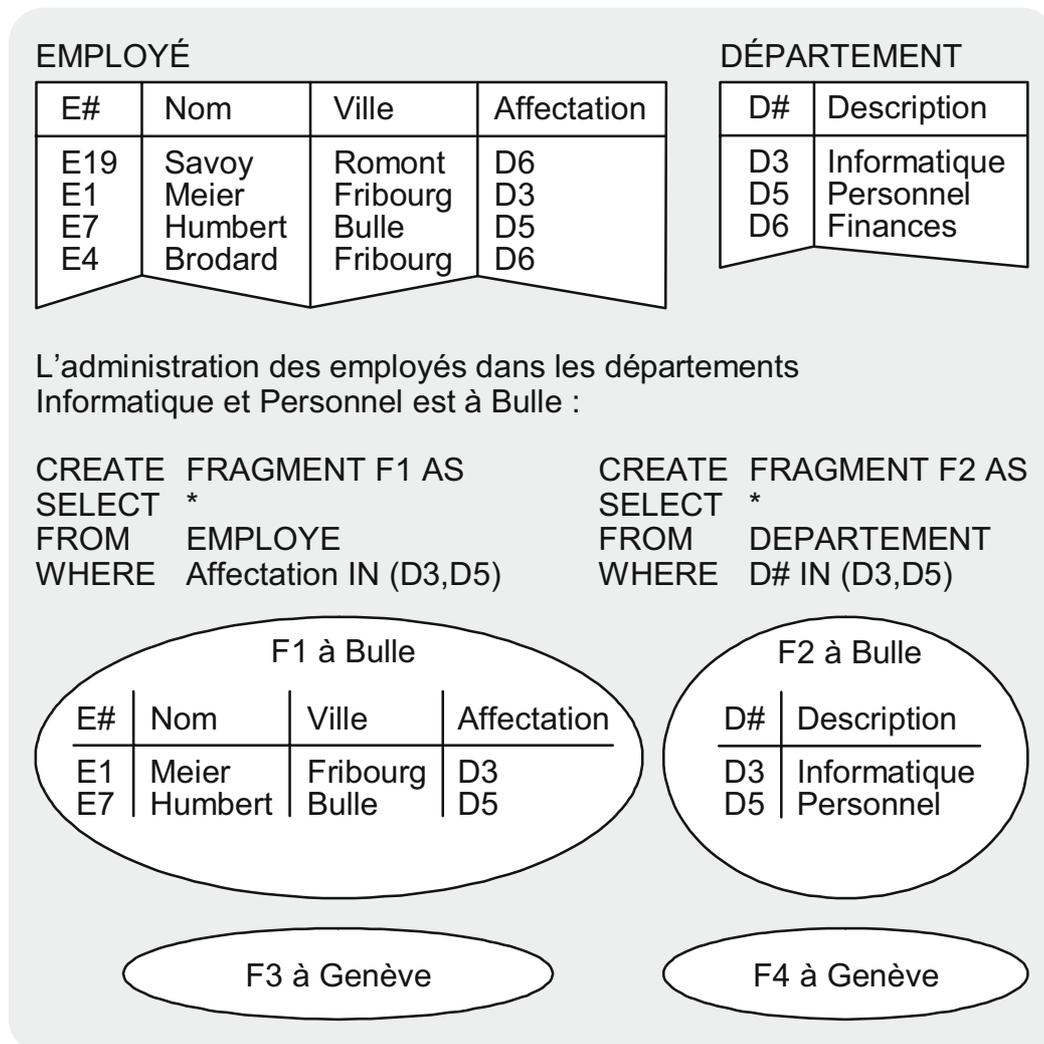
6.2 Les bases de données réparties

Les bases de données réparties reposent sur un schéma logique unifié

Dans une entreprise, privée ou publique, les *bases de données réparties* ou distribuées (*distributed databases*, en anglais) permettent de réaliser des applications qui nécessitent le stockage, la maintenance et le traitement des données en plusieurs endroits différents. Une *base de données est décentralisée ou répartie* lorsqu'elle est modélisée par *un seul schéma logique de base de données*, mais implémentée dans *plusieurs fragments de tables physiques sur des ordinateurs géographiquement dispersés*. L'utilisateur d'une base de données répartie se focalise sur sa vue logique des données et n'a pas besoin de se préoccuper des fragments physiques. C'est le système de bases de données qui se charge lui-même d'exécuter les opérations, soit localement, soit en les distribuant sur plusieurs ordinateurs en cas de besoin.

La figure 6-1 montre un exemple simple de base de données répartie. Le partitionnement des tables EMPLOYÉ et DÉPARTEMENT en plusieurs fragments physiques est une tâche cruciale dévolue à l'administrateur de bases de données, et non pas à l'utilisateur. Dans cet exemple, admettons que les départements Informatique et Personnel se trouvent à Bulle, et le département Finances à Genève. Le fragment F1 est une sous-table de la table EMPLOYÉ et contient uniquement les employés des départements Informatique et Personnel. Le fragment F2 qui provient de la table DÉPARTEMENT contient la description des départements situés à Bulle. Les deux fragments F3 et F4 contiennent respectivement les employés et les départements situés à Genève.

L'administrateur de bases de données définit les fragments physiques



*Figure 6-1
Fragmentation horizontale des tables EMPLOYÉ et DÉPARTEMENT*

Nous parlons de *fragmentation horizontale* lorsqu'une table est partitionnée horizontalement, conservant ainsi la structure originale

Fragmentation horizontale

des lignes de la table considérée. En principe, les fragments distincts ne doivent pas se chevaucher et leur union permet de reconstituer la table originale.

*Fragmentation
verticale et
fragmentation
mixte*

Au lieu du partitionnement horizontal, une table peut *se diviser en fragments verticaux* qui regroupent chacun plusieurs colonnes, incluant la clé d'identification. Les tuples sont donc fragmentés. Par exemple, pour garantir la confidentialité des données, certains attributs de la table EMPLOYÉ tels que le salaire, les niveaux de qualification, le potentiel de développement, etc. devraient être stockés dans un fragment vertical au département Personnel. Les attributs restants constituent d'autres fragments qu'on peut stocker dans les différents départements. Il est possible de concevoir un partitionnement mixte composé de fragments horizontaux et verticaux.

*Il faut assurer
l'autonomie locale*

Une fonction importante d'un système de bases de données réparties est de *garantir l'autonomie locale*. L'utilisateur doit pouvoir travailler avec ses données locales de manière autonome, même si d'autres sites du réseau sont indisponibles¹.

*Le concept de
transaction répartie
doit être mis en
œuvre*

Outre l'autonomie locale, le *principe du traitement décentralisé* est primordial. En vertu de ce principe, un système de bases de données doit être capable de traiter localement des requêtes sur les différents sites du réseau. Dans les applications décentralisées qui traitent des données réparties sur plusieurs fragments, le système de bases de données doit permettre la lecture et la modification des tables à distance. Pour ce faire, il faut mettre en œuvre le concept de transaction et de reprise réparties en introduisant dans les systèmes de bases de données réparties des mécanismes de contrôle spécifiques que nous étudierons ci-après en détail.

*Comment
fonctionne la
validation en deux
phases ?*

La cohérence des bases de données décentralisées est garantie en permanence grâce au *protocole de validation en deux phases (two-phase commit protocol, en anglais)*. Ce protocole repose sur un programme de coordination globale qui supervise les systèmes de

¹ L'extraction périodique de clichés instantanés (*snapshots, en anglais*) augmente le degré d'autonomie locale.

bases de données locaux selon le mécanisme suivant : dans la première phase, toutes les tâches participantes signalent au programme de coordination que leurs transactions locales sont terminées. Le programme de coordination leur répond en envoyant un «PREPARE FOR COMMIT» pour préparer la conclusion normale de la transaction. Suite à cette invitation, tous les programmes participants appliquent les mises à jour définitives aux bases de données locales. Chaque participant communique le résultat de cette opération au programme de coordination centrale en émettant un «OK» ou «NOT OK». Dans la deuxième phase, le programme de coordination envoie un «COMMIT» s'il a reçu des «OK» de tous les participants. En revanche, si au moins un des participants a signalé l'échec de l'opération par un «NOT OK» dans la phase 1, le programme de coordination émet un «NOT OK» dans la phase 2. Dans ce cas, les participants ayant terminé avec succès leur mise à jour auparavant doivent maintenant défaire leurs transactions. Le protocole de validation en deux phases présente l'avantage suivant : soit toutes les tâches participantes se terminent avec succès et la base de données est mise à jour correctement, soit aucune modification n'est intervenue dans la base de données en cas d'échec.

La *stratégie de traitement interne des requêtes de bases de données réparties* joue un rôle important. Considérons l'exemple dans la figure 6-2 où nous désirons extraire une liste des noms d'employés et des descriptions de leurs départements. La requête d'interrogation est une commande SQL sous sa forme usuelle sans aucune référence aux fragments. Il incombe au système de bases de données de définir maintenant sa stratégie d'exécution optimale de notre requête décentralisée. Les tables EMPLOYÉ et DÉPARTEMENT sont toutes deux partitionnées en fragments stockés à Bulle et à Genève. C'est pourquoi *certaines opérations* doivent s'exécuter *localement et en parallèle*. Chaque site opère de manière autonome la jointure d'un fragment de la table EMPLOYÉ et d'un fragment de la table DÉPARTEMENT. À la fin des opérations locales, l'union des résultats partiels produit la liste désirée.

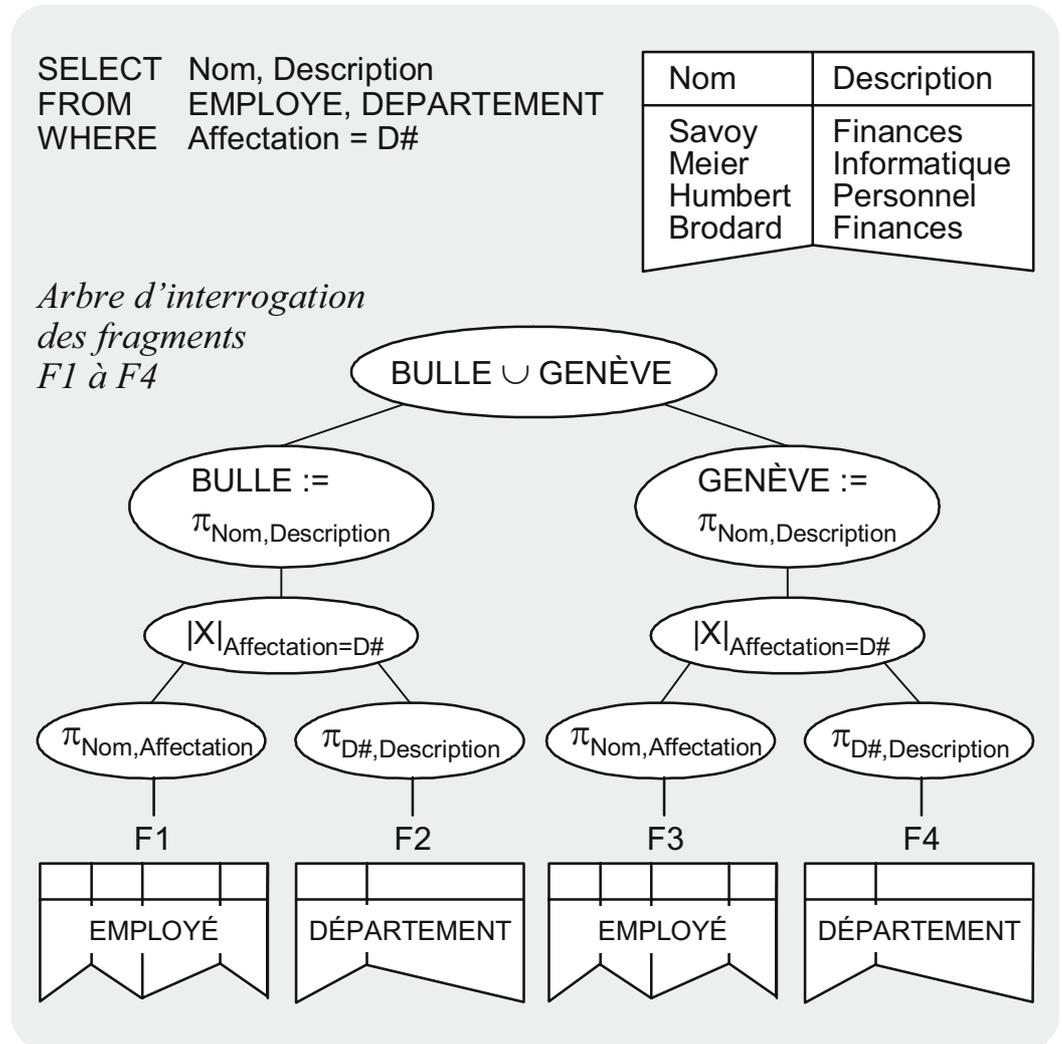
Pour optimiser l'arbre d'interrogation, chaque site effectue d'abord des projections sur les attributs Nom et Description, qui

*Traitement optimal
des requêtes
réparties*

*Un exemple de
requête répartie*

devront figurer dans le résultat final. Ensuite, l'opérateur de jointure est appliqué séparément aux fragments de tables réduits à Bulle et à Genève. Enfin, avant leur union, les deux résultats intermédiaires sont réduits encore une fois par une projection sur les attributs désirés, Nom et Description.

Figure 6-2
Arbre
d'interrogation
optimisé dans une
stratégie de
jointure répartie



Augmenter le
parallélisme dans
le traitement des
requêtes réparties

En principe, pour traiter des requêtes décentralisées, la stratégie classique consiste à évaluer le plus tard possible les opérateurs d'union et de jointure². Cela permet d'atteindre un *haut degré de parallélisme* dans l'exécution et d'augmenter la performance du traitement des requêtes réparties. Ainsi, pour mettre en œuvre la stratégie d'optimisation, il faut placer les opérateurs d'union le plus

² Une opération dite de semi-jointure peut également contribuer à réduire le coût de transmission au travers d'un choix approprié de projections.

près possible du nœud racine dans un arbre d'interrogation, alors que les sélections et les projections doivent se trouver près des feuilles de cet arbre.

La stratégie de l'union, représentée en figure 6-2, est optimale car elle permet d'utiliser en parallèle les fragments situés sur des sites différents. Normalement, dans la conception d'une base de données répartie on n'arrive pas à créer des fragments de manière à pouvoir générer localement les résultats intermédiaires de toutes les requêtes futures. C'est pourquoi, au cours du temps, un système de base de données répartie doit chercher à adapter ses fragments à chaque situation particulière sur la base du profil des requêtes. Parfois, il est opportun d'implémenter des fragments de tables redondants sous contrôle du système de bases de données lorsque cette duplication permet d'assurer un haut degré de disponibilité du système et des informations.

La duplication augmente le degré de disponibilité

Système de bases de données réparties

Un système de gestion de bases de données réparties doit remplir les conditions suivantes :

- il repose *sur un seul schéma de base de données logique dont la mise en œuvre consiste en plusieurs fragments physiques résidant sur des ordinateurs géographiquement dispersés* ;
- il garantit la *transparence dans la dispersion des bases de données* de sorte que les requêtes ad hoc et les programmes d'application ne doivent pas se préoccuper de la répartition physique des données, c'est-à-dire de la création des fragments ;
- il assure l'autonomie locale en permettant le traitement local des données décentralisées, même si d'autres sites du réseau ne sont pas accessibles ;
- il garantit la cohérence des bases de données réparties grâce au *protocole de validation en deux phases*, et optimise les requêtes et les manipulations de données réparties grâce à un programme de coordination.

Principales caractéristiques d'un système de bases de données réparties

La conception des bases de données réparties est un processus exigeant

Les premiers prototypes de systèmes de bases de données réparties étaient développés au début des années 1980. De nos jours, des produits sont déjà commercialisés, qui satisfont dans une large mesure aux critères d'un SGBD réparti énoncés ci-haut. Ainsi, la répartition des tables entières sur différents ordinateurs est supportée, alors que la fragmentation des tables n'est possible que de manière limitée. En outre, des progrès sont encore à réaliser tant dans la conception optimale d'une base de données répartie que dans l'optimisation des manipulations de données décentralisées.

6.3 Les bases de données temporelles

Les systèmes de bases de données actuels ignorent les concepts temporels

De nos jours, les systèmes de bases de données relationnelles sont conçus dans le but de gérer les informations du présent (informations courantes) stockées dans des tables. Si, en revanche, l'utilisateur désire interroger et analyser ses bases de données relationnelles en considérant le facteur temps, il est personnellement responsable de la gestion et de la maintenance des données historiques et futures. Le système de bases de données ne lui offre aucun support pour le stockage, la recherche ou l'analyse des informations comprenant des aspects temporels.

Point temporel et intervalle de temps

La notion du temps introduit une *grandeur physique unidimensionnelle* dont les valeurs sont globalement ordonnées. Ceci permet donc de comparer deux valeurs quelconques sur l'axe du temps pour déterminer si l'une est «inférieure» (antérieure) ou «supérieure» (postérieure) à l'autre. Les données temporelles qui nous intéressent peuvent non seulement désigner un jour ou un instant précis (par exemple «le 1er avril 1989 à 14 heures»), mais encore une durée sous la forme d'un intervalle de temps. Par exemple, l'ancienneté d'un employé s'exprime en nombre d'années. Il convient de noter qu'une donnée temporelle peut être interprétée comme un point temporel ou comme une durée selon le point de vue de l'utilisateur.

Le temps valide est un concept clé

Dans une *base de données temporelle* (*temporal database*, en anglais), les valeurs de données, les tuples ou des tables entières sont définis *par rapport à l'axe du temps*. Une donnée temporelle peut

revêtir plusieurs significations différentes pour un objet particulier dans la base. Ainsi, un *temps valide* peut désigner soit un instant précis auquel se produit un événement déterminé, soit un intervalle de temps qui définit la durée de validité des valeurs de données associées. Par exemple, l'adresse d'un employé est valable jusqu'au prochain changement de domicile.

Un autre type de donnée temporelle est le *temps transactionnel* qui désigne le point temporel auquel un objet spécifique a été introduit, modifié ou détruit dans une base de données. Normalement, c'est le système de base de données qui gère lui-même les temps transactionnels au moyen d'un journal. Pour cette raison, dans la suite de notre développement, quand nous parlons de temps, il s'agira toujours du temps valide.

Pour enregistrer les dates de validité, la plupart des systèmes de bases de données relationnelles supportent à l'heure actuelle deux types de données : le type DATE *permet de représenter une date* composée de l'année, du mois et du jour, le type TIME *définit le temps en heures, minutes et secondes*. Pour représenter une durée, aucun type de donnée particulier n'est requis, car les nombres entiers et décimaux suffisent. Le calcul avec des données temporelles s'effectue donc de manière naturelle.

Dans l'exemple en figure 6-3, nous avons complété la table EMPLOYÉ par deux attributs, la date de naissance et la date d'engagement. Ce sont donc des attributs liés au temps, grâce auxquels nous pouvons par exemple demander au système de produire une liste des employés engagés avant l'âge de vingt ans.

Jusqu'à présent, la table EMPLOYÉ donne une image actuelle de la base de données à un instant précis. Par conséquent, nous ne pouvons l'interroger ni sur le passé ni sur le futur, car nous ne disposons d'aucune information sur le *temps valide* des valeurs de données. Si, par exemple, l'employé Humbert change de fonction, nous écrasons la description existante de sa fonction par une nouvelle qui devient la valeur courante. En revanche, nous ne savons pas depuis quand ni jusqu'à quelle date l'employé Humbert occupait une fonction particulière.

Le temps transactionnel est journalisé

Les types de données temporels DATE et TIME

Exemple d'attributs liés au temps

Une table contenant des dates n'est pas nécessairement temporelle

Figure 6-3
La table EMPLOYÉ
avec des données
de type DATE

E#	Nom	Date de naissance	Ville	Date d'embauche	Fonction
E19	Savoy	1948-02-19	Romont	1979-10-01	Comptable
E1	Meier	1958-07-09	Fribourg	1984-07-01	Analyste
E7	Humbert	1969-03-28	Bulle	1988-01-01	Chef du personnel
E4	Brodard	1952-12-06	Fribourg	1978-04-15	Réviseur

Extraire les employés engagés par l'entreprise avant l'âge de vingt ans :

```
SELECT E#, Nom
FROM EMPLOYE
WHERE Date_Embauche -
      Date_Naissance <= 20
```

E#	Nom
E7	Humbert

L'estampille
indique la validité

Pour exprimer la *validité d'une entité*, il faut introduire généralement deux attributs. Le point temporel «valable dès» (*valid from*, en anglais) détermine l'instant à partir duquel un tuple ou une valeur de donnée est valide. L'attribut «valable jusqu'à» (*valid to*, en anglais) indique le point temporel où la période de validité prend fin. Au lieu de définir deux instants *VALID_FROM* et *VALID_TO*, l'attribut *VALID_FROM* suffit sur l'axe du temps. En effet, chaque instant *VALID_TO* est implicitement déterminé par le prochain instant *VALID_FROM*, car les intervalles de validité d'une entité ne se chevauchent pas.

Les tables
temporelles
permettent de
connaître
l'historique des
données

Dans la table temporelle *TEMP_EMPLOYÉ* à la figure 6-4, la colonne associée à l'attribut *VALID_FROM* contient les dates marquant le début de validité des tuples de l'employé E1 (Meier). Cet attribut *doit* être la clé qui sert à identifier de manière unique non seulement les valeurs de données courantes, mais aussi des valeurs passées et futures. Les quatre tuples dans la table *TEMP_EMPLOYÉ* sont interprétés comme suit : l'employé Meier habitait à Bulle du 1.7.1984 au 12.9.1986, puis à Fribourg jusqu'au 31.3.1989 et de nouveau à Bulle dès le 1.4.1989. Depuis son entrée en fonction dans l'entreprise jusqu'au 3.5.1987 il était employé comme programmeur, puis du 4.5.1987 au 31.3.1989 comme analyste-programmeur, enfin dès le 1.4.1989 comme analyste. *TEMP_EMPLOYÉ* est effectivement une table temporelle, car outre les données courantes, elle contient aussi des valeurs se rapportant au passé. En particulier, elle apporte des

réponses aux requêtes d'interrogation qui ne portent pas uniquement sur l'instant présent ou sur des intervalles de temps actuels.

TEMP_EMPLOYÉ (Extrait)

E#	VALID_FROM	Nom	Ville	Date d'embauche	Fonction
E1	01.07.1984	Meier	Bulle	1984-07-01	Programmeur
E1	13.09.1986	Meier	Fribourg	1984-07-01	Programmeur
E1	04.05.1987	Meier	Fribourg	1984-07-01	Analyste-progr.
E1	01.04.1989	Meier	Bulle	1984-07-01	Analyste

Déterminer la fonction de l'employé Meier au 01.01.1988 :

SQL initial :

```
SELECT Fonction
FROM   TEMP_EMPLOYE A
WHERE  A.E# = E1 AND
       A.VALID_FROM =
( SELECT  MAX(VALID_FROM)
  FROM TEMP_EMPLOYÉ B
  WHERE  B.E# = E1 AND
        B.VALID_FROM <=
        '01.01.1988' )
```

SQL temporel :

```
SELECT Fonction
FROM   TEMP_EMPLOYE
WHERE  E# = E1 AND
       VALID_AT(01.01.1988)
```

Fonction
Analyste-progr.

À titre d'exemple, dans la figure 6-4 nous désirons connaître la fonction de l'employé Meier au 1^{er} janvier 1988. Au lieu de la commande SQL initiale qui exprime une requête imbriquée et fait appel à la fonction MAX (voir 3.4.1), nous pouvons imaginer un langage qui prend en charge les requêtes temporelles. Le mot-clé VALID_AT nous permet de spécifier un point temporel par rapport auquel s'effectuera la recherche de toutes les entrées valides.

Système de bases de données temporelles

Un système de gestion de bases de données temporelles

- prend en charge l'ensemble des valeurs de l'axe du temps comme *temps valide* en ordonnant chronologiquement les tuples ou les valeurs d'attributs, et

Figure 6-4
Extrait d'une table
temporelle
TEMP_EMPLOYÉ

*Un exemple de
requête portant sur
le passé*

*Propriétés
fondamentales
d'un système de
bases de données
temporelles*

- dispose *dans son langage des éléments temporels* qui permettent de composer des requêtes d'interrogation sur le futur, le présent et le passé.

Propositions de langage SQL temporel

Dans le domaine des *bases de données temporelles*, plusieurs modèles de langage sont proposés pour simplifier la formulation des requêtes temporelles et la manipulation de données liées au temps. En particulier, les opérateurs de l'algèbre relationnelle et du calcul relationnel doivent être étendus afin de rendre possible par exemple la jointure des tables temporelles. En outre, nous devons adapter les contraintes d'intégrité référentielle pour prendre en compte les aspects temporels.

Stockage des bases de données temporelles

Les données historiques et futures augmentent de manière sensible la taille des bases de données relationnelles, ce qui nécessite des méthodes de mise en œuvre sophistiquées. Dans la *méthode des différences*, seules les valeurs de données modifiées sont enregistrées comme différences, et non plus des tuples ou des tables entières. Pour ce faire, il faut développer des algorithmes capables de reconstruire la situation courante à partir des états antérieurs et des différences, et de traiter des requêtes portant sur le passé. Dans les laboratoires de recherche et de développement, des méthodes de stockage et des extensions du langage font déjà l'objet de prototypes avancés. Toutefois, peu de systèmes de bases de données commercialisés de nos jours prennent en charge les concepts temporels.

6.4 Les bases de données relationnelles-objet

Les bases de données relationnelles supportent difficilement les données structurées et semi-structurées

Dans un modèle relationnel, les informations sont stockées dans de simples tables. Si les données comportent une structure, nous appliquons les règles de normalisation pour les décomposer et les répartir ensuite sur des tables séparées dans une base de données relationnelle. Les groupes répétitifs ne sont pas admis à l'intérieur d'une table. En outre, l'utilisateur doit définir et gérer tous les liens entre les données au travers des attributs communs. Par conséquent, un tel système de bases de données ne peut pas tirer profit des

propriétés structurelles pour optimiser le stockage des données et le traitement des requêtes.

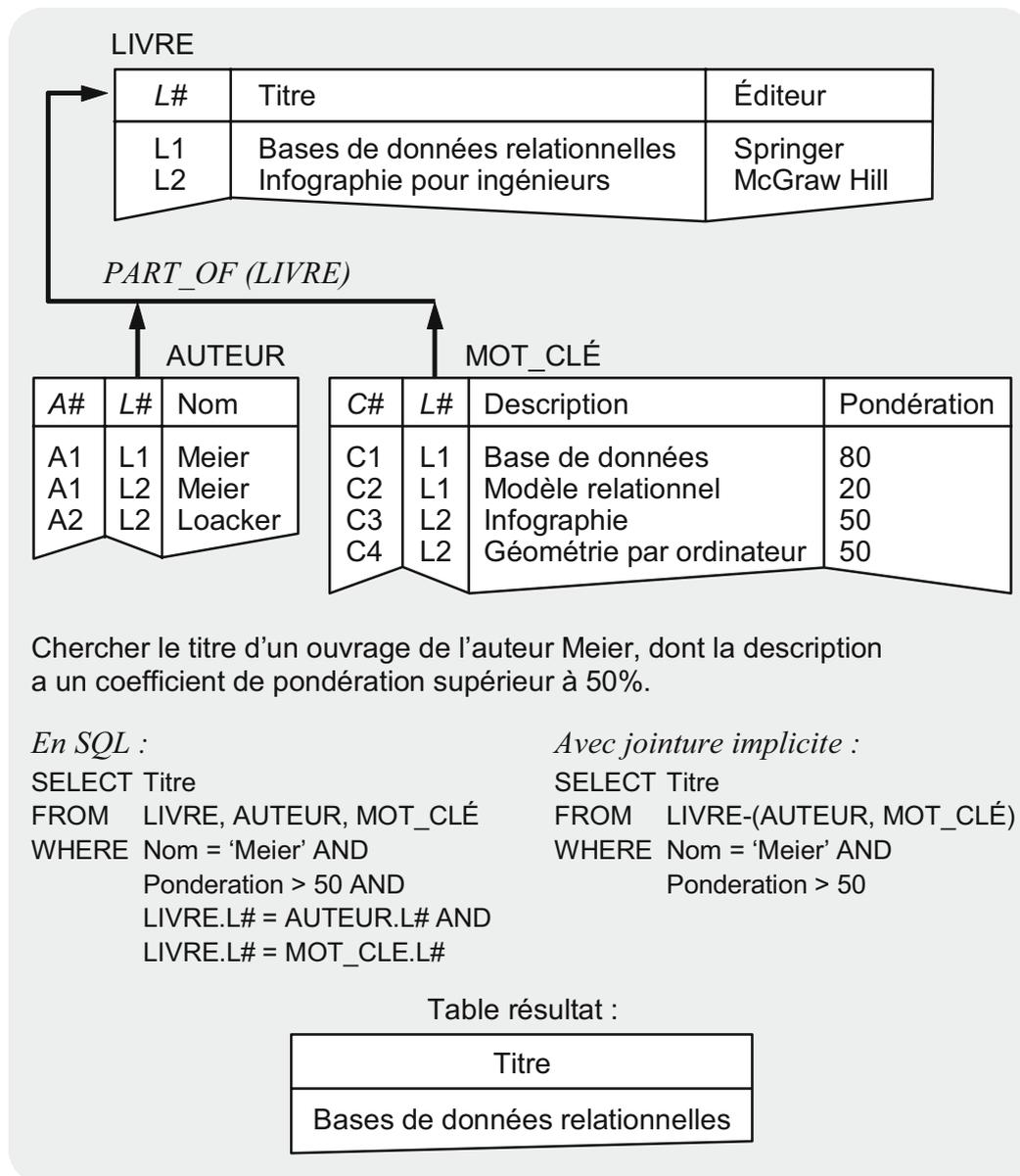


Figure 6-5
Interrogation d'un objet structuré avec et sans l'opérateur de jointure implicite

Considérons à titre d'exemple la création d'une base de données relationnelle pour gérer les livres dans une bibliothèque. À cette fin, nous devons créer plusieurs tables, dont trois sont présentées dans la figure 6-5 : le titre et l'éditeur de chaque ouvrage sont stockés dans la table LIVRE. Nous créons une table distincte AUTEUR pour décrire la paternité de chaque ouvrage, sachant que plusieurs auteurs peuvent contribuer à un ouvrage particulier et qu'inversement, un auteur peut rédiger plusieurs ouvrages. L'attribut «Nom» ne dépend pas fonctionnellement de la totalité de la clé définie par concaténation des

Un livre consiste en plusieurs tables et constitue un objet structuré

identifiants de l'auteur et du livre. C'est pourquoi la table AUTEUR n'est ni en deuxième forme normale ni dans une forme normale supérieure. Il en est de même pour la table MOT_CLÉ, car il existe une liaison de type complexe-complexe entre les livres et les mots-clés. En effet, la «Pondération» est typiquement un attribut de liaison, mais la «Désignation» ne dépend pas fonctionnellement de la totalité de la clé définie par concaténation des identifiants du mot-clé et du livre. Par conséquent, en vertu des principes de normalisation, nous devons disposer de cinq tables dans notre système de gestion des livres. Pour ce faire, outre les tables AUTEUR et MOT_CLÉ qui traduisent les liens dans le modèle, nous devons ajouter des tables distinctes pour stocker les attributs des auteurs et des mots-clés.

Les bases de données relationnelles supportent les objets structurés de manière imparfaite

L'utilisateur aurait de la peine à comprendre l'éclatement des informations sur un livre en plusieurs tables avec des inconvénients qui en résultent, car à son point de vue, la structure des propriétés d'un ouvrage devrait être définie dans une seule table. Les langages relationnels de requête et de manipulation de données ont précisément pour but de faciliter la gestion des données de notre bibliothèque au travers de simples opérateurs. Des faiblesses apparaissent aussi au niveau de la performance : le système de bases de données doit parcourir plusieurs tables et effectuer des jointures onéreuses en temps de calcul pour trouver un livre particulier. C'est pour remédier à ces faiblesses que des extensions du modèle relationnel ont été proposées.

Les clés de substitution servent à structurer les données et sont gérées par le système de bases de données

Une première extension de la technologie des bases de données relationnelles consiste à *déclarer explicitement les propriétés structurelles dans le système de base de données au moyen de clés de substitution. La valeur d'une clé de substitution est générée par le système et ne peut pas être modifiée (valeur invariante)*. Elle identifie de manière unique chaque tuple dans la base de données. Grâce à leur propriété d'invariance, les clés de substitution peuvent servir à définir des liens contrôlés par le système à différentes places dans une base de données relationnelle. Elles permettent aussi de définir des contraintes d'intégrité référentielle, ainsi que des structures de généralisation et d'agrégation.

Retournons maintenant à la table LIVRE représentée en figure 6-5, et définissons l'identifiant du livre L# comme clé de substitution. L'identifiant L# se retrouve dans les tables dépendantes AUTEUR et MOT_CLÉ pour établir la liaison PART_OF(LIVRE) (voir la règle d'agrégation 7 à la section 2.3.3). Cette référence a deux objectifs : premièrement elle indique explicitement au système de bases de données la propriété structurelle des données sur les livres, les auteurs et les mots-clés. Deuxièmement elle lui permet de tirer profit de cette structure pour traiter les requêtes d'interrogation, tout ceci pour autant que des extensions conséquentes soient implémentées dans le langage relationnel de requête et de manipulation de données. L'exemple est donné par l'opérateur de jointure hiérarchique implicite dans la clause FROM, qui connecte à la table LIVRE les sous-tables dépendantes AUTEUR et MOT_CLÉ. Les prédicats de jointure dans la clause WHERE deviennent alors superflus, car le système de bases de données les connaît déjà grâce à la définition explicite de la structure PART_OF.

Les clés de substitution permettent de créer les structures PART_OF

En outre, si la structure PART_OF, ou de manière analogue, la structure IS_A (voir 2.2.3) est communiquée explicitement au système de bases de données, les structures de stockage pourront être implémentées de manière efficace. Notre exemple se développe alors comme suit : tout en conservant notre vue logique composée de trois tables LIVRE, AUTEUR et MOT_CLÉ, nous allons stocker physiquement les données sur les livres comme des objets structurés³. De cette manière, l'extraction d'un livre s'exécute en un seul accès à la base de données. La vue classique des tables est préservée, car nous pouvons continuer à interroger les tables de la structure d'agrégation comme auparavant.

Les structures PART_OF et IS_A sont définies explicitement dans le système de bases de données

Une autre technique de gestion des informations structurées consiste à *abandonner la première forme normale*⁴ et admettre les tables mêmes comme attributs. Pour illustrer cette approche, nous modifions l'exemple de notre bibliothèque en regroupant toutes les informations sur les livres, les auteurs et les mots-clés en une seule

Utilité des tables imbriquées

³ Dans la littérature scientifique, ils s'appellent aussi «objets complexes».

⁴ Le modèle NF² (NF² = Non First Normal Form) autorise des tables imbriquées.

table représentée en figure 6-6. C'est également une approche relationnelle-objet, car chaque livre est géré comme un objet dans une table unique OBJET_LIVRE. Un *système de bases de données relationnel-objet* (*object-relational database system*, en anglais) contient explicitement les propriétés structurelles et comporte des opérateurs sur les objets et les sous-objets.

Figure 6-6
Table OBJET_LIVRE
avec des attributs
de type Relation

OBJET_LIVRE							
L#	Titre	Éditeur	Auteur		Mot-clé		Pondération
			A#	Nom	C#	Description	
L1	Bases de...	Springer	A1	Meier	C1	Base de données	80
					C2	Modèle relationnel	20
L2	Infographie...	McGraw Hill	A1	Meier	C3	Infographie	50
			A2	Loacker	C4	Géométrie...	50

Les systèmes de bases de données relationnels-objet doivent supporter le paradigme de l'orientation objet

Lorsqu'un système de bases de données prend en charge des types d'objets structurés comme ceux dans les figures 6-5 et 6-6, il s'agit d'un système relationnel-objet au point de vue structurel. Outre l'identification des objets, la description des structures et la disponibilité des opérateurs génériques (des méthodes telles que la jointure implicite, etc.), un système de bases de données entièrement relationnel-objet offre à l'utilisateur la possibilité de définir de nouveaux types d'objets (classes) et de nouvelles méthodes. L'utilisateur doit pouvoir créer lui-même les méthodes nécessaires à un type d'objets spécifique. Il doit pouvoir tirer profit du mécanisme de «l'héritage» en réutilisant les concepts existants au lieu de réinventer les types d'objets et les méthodes.

Opérateurs génériques pour l'agrégation et la généralisation

Dans un système de bases de données relationnel-objet, les objets structurés sont des unités de traitement auxquelles on applique des opérateurs génériques appropriés. La création de classes à partir des structures PART_OF et IS_A est possible et prise en charge par des méthodes de stockage, d'interrogation et de manipulation de données.

Système de bases de données relationnel-objet

Un système de gestion de bases de données relationnel-objet (SGBDRO) comporte les caractéristiques majeures suivantes :

- Il permet la *définition de types d'objets* (appelés classes en programmation orientée objet) qui peuvent être eux-mêmes formés d'autres types d'objets.
- Chaque objet de la base de données est *structuré et identifié au moyen de clés de substitution*.
- Un système de gestion de bases de données relationnel-objet prend en charge des *opérateurs génériques* (méthodes) qui s'appliquent aux objets et aux sous-objets. La représentation interne des objets reste invisible à l'extérieur (encapsulation des données).
- Les propriétés des objets sont transmises par héritage. Le *mécanisme d'héritage* s'applique aussi bien à la structure qu'aux opérateurs associés.

Un système de bases de données relationnel-objet doit se conformer à la *norme SQL étendu* (connue sous le nom de SQL:2003) qui requiert les extensions orientées objet suivantes : identification d'objets (clés de substitution), types de données prédéfinis pour les ensembles, les listes et les champs, types de données abstraits généraux avec encapsulation, types paramétrisables, hiérarchies de types et de tables avec héritage multiple et fonctions définies par l'utilisateur (méthodes).

6.5 Les bases de données multidimensionnelles

Les bases de données et les applicatifs opérationnels sont axés sur un domaine fonctionnel bien défini. Les transactions ont pour but de produire rapidement et avec précision des données qui font tourner les affaires de l'entreprise. Nous parlons de *traitement transactionnel en ligne* (*Online Transaction Processing* ou OLTP, en anglais) pour

Principales propriétés d'un système de bases de données relationnel-objet

La nouvelle norme SQL contient des extensions orientées objet

Que signifie l'acronyme OLTP ?

désigner la mission principale d'un système applicatif dit de production.

Quel est l'intérêt
d'OLAP ?

La mise à jour quotidienne des données opérationnelles écrase des informations importantes à la prise de décision. En outre, les bases de données de type opérationnel sont essentiellement conçues et optimisées pour prendre en charge les affaires courantes de l'entreprise, et non pas pour l'analyse décisionnelle des informations. C'est pourquoi, au cours des dernières années, parallèlement aux systèmes transactionnels, émergent de nouvelles bases de données et applicatifs indépendants, destinés à l'analyse de données et à l'aide à la décision. Ce sont des systèmes dits de *traitement analytique en ligne* (*Online Analytical Processing* ou OLAP, en anglais).

Le noyau d'une
base de données
multi-
dimensionnelle

Le cœur d'un système OLAP est sa *base de données multidimensionnelle* (*multi-dimensional database*, en anglais) hébergeant des informations décisionnelles qui se prêtent à des analyses en plusieurs dimensions (*cube de données ; data cube*, en anglais). Une telle base de données est volumineuse car elle accumule des valeurs de données décisionnelles au cours du temps. Par exemple, à des fins d'analyse une base de données multidimensionnelle peut contenir les chiffres d'affaires trimestriels d'une entreprise, par produits et par territoires de vente.

Les dimensions
désignent les axes
d'un cube de
données

La figure 6-7 montre un exemple d'analyse multidimensionnelle et illustre la conception d'une base de données multidimensionnelle. Nous nous intéressons ici à trois perspectives d'analyse : le produit, la région et le temps. La notion de *dimension* (*dimension*, en anglais) définit les axes d'un cube multidimensionnel. La conception des dimensions est cruciale, car elles déterminent les possibilités d'analyse d'après les axes. L'ordonnancement des dimensions ne joue aucun rôle, ce sont les utilisateurs qui choisissent chacun leur perspective d'analyse. Ainsi, un directeur de la fabrication se concentre sur la dimension produit alors qu'un vendeur s'intéresse au chiffre d'affaires réalisé dans sa région.

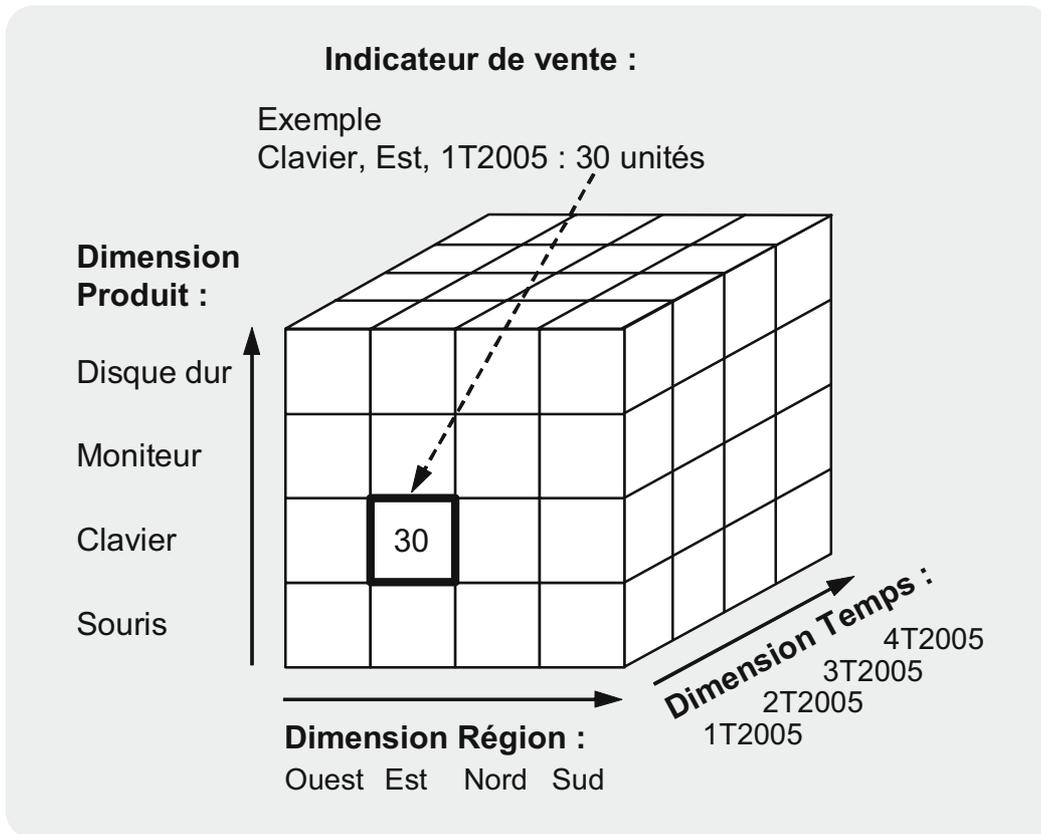


Figure 6-7
Un cube
multidimensionnel
avec plusieurs
perspectives
d'analyse

Une dimension peut elle-même contenir une structure : la dimension produit peut comporter des groupes de produits ; dans la dimension temps, chaque année peut être structurée en trimestres, en jours, en semaines et en mois. Chaque dimension comporte par conséquent des *niveaux d'agrégation* en fonction des besoins d'analyse du cube multidimensionnel.

*Structure
hiérarchique des
dimensions*

Du point de vue logique, dans une base de données multidimensionnelle ou un cube de données, nous devons spécifier non seulement les dimensions mais aussi des indicateurs⁵. Par *indicateur* (*indicator*, en anglais), nous entendons un indice ou un paramètre qui aide à la prise de décision. Les indicateurs traduisent les propriétés quantitatives et qualitatives de la marche des affaires d'une entreprise. Outre les indicateurs financiers, il existe d'importants indicateurs du marché et des ventes, de la base clientèle et de son comportement, des processus d'affaires, du potentiel d'innovation ou

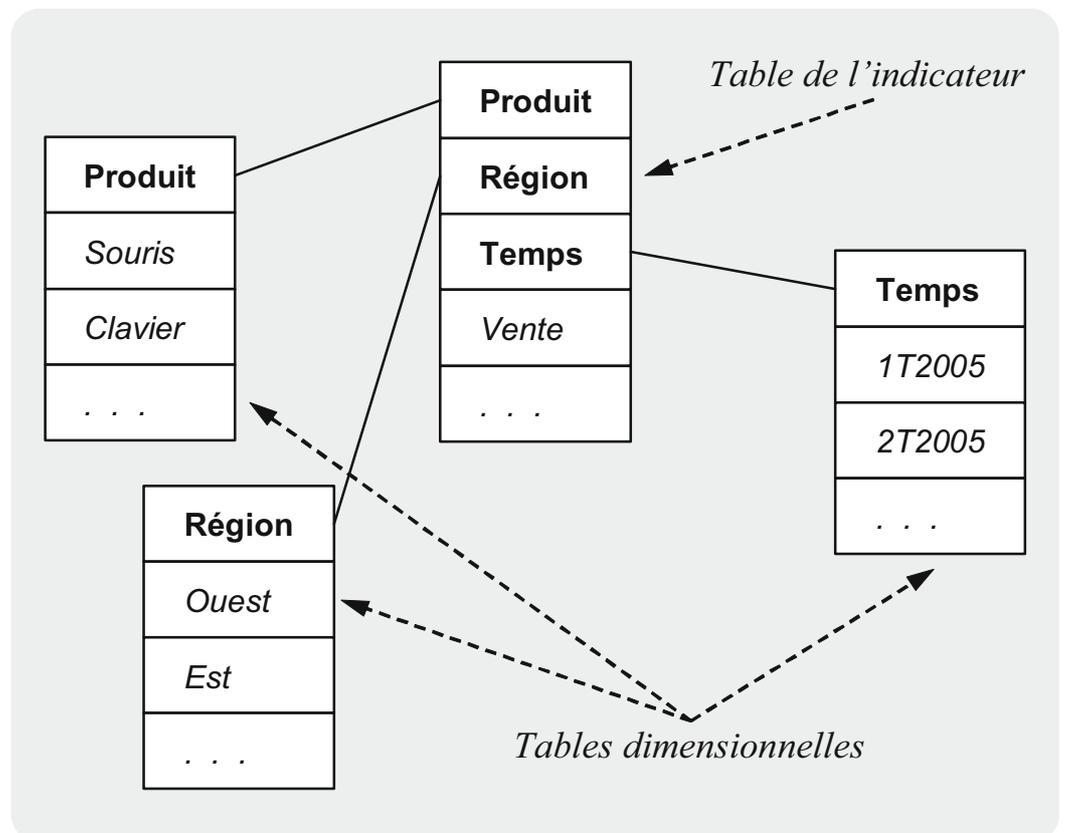
*Les valeurs des
indicateurs
constituent une
aide à la décision*

⁵ Dans la littérature, les auteurs parlent souvent de faits à la place d'indicateurs. Voir aussi la section 6.7 qui traite des faits et des règles dans les bases de connaissances.

du savoir-faire du personnel. Les dimensions et les indicateurs constituent les éléments de base d'un système destiné au soutien à la prise de décision, à la production des rapports de gestion internes et externes, et à la mesure des performances assistée par ordinateur.

Un *schéma en étoile* (*star schema*, en anglais) se caractérise par la classification des données en deux groupes : les données de l'indicateur et les données des dimensions. Ces groupes sont représentés sous forme tabulaire dans la figure 6-8. La table de l'indicateur est placée au centre, entourée de tables dimensionnelles, une par dimension. Ainsi, les tables dimensionnelles sont connectées à la table de l'indicateur en forme d'étoile.

Figure 6-8
Schéma en étoile
d'une base de
données multi-
dimensionnelle



Extensions du schéma en étoile

Lorsqu'une dimension possède une structure, la table correspondante se connecte à d'autres tables dimensionnelles, donnant lieu à un *schéma en flocons* (*snowflake schema*, en anglais) qui reflète les niveaux d'agrégation de chaque dimension. Par exemple, dans la figure 6-8, nous pouvons concevoir une nouvelle table dimensionnelle qui contient les jours de janvier à mars 2005 et qui se rattache au premier trimestre de l'an 2005 dans la table de la

dimension temps. Si nous désirons faire des analyses mensuelles, il faudra définir une nouvelle table dimensionnelle associée à la dimension mois et la connecter à la table de la dimension jour, etc.

La réalisation d'une base de données multidimensionnelle peut reposer sur le modèle relationnel classique, mais cette option engendre de sérieux inconvénients. La figure 6-9 nous montre comment implémenter la table de l'indicateur et les tables dimensionnelles dans un schéma en étoile.

L'approche relationnelle convient mal à la mise en œuvre

Déterminer le chiffre d'affaires de Morel, directeur commercial, résultant de la vente des produits d'Apple au premier trimestre 2005 :

```
SELECT SUM(Recettes)
FROM D_PRODUIT D1, D_REGION D2, D_TEMPS D3, F_VENTE F
WHERE D1.P# = F.P# AND
      D2.R# = F.R# AND
      D3.T# = F.T# AND
      D1.Fournisseur = 'Apple' AND
      D2.Directeur_Commercial = 'Morel' AND
      D3.Annee = 2005 AND
      D3.Trimestre = 1
```

D_PRODUIT

P#	Désignation	Fournisseur
P2	Clavier	Apple

D_RÉGION

R#	Nom	Directeur_Commercial
R2	Est	Morel

D_TEMPS

T#	Année	Trimestre
T1	2005	1

F_VENTE

P#	R#	T#	Quantité	Recettes
P2	R2	T1	30	160'000

La table de l'indicateur est représentée par la relation F_VENTE (Figure 6-9) dotée d'une clé multidimensionnelle. Les parties de cette clé composée sont les clés des tables dimensionnelles D_PRODUIT, D_REGION et D_TEMPS. Pour déterminer par exemple le chiffre d'affaires qu'a réalisé Morel, directeur commercial, par la vente des produits d'Apple au premier trimestre de l'an 2005, nous devons formuler une coûteuse jointure de la table de l'indicateur et des tables dimensionnelles en présence (voir l'instruction SQL à la figure 6-9).

*Figure 6-9
Implémentation
d'un schéma en
étoile basé sur le
modèle relationnel*

*Analyse du cube
multi-
dimensionnelle*

Inconvénients des bases de données relationnelles dans la gestion du cube multidimensionnelle

La puissance d'un système de bases de données relationnelles atteint ses limites face à la grande taille des bases de données multidimensionnelles. En outre, pour un schéma en étoile, la formulation des requêtes est une tâche ardue et sujette à l'erreur. La liste des faiblesses s'allonge si l'on travaille avec le modèle relationnel classique et le SQL traditionnel : pour définir des niveaux d'agrégation, le schéma en étoile doit évoluer vers un schéma en flocons et les tables physiques qui en résultent dégradent le temps de réponse. Lorsque l'utilisateur d'une base de données multidimensionnelle désire extraire des informations détaillées (zoom en profondeur ou *forage vers le bas* ; *drill-down*, en anglais) ou explorer d'autres niveaux d'agrégation (*forage vers le haut* ; *roll-up*, en anglais), le langage SQL classique ne pourra plus répondre à ses besoins d'analyse. En outre, la coupe ou le pivotage des parties d'un cube multidimensionnel est une opération très fréquente dans l'analyse de données et requiert des logiciels, voire des matériels, spécifiques.

De la base de données à l'entrepôt de données

Nombre de fournisseurs de produits de bases de données ont compris ces faiblesses, et proposent dans leur assortiment de logiciels des outils adaptés aux *entrepôts de données* (*data warehouse*, en anglais) ou à la *fouille de données* (*data mining*, en anglais). L'entrepôt de données désigne précisément un environnement logiciel conçu pour exploiter une base de données multidimensionnelle, incluant des outils pour définir les données (maintenance des tables dimensionnelles) d'une part, et pour intégrer différents ensembles de données d'autre part. Au niveau du langage, des extensions ont été apportées à SQL pour simplifier la formulation des opérations sur un cube multidimensionnel, y compris les agrégations.

Système de bases de données multidimensionnelles

Les critères d'un système de bases de données multidimensionnelles (entrepôt de données)

Un système de gestion de bases de données multidimensionnelles (SGBDM), communément appelé entrepôt de données, prend en charge un cube multidimensionnel en offrant les fonctionnalités suivantes :

- au niveau conceptuel, il est possible de définir des tables dimensionnelles avec un nombre quelconque de niveaux *d'agrégation* ;
- le langage de requête et d'analyse dispose des fonctions de forage vers le bas (*drill-down*, en anglais) et vers le haut (*roll-up*, en anglais) ;
- il est facile de sélectionner une tranche dans un cube multidimensionnel (*slicing*, en anglais) et de *spécifier les valeurs d'une ou de plusieurs dimensions* pour extraire un sous-cube (*dicing*, *rotation*, en anglais) ;
- les tables d'indicateurs sont implémentées sans désigner de dimension ; en d'autres termes, les données sont stockées dans des *structures de données multidimensionnelles*.

L'intégration de multiples sources de données internes et externes dans un entrepôt de données permet à l'entreprise d'obtenir un ensemble de données cohérentes qu'elle pourra analyser sous différents angles. En général, à l'exception des mises à jour périodiques, on accède à une base de données multidimensionnelle en mode lecture seule à des fins d'analyse. La dimension temps est un composant inhérent à un entrepôt de données, elle offre une perspective importante dans les analyses statistiques. Lorsque nous utilisons des outils d'exploration de gisements de données (*data mining*, en anglais), destinés par exemple à la classification, la prévision et l'acquisition de connaissances, nous passons du domaine des bases de données multidimensionnelles à celui des bases de connaissances.

De l'entrepôt de données à l'extraction de la connaissance

6.6 Les bases de données floues

Dans le domaine des bases de données, relationnelles en particulier, les valeurs d'attributs sont dépourvues d'ambiguïté et les requêtes d'interrogation produisent des résultats précis. Les bases de données relationnelles se caractérisent en effet par les propriétés suivantes :

Les valeurs dans une base de données relationnelle sont précises

- Les valeurs d'attributs dans une base de données sont *précises*, c'est-à-dire non ambiguës. Nous connaissons déjà la règle de la première forme normale qui exige que ces valeurs soient atomiques et qu'elles appartiennent à un domaine de valeurs clairement définies. Il est exclu de donner une valeur imprécise à un attribut en définissant par exemple que le délai de livraison de tel fournisseur est de «2 ou 3 ou 4 jours». Les valeurs vagues ne sont pas admises non plus ; ainsi, nous ne pouvons pas spécifier un délai de livraison «d'environ 3 jours».
- Les valeurs d'attributs stockées dans une base de données relationnelle sont *certaines*, c'est-à-dire que chaque valeur individuelle est soit connue soit inconnue, à l'exception de la valeur nulle assignée à un attribut dont la valeur n'est pas ou pas encore connue. En outre, les systèmes de bases de données ne nous offrent pas la possibilité de concevoir des modèles en y introduisant l'incertitude stochastique. En d'autres termes, les distributions de probabilités des valeurs d'attributs sont exclues ; il reste difficile d'affirmer si une valeur d'attribut particulière correspond à une valeur réelle ou pas.
- Les requêtes d'interrogation sont *bien définies*. Elles présentent toutes un caractère dichotomique : toute valeur utilisée comme critère d'interrogation correspond ou ne correspond pas aux valeurs de l'attribut associé dans la base de données. Le cas où la valeur du critère d'interrogation est égale à la valeur stockée «plus ou moins» n'existe pas.

*Bases de données
en combinaison
avec la logique
floue*

Ces dernières années, les connaissances dans le domaine de la logique floue (*fuzzy logic*, en anglais) ont été appliquées aux modèles et aux bases de données. La plupart des travaux sont de nature théorique ; cependant, des équipes de recherche ont montré leur portée pratique par la mise en œuvre des modèles et des systèmes de bases de données floues.

*Bases de données
et les faits flous*

L'introduction des *faits incomplets, vagues ou imprécis* dans les modèles de données ouvre un vaste champ d'applications. La logique floue a rendu possible des extensions tant dans le modèle entité-association que dans le modèle relationnel. Les formes normales

classiques dans la théorie des bases de données s'étendent aux dépendances fonctionnelles floues.

Des projets de recherche portent également sur *l'extension des langages d'interrogation relationnels à la logique floue*. Ainsi, des langages d'interrogation flous ont été développés et partiellement implantés dans des prototypes. À titre d'illustration, nous présentons ci-après un langage de classification floue, fCQL (fuzzy Classification Query Language), appliqué à un problème concret de classification.

Extension des langages d'interrogation relationnels

Dans l'analyse et l'évaluation des partenaires commerciaux d'une entreprise, il est intéressant de les regrouper par classes selon leurs caractéristiques communes. Ainsi par exemple nous définissons un ensemble regroupant des «partenaires ayant des problèmes de qualité à la livraison» et un deuxième ensemble de «partenaires avec qui nous renforcerons les liens commerciaux». Ce regroupement en classes signifie une réduction de la complexité. Il permet au chargé des relations de l'entreprise de mieux analyser les liens commerciaux avec ses partenaires et de les entretenir de manière mieux ciblée.

Classification des partenaires commerciaux

Dans une classification floue, chaque partenaire commercial appartient à une ou plusieurs classes selon les valeurs de leurs attributs ; on parle de segmentation floue des partenaires. Par exemple, un partenaire peut appartenir simultanément à la classe des «partenaires ayant des problèmes de qualité» avec une valeur d'appartenance de 0.3 et à la classe des «partenaires avec qui nous renforcerons les liens commerciaux» avec une valeur de 0.7. La classification floue permet donc une interprétation différenciée de l'appartenance à des classes.

Avantages de la classification floue

La figure 6-10 montre l'espace de classification pour évaluer les partenaires commerciaux sur la base de deux attributs dans notre exemple simple : le retard de livraison et la qualité. Le retard de livraison s'exprime en nombre de jours entre la date de livraison promise et le jour de livraison effective. Pour juger de la qualité de la livraison, nous utilisons les prédicats «excellente», «moyenne», «suffisante» et «médiocre». Ces domaines de valeurs, retard de livraison et qualité de la livraison, sont partitionnés par l'utilisateur en

Différenciation verbale de la qualité et du retard de livraison

quatre classes d'équivalence C1, C2, C3 et C4. L'importance de chaque classe se reflète à travers les mesures d'actions qu'elle recommande ; ainsi par exemple, le programme d'actions «Réexaminer les relations d'affaires» appartient à la classe C4.

Figure 6-10
Espace de
classification défini
par les attributs
Qualité et Retard
de livraison

D(Retard de livraison)						
10	C2 Réclamer pour cause de retard de livraison	C4 Réexaminer les relations d'affaires				
9						
8						
7						
6						
5	C1 Développer les relations d'affaires	C3 Discuter du problème de qualité				
4						
3						
2						
1						
		excellente	moyenne	suffisante	médiocre	D(Qualité)

*Emploi des
variables
linguistiques*

Les classes sont caractérisées par les constructions du langage (variables linguistiques). Nous associons à l'attribut «retard de livraison» les termes «acceptable» pour {1,2,3,4,5} et «inacceptable» pour {6,7,8,9,10}. Si l'attribut «qualité» reçoit la valeur «excellente» ou «moyenne», nous lui associons la caractéristique «bonne», «mauvaise» dans le cas contraire.

*Fonction
d'appartenance*

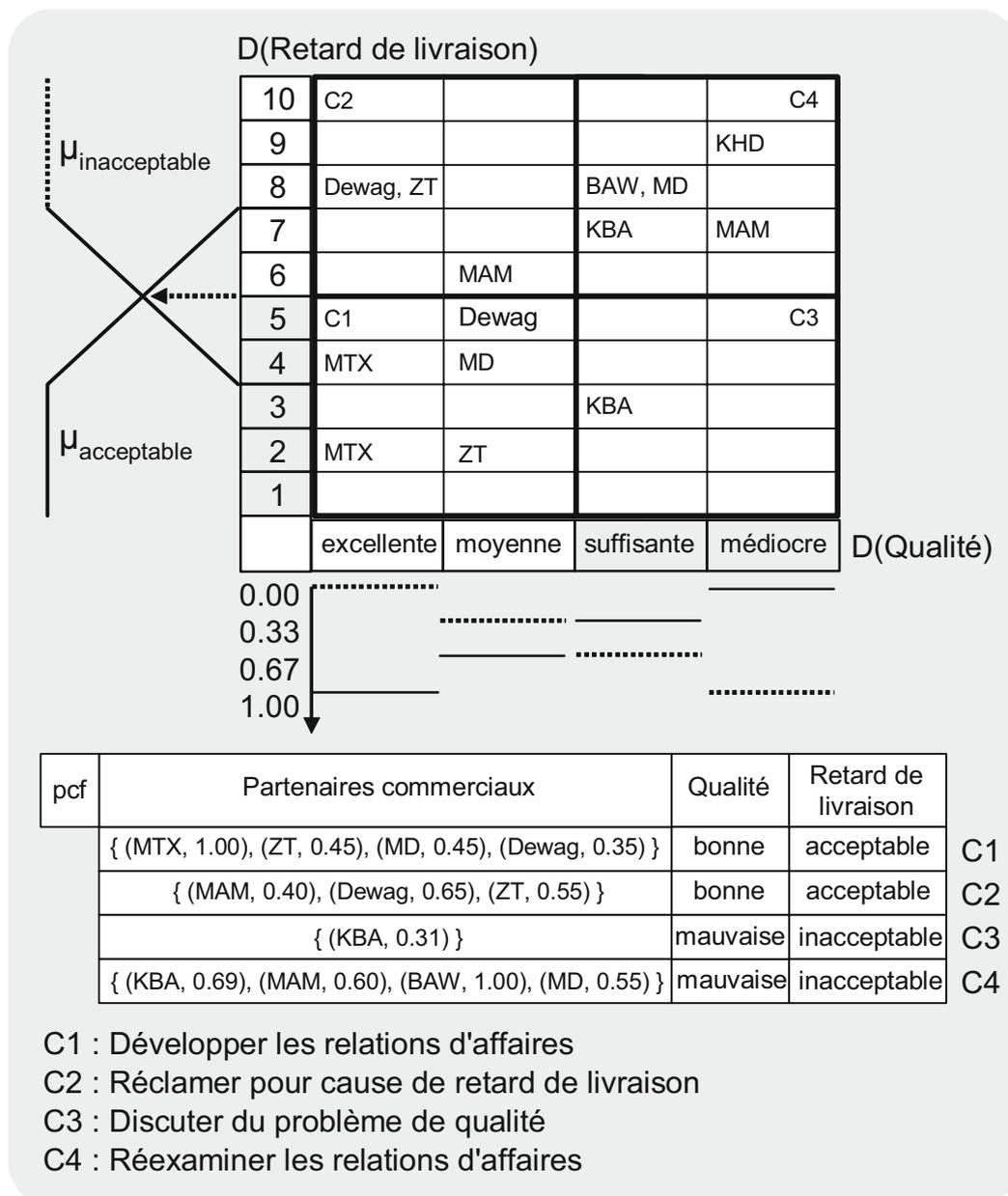
Dans la figure 6-11, les partenaires commerciaux BAW, Dewag, KBA, MD, MTX, MAM et ZT font l'objet d'une classification floue, basée sur une fonction d'appartenance μ qui prend les valeurs comprises entre 0 et 1. Un partenaire commercial peut se trouver dans plusieurs classes à la fois en fonction du retard de livraison et de la qualité de sa livraison.

*Emploi des termes
vagues*

Les fonctions d'appartenance $\mu_{\text{acceptable}}$ et $\mu_{\text{inacceptable}}$, associées aux termes vagues «acceptable» et «inacceptable», définissent un partitionnement flou du domaine des valeurs de l'attribut «retard de livraison». Dans la figure 6-11, un retard de livraison avec un degré

d'appartenance de 0.5 est à la fois acceptable et inacceptable ; en d'autres termes, la valeur d'appartenance du retard de 5 jours aux ensembles flous $\mu_{\text{acceptable}}$ et $\mu_{\text{inacceptable}}$ est égale à 0.5. Par conséquent, contrairement à la classification classique, il n'est pas possible de juger un retard de livraison de 5 jours comme acceptable ou inacceptable de manière nette et précise. Le domaine de l'attribut «qualité» se subdivise d'après les termes «bonne» et «mauvaise». Il en résulte des classes entre lesquelles la transition est graduelle.

Figure 6-11
Segmentation floue
des partenaires
commerciaux



Appartenance des fournisseurs à différentes classes

Dans la table pcf (Partenaires commerciaux flous) à la figure 6-11, les ensembles flous dans la colonne «Partenaires commerciaux» sont le résultat d'une décomposition floue. Ainsi par exemple, le fournisseur Dewag appartient à la classe «Développer les relations d'affaires» (C1) avec un degré d'appartenance de 0.35 et à la classe «Réclamer pour cause de retard de livraison» avec une valeur de 0.65. Comme le principal problème du partenaire commercial Dewag se situe plutôt au niveau du délai de livraison qu'à celui de la qualité, des actions appropriées ont été pertinemment recommandées grâce à la segmentation floue. Celle-ci permet aussi d'envisager un programme d'actions ciblées pour développer des relations d'affaires avec les partenaires commerciaux en cas de besoin.

Le langage de classification flou fcQL

Les requêtes de classification en langage fcQL (fuzzy Classification Query Language) utilisent des variables linguistiques dans un contexte vague. Ceci a l'avantage que l'utilisateur doit connaître non pas un contexte et une valeur cible précise, mais seulement le nom de colonne associé à l'attribut d'un objet et la table ou la vue qui contient les valeurs de l'attribut considéré. Pour considérer une classe particulière, l'utilisateur peut soit la déclarer soit indiquer les valeurs linguistiques de la classe en question. En d'autres termes, les requêtes de classification opèrent sur les dénominations verbales des attributs et des classes :

CLASSIFY	Objet
FROM	Table
WITH	Critère de classification

fcQL s'appuie sur SQL

Le langage fcQL s'appuie sur SQL. Les noms de colonnes de l'objet à classifier remplacent la liste d'attributs de projection dans la clause SELECT. La clause WITH spécifie un critère de classification alors qu'en SQL la clause WHERE contient un critère de sélection.

Exemples en fcQL

Considérons par exemple la table SUPPLIER, un ensemble flou de fournisseurs caractérisés par deux attributs, «retard de livraison» et «qualité de la livraison». La requête fcQL :

CLASSIFY	Fournisseur
FROM	SUPPLIER

produit une classification de tous les fournisseurs enregistrés dans la table SUPPLIER. Pour interroger une classe particulière, nous formulons par exemple la requête :

```
CLASSIFY    Fournisseur
FROM        SUPPLIER
WITH        CLASS IS Problème_Livraison
```

Au lieu d'utiliser la définition d'une classe, on peut sélectionner un ensemble objet déterminé en spécifiant les valeurs linguistiques des classes d'équivalence. Considérons la requête :

```
CLASSIFY    Fournisseur
FROM        SUPPLIER
WITH        Retard_Livraison IS acceptable
            AND Qualité IS bonne
```

Elle contient l'identificateur de l'objet à classier (Fournisseur), le nom de la table de base (SUPPLIER), les noms d'attributs définis dans le critère (Retard_Livraison, Qualité) et les noms de classes d'équivalence prédéfinies (acceptable, bonne).

L'exemple étudié dans cette section nous permet de caractériser les bases de données floues comme suit :

Système de bases de données floues

Un système de gestion de bases de données floues (SGBDF) est un système de bases de données doté des caractéristiques suivantes :

- Le modèle de données est relationnel flou, c'est-à-dire qu'il admet des *valeurs d'attributs imprécises, vagues et incertaines*.
- Les dépendances entre les attributs s'expriment par des *formes normales floues*.
- La logique floue permet de construire une théorie du calcul relationnel flou et de l'algèbre relationnelle floue comme extension du calcul relationnel et de l'algèbre relationnelle.

*Concepts
fondamentaux d'un
système de bases
de données floues*

- Un langage d'interrogation étendu, utilisant les variables linguistiques, permet de *formuler des requêtes floues*.

Nécessité de la collaboration inter-disciplinaire

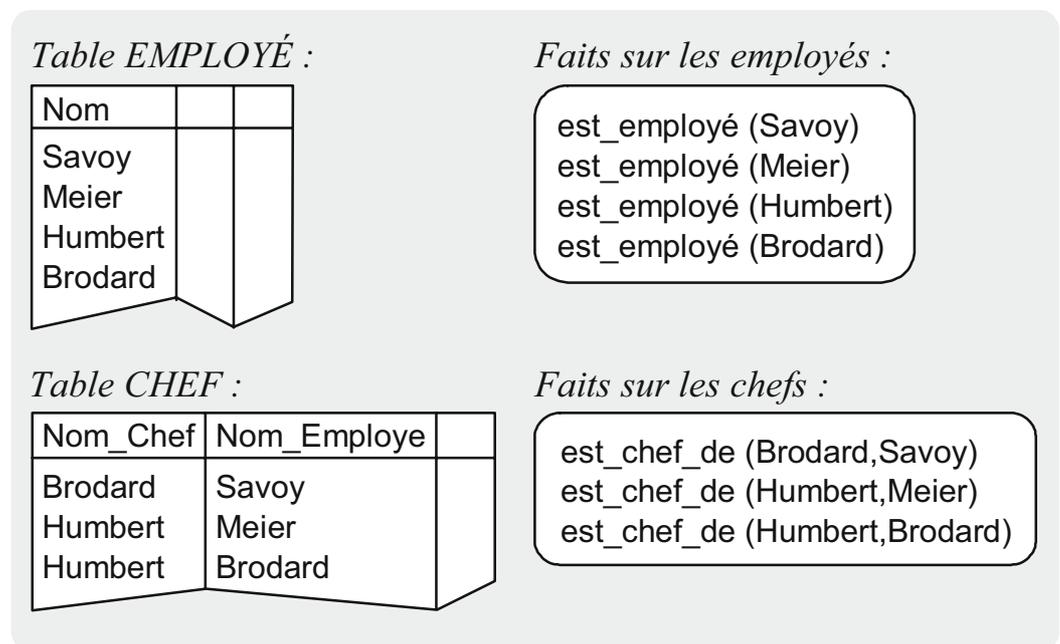
Depuis des années, des informaticiens menaient séparément leur recherche dans le domaine de la logique floue et des systèmes de bases de données relationnelles. Les résultats de leurs travaux étaient publiés surtout dans la littérature sur la logique floue, mais peu diffusés et reconnus dans le domaine des bases de données. Nous espérons que ces deux domaines de recherche se rapprochent dans le futur et que les chefs de file dans la technologie des bases de données reconnaissent le potentiel des bases de données floues et des langages de requête flous.

6.7 Les bases de connaissances

Bases de données avec des faits et des règles

Les *bases de connaissances* (*knowledge databases* ou *deductive databases*, en anglais) permettent de gérer non seulement les données - appelées *faits* - mais aussi les *règles* qui permettent de déduire de nouvelles entrées ou de nouveaux faits dans une table.

Figure 6-12
Tables versus faits



Représentation des faits par les tables

Considérons, à la figure 6-12, la table EMPLOYÉ qui, pour simplifier, ne contient que la colonne des noms d'employés. En rapport avec cette table, nous définissons les faits ou les prédicats sur les employés. Nous considérons ces prédicats comme étant des faits

dont la valeur de vérité est implicitement VRAIE (TRUE). Ainsi, il est vrai que Monsieur Humbert est un employé, et cette réalité s'exprime par le fait «est_employé (Humbert)». Pour connaître les supérieurs directs des employés, nous créons une nouvelle table CHEF dont chaque tuple consiste en une paire de données : le nom du supérieur direct et celui de l'employé subordonné. En rapport avec cette table, nous définissons les faits «est_chef_de (A,B)» pour exprimer la réalité que «A est le chef direct de B».

La hiérarchie du commandement est schématisée par un arbre dans la figure 6-13. Pour savoir qui est le chef direct de l'employé Meier, une requête SQL interroge la table CHEF et trouve Humbert, le supérieur recherché. Le même résultat s'obtient en exécutant une requête dans un langage de programmation logique (dans le genre de Prolog).

Analyse des faits

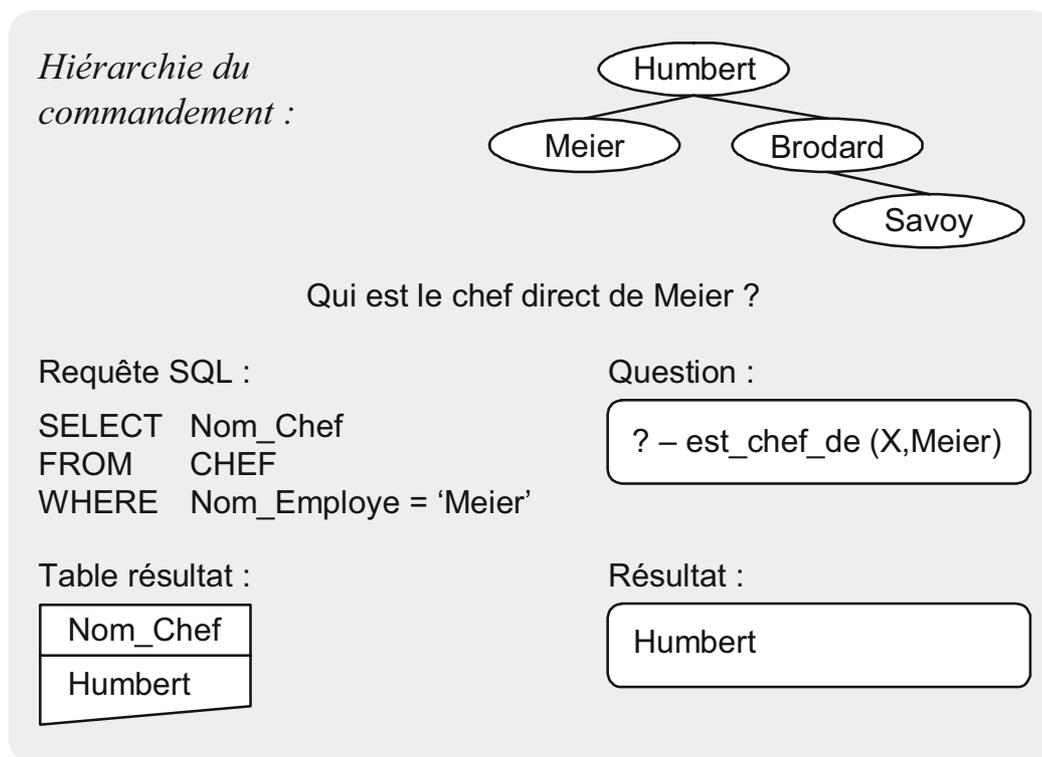


Figure 6-13
Analyse des tables
et des faits

Outre les faits, nous définissons des règles pour déduire le contenu inconnu des tables. Dans le contexte du modèle relationnel, nous parlons alors de tables dérivées (*derived relation* ou *deduced relation*, en anglais). La figure 6-14 donne l'exemple simple d'une table dérivée et de la règle de déduction correspondante qui permet de connaître le supérieur du chef direct de chaque employé. Dans les

Comment
construire des
règles dans le
modèle
relationnel ?

grandes entreprises ou les sociétés avec des succursales éloignées, cette règle s'avère intéressante, notamment lorsque le chef direct d'un employé est absent et qu'on doit s'adresser au supérieur du chef en question par messagerie électronique.

Figure 6-14
Dédiction de
nouvelles
connaissances

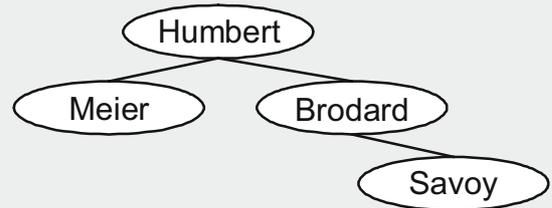
Table dérivée :

```
CREATE VIEW SUPERIEUR AS
SELECT X.Nom_Chef, Y.Nom_Employe
FROM CHEF X, CHEF Y
WHERE X.Nom_Employe = Y.Nom_Chef
```

Règle :

```
est_supérieur_de (X,Y)
IF est_chef_de (X,Z) AND
est_chef_de (Z,Y)
```

*Hiérarchie du
commandement :*



Afficher les couples

«Employé / Supérieur à deux échelons plus hauts»

Requête SQL :

```
SELECT *
FROM SUPERIEUR
```

Question :

```
? – est_supérieur_de (X,Y)
```

Table résultat :

Nom_Chef	Nom_Employé
Humbert	Savoy

Résultat :

```
Humbert Savoy
```

Création de
tables dérivées
par des vues

La définition d'une table dérivée correspond à la création d'une vue. Dans notre exemple, une vue appelée SUPÉRIEUR est créée pour déterminer le supérieur du chef direct de chaque employé. Elle résulte de la jointure de la table CHEF avec elle-même. Nous pouvons construire une règle de dérivation équivalente à la vue SUPÉRIEUR. La règle «est_supérieur_de (X,Y)» résulte du fait qu'il existe un Z tel que X est chef direct de Z et que Z est lui-même chef direct de Y. Par conséquent, X est le supérieur de Y à deux échelons plus hauts, car Z se situe hiérarchiquement entre X et Y.

Une base de données qui comporte à la fois des faits et des règles constitue une *base de méthodes ou de connaissances*, car non seulement elle contient des faits évidents tels que «Humbert est un employé» ou «Humbert est le chef direct de Meier et Brodard», mais elle permet aussi des déductions telles que «Humbert est le chef de Savoy à deux échelons supérieurs».

Passage aux bases de connaissances

La vue SUPÉRIEUR, définie dans la figure 6-14, sert à déterminer le supérieur du chef direct de chaque employé. Dans notre exemple, la requête SQL qui interroge cette vue produit une table résultat. Celle-ci contient le fait qu'il existe un seul lien hiérarchique reliant l'employé Savoy à son supérieur Humbert à deux échelons plus hauts. L'application de la règle de dérivation correspondante «est_supérieur_de» conduit au même résultat.

Analyse logique en SQL

Une base de données déductive contenant des faits et des règles supporte en outre le *principe de récursion* qui permet de faire un nombre quelconque de déductions correctes fondées sur les règles contenues dans la base en question. Chaque prédicat dont la valeur de vérité est VRAIE engendre de nouveaux prédicats.

Les bases de données deductives permettent l'analyse récursive

Le principe de récursion s'applique *soit aux objets de la base de données soit aux règles de dérivation*. Par objets définis récursivement, nous entendons des structures qui comportent elles-mêmes d'autres structures, et qui peuvent être représentées comme des structures d'objets hiérarchiques ou en réseau pareillement aux deux concepts d'abstraction, la généralisation et l'agrégation. Les prédicats peuvent aussi être calculés de manière récursive. Ainsi, dans l'exemple de la hiérarchie du commandement, à partir des faits «est_employé_de» et «est_chef_de» il est possible de déduire tous les liens directs et indirects reliant chaque employé à ses supérieurs.

Le principe de récursion

Le processus de calcul qui dérive tous les tuples dépendants par transitivité dans une table, définit la *fermeture transitive* de la table en question. Cet opérateur n'appartient pas à l'ensemble des opérateurs de base de l'algèbre relationnelle. La fermeture transitive représente plutôt une extension naturelle de l'algèbre relationnelle. Elle est formée non pas d'un nombre fixe d'étapes de calcul, mais d'une

Condition de fermeture transitive

combinaison d'opérateurs relationnels de jointure, de projection et d'union, qui varie d'après le contenu d'une table.

Ces considérations nous amènent à la définition suivante en guise de synthèse :

Système à bases de connaissances

Principaux composants d'un système de bases de connaissances

Un système de gestion de bases de connaissances prend en charge des bases de données déductives ou bases de connaissances si :

- à côté des données proprement dites, c'est-à-dire des *faits*, le système contient aussi des *règles* ;
- le *mécanisme d'inférence* permet de dériver de nouveaux faits à partir des faits et règles existants ;
- le système supporte la *réursion* qui permet, entre autres, de calculer la fermeture transitive d'une table.

De la base de connaissances au système expert

La notion de *système expert* désigne un système d'information qui dispose des connaissances et des déductions propres à un domaine d'application spécifique. Une base de connaissances est constituée de trois éléments de base : les faits, les règles et un mécanisme d'inférence pour déduire de nouvelles connaissances. L'évolution des bases de données, des langages de programmation et de l'intelligence artificielle se caractérise par l'interdépendance croissante de ces trois domaines et apportera dans l'avenir de nouvelles approches de solutions efficaces aux problèmes pratiques de l'entreprise (voir les travaux de recherche sur la fouille de données (*data mining*, en anglais) et sur la découverte de connaissances dans les bases de données (*KDD, Knowledge Discovery in Databases*, en anglais)).

6.8 Notes bibliographiques

Littérature de recherche sur les bases de données réparties

Ceri et Pelagatti (1985) ont publié un ouvrage de référence sur les systèmes de bases de données réparties, un aperçu de ce domaine peut être lu chez Özsu et Valduriez (1991). Le livre de Dadam (1996) traite des bases de données réparties et des systèmes clients-serveurs. Dans son ouvrage de langue allemande, Rahm (1994) aborde à la fois

les aspects liés aux systèmes de bases de données réparties et l'architecture des systèmes parallèles multiprocesseurs. Les travaux de fond sur les systèmes de bases de données réparties reposent sur les extensions du «System R» dans Williams et al. (1982), et sur «Ingres» dans Stonebraker (1986). Les développements de Rothnie et al. (1980) présentent un grand intérêt avec le système de bases de données «SDD-1» qui fut le premier prototype des systèmes répartis.

L'intégration du temps dans les bases de données relationnelles fait l'objet de nombreuses propositions ; citons les travaux de Clifford et Warren (1983), Gadia (1988), Jensen (1992) et Snodgrass (1987). Snodgrass et ses collègues de recherche ont défini une extension temporelle du langage SQL ; le langage étendu fut baptisé TSQL2 (voir Snodgrass et al. (1994)). Les projets de recherche actuels sur les bases de données temporelles sont décrits dans Etzion et al. (1998).

Dittrich (1988), Lorie et al. (1985), Meier (1987), Schek et Scholl (1986) présentent l'extension des bases de données relationnelles par l'approche orientée objet. Les livres de Bancilhon (1992), Bertino et Martino (1993), Cattell (1994), Geppert (2002), Heuer (1997), Hughes (1991) et Kim (1990) abordent les bases de données orientées objet. Lausen et Vossen (1996) traitent des aspects fondamentaux des langages de bases de données relationnelles-objet et orientées objet. Les ouvrages en allemand de Kemper et Eickler (2001), Lang et Lockemann (1995), Saake et al. (1997) examinent les récents développements des systèmes de bases de données relationnelles et orientées objet. Meier et Wüst (2003) présentent une introduction aux bases de données orientées objet et relationnelles-objet, destinée aux praticiens. Le livre de Stonebraker (1996) explique les systèmes de bases de données relationnelles-objet. Dans Meier et Wüst (1995) on trouve un aperçu des principaux systèmes de bases de données orientées objet sur le marché ainsi qu'une étude comparative des produits en question. Türker (2003) suit les travaux récents sur l'extension de la norme SQL. Coad et Yourdon (1991), Martin et Odell (1992) définissent les briques de base pour concevoir des bases de données orientées objet. Stein (1994) compare dans son livre les différentes méthodes d'analyse orientées objet. Dietrich et Urban (2005) étudient les principaux thèmes suivants : les avantages du

Littérature sur le traitement du temps dans les bases de données

Travaux sur les bases de données orientées objet et relationnelles-objet

langage UML par rapport à l'approche entité-association dans la modélisation orientée objet, le passage du modèle conceptuel orienté objet au modèle de données relationnel, les bases de données orientées objet, les spécificités du relationnel-objet dans la norme SQL.

*Livres de référence
sur les entrepôts
de données et la
fouille de données*

Ces dernières années ont vu paraître de nombreux ouvrages dédiés aux entrepôts de données (*data warehouse*, en anglais) et à la fouille de données (*data mining*, en anglais). Le livre d'Inmon (2002) est reconnu comme référence sur les entrepôts de données, thème également traité par Kimball et al. (2000), Goglin (1998), Gouarné (1998). Parmi les travaux en allemand, citons Gluchowski et al. (1997), Mucksch et Behme (1996). Jarke et al. présentent les fondements de l'entrepôt de données et les projets de recherche dans ce domaine. Berson et Smith (1997), Witten et Frank (1999) traitent de la fouille de données. Plusieurs auteurs ont contribué à l'étude de l'intégration des bases de données au Web dans l'ouvrage de Meier (2000) dédié à ce thème.

*Travaux de
recherche sur les
bases de données
floues*

Depuis quelques années, des travaux de recherche se consacrent à l'application de la logique floue (*fuzzy logic*, en anglais) à la modélisation de données et aux bases de données : citons par exemple Bordogna et Pasi (2000), Bosc et al. (2004), Bosc et Kacprzyk (1995), Chen (1998), Petry (1996), Pons et al. (2000). L'utilisation de la logique floue a conduit les chercheurs à proposer des extensions, aussi bien au modèle entité-association qu'au modèle relationnel. Ainsi, dans les travaux de Chen (1992), l'extension des formes normales classiques de la théorie des bases de données consiste à définir des dépendances fonctionnelles floues (voir aussi Shenoï et al. (1992)). Dans le livre de Kerre et Chen (1995) on trouve d'autres propositions de modèles de données flous pour concevoir les bases de données. Takahashi (1995) présente FQL (Fuzzy Query Language), un langage de requête floue basé sur le calcul relationnel. Kacprzyk et Zadrozny (1995) proposent FQUERY, langage utilisant des termes flous et implémenté dans un prototype sous Microsoft Access. Une autre approche faisant appel à la classification floue a été initialement proposée par Schindler (1998). Elle est à l'origine du langage de

classification floue fCQL (fuzzy Classification Query Language) dont un prototype a été également mis au point.

Dans les systèmes de bases de données déductives, le langage de règles est généralement connu sous le nom de Datalog, terme combinant le concept de donnée et le langage de programmation logique Prolog. L'ouvrage de Clocksin et Mellish (1994) est une référence classique parmi les manuels de Prolog. Maier et Warren (1988) donne une présentation formelle de la programmation logique des bases de données. Le livre allemand de Cremers et al. (1994) traite en profondeur le domaine des bases de données déductives. Celles-ci occupent aussi une place importante dans les travaux de Gallaire et al. (1984), de Gardarin et Valduriez (1989) et de Gardarin (2000).

*Livres consacrés
aux bases de
données
déductives*

Révision

1. a) *Définissez le concept de gestion de données.*

La gestion de données désigne l'ensemble des tâches et fonctions opérationnelles, organisationnelles et techniques dans les domaines de l'architecture, de l'administration et de la technologie de bases de données, visant à assurer le stockage et l'usage de l'ensemble des données de l'entreprise.

Section 1.4

- b) *Présentez trois profils professionnels en gestion de données.*

Les architectes de données sont responsables de la maintenance de l'architecture de données d'entreprise et de la conception logique des bases de données. Les administrateurs de données assurent la gestion des définitions de tables et d'attributs adoptées par l'entreprise et stockées dans un système de dictionnaire de données. Les experts en bases de données sont chargés de la conception physique et de l'installation des bases de données, de la définition des procédures de sauvegarde et de restauration des bases de données après panne.

Sections 1.4, 2.6, 3.1, 3.7, 4.5

2. a) *Que signifie un système de gestion de bases de données relationnelles (SGBDR) ?*

Un SGBDR est constitué de deux modules «relationnels» : le module de stockage et le module de gestion. Le module de stockage permet d'enregistrer les données et leurs liaisons dans des tables. Outre les tables de données appartenant aux utilisateurs, il existe des tables systèmes prédéfinies où sont stockées les définitions de données. Dans le module de

gestion, le composant le plus important est un langage relationnel de définition et de manipulation de données (par exemple, le langage normalisé SQL). Ce langage inclut également les fonctions utilitaires qui garantissent l'intégrité, la protection et la sécurité des données.

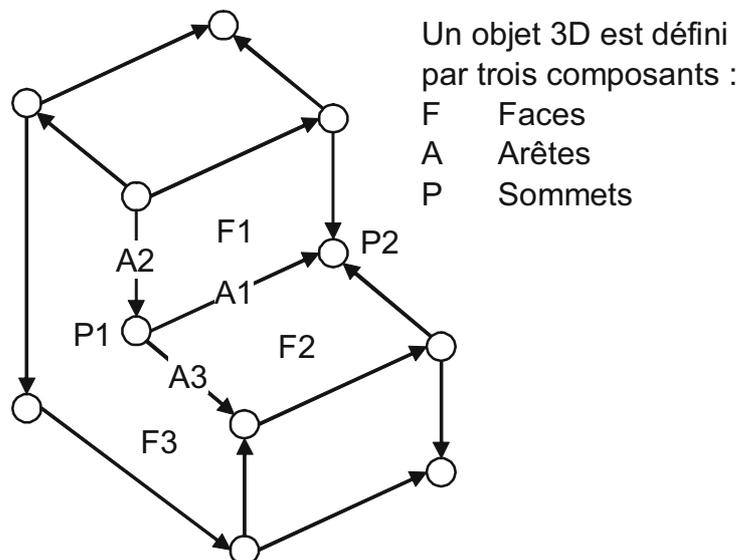
Sections 1.2, 1.3

- b) *Que signifie l'indépendance des données ? Quel est le module d'un SGBDR qui assure cette propriété ?*

Dans un système de bases de données, nous parlons d'indépendance des données lorsque les fonctions du système permettent de séparer les données des programmes d'application. Grâce à cette propriété, nous pouvons effectuer des modifications dans les bases de données sans devoir adapter nos programmes. C'est le module de gestion d'un SGBDR qui permet de réaliser l'indépendance des données.

Section 1.3

3. a) *Créez une liste d'informations factuelles pertinentes pour décrire des objets tridimensionnels limités par des faces planes (polyèdre).*



Analyse de données des objets 3D à faces planes :

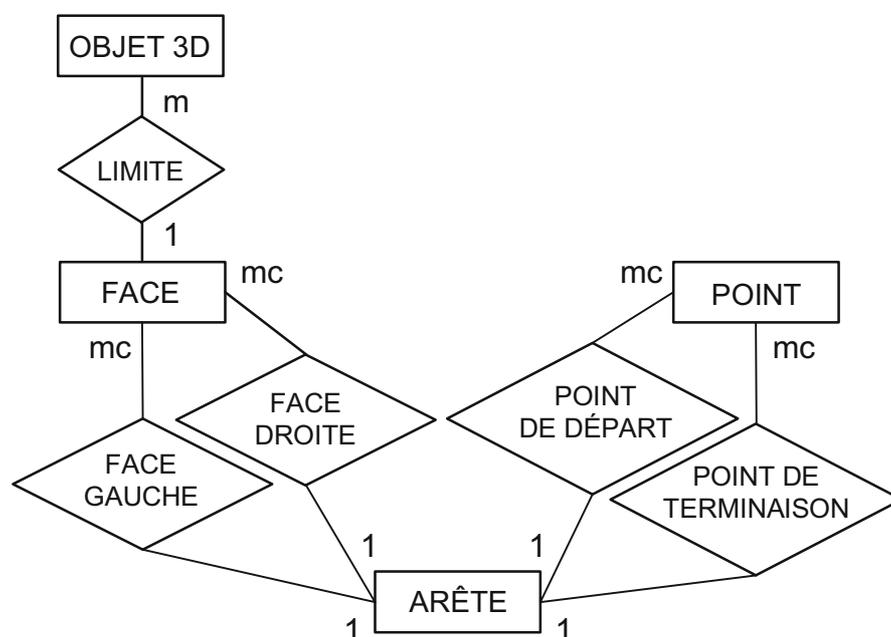
- un objet 3D est formé de plusieurs faces. Chaque face appartient à un seul objet.
- une face est définie par plusieurs arêtes orientées. Chaque arête a *exactement deux* faces adjacentes (par exemple, l'arête A1 possède deux faces adjacentes, F1 à gauche et F2 à droite).
- une arête possède exactement deux sommets ou points extrêmes : un point de départ et un point de terminaison (par exemple, pour l'arête A1, le point de départ est P1, et le point de terminaison P2). Plusieurs arêtes peuvent partir d'un sommet ou y aboutir.

Section 2.1

- b) *Développez un modèle entité-association pour les objets 3D à faces planes (Hypothèse : objets 3D sans ouverture).*

Les relations entre les faces, les arêtes et les sommets s'expriment par deux ensembles de liens duals. L'ensemble d'entités POINT contient les coordonnées cartésiennes.

Section 2.2



4. a) *Transformez le modèle entité-association des objets 3D à faces planes (voir exercice 3b) en un schéma de base de données relationnelle. Quelles sont les règles de transformation appliquées à ce cas précis ?*

Chacun des quatre ensembles d'entités et des cinq ensembles de liens donne lieu à une table (en vertu des règles 1 et 2), soit au total 9 tables. Nous optimisons ce schéma de base de données en appliquant la règle 4 aux liaisons de type simple-complexe. Le nouveau schéma qui en résulte contient 4 tables : l'ensemble de liens LIMITE s'exprime au travers de la table FACE en y définissant une clé étrangère O#_Limite. De même, les ensembles de liens duals FACE DROITE et FACE GAUCHE sont intégrés dans la table ARÊTE en y déclarant les clés étrangères F#_Fdroite et F#_Fgauche. Enfin, les ensembles de liens duals POINT DE DÉPART et POINT DE TERMINAISON sont également intégrés dans la table ARÊTE en y ajoutant les clés étrangères P#_Pdépart et P#_Pterminaison.

Section 2.2

- b) *Représentez le schéma de base de données relationnelle optimal pour les objets 3D à faces planes.*

OBJET_3D

O#	Désignation

FACE

F#	Couleur	O#_Limite

POINT

P#	X	Y	Z

ARÊTE

A#	F#_Fgauche	F#_Fdroite	P#_Pdépart	P#_Pterminaison

Section 2.2

5. a) *Que signifie un langage relationnel complet ?*

Un langage de requête relationnel est complet au sens de l'algèbre relationnelle, s'il supporte au moins les trois opérateurs ensemblistes de base (l'union, la différence et le produit cartésien) et les deux opérateurs de relation (la projection et la sélection).

Sections 3.2, 3.3

Remarque : Les opérateurs d'intersection, de jointure (*join*, en anglais) et de division sont facultatifs, car nous pouvons les exprimer en fonction des opérateurs susmentionnés.

- b) *Quels sont les éléments additionnels que nous devons inclure dans un langage relationnel complet pour qu'il réponde aux exigences de la pratique ? (Citez au moins trois éléments)*

Il faut ajouter les constructions du langage permettant la définition de données (langage de définition de données), la manipulation de données (langage de manipulation de données), la gestion des droits d'utilisation, l'exécution des fonctions de contrôle (par exemple, la reprise après panne, le redémarrage) et la vérification de l'intégrité référentielle.

Section 3.3

6. a) *Dans la base de données des polyèdres limités par des faces planes (voir le schéma de base de données relationnelle à l'exercice 4b, dressez une liste de toutes les désignations d'objets qui présentent des faces rouges.*

```
SELECT Désignation
FROM OBJET_3D, FACE
WHERE O# = O#_Limite AND Couleur = 'rouge'
```

Sections 3.2, 3.4

- b) *Dans l'objet limité par des faces planes, présenté à l'exercice 3a, quelles sont les faces touchant le sommet commun P1 ?*

```

SELECT  F#_Fgauche
FROM    ARÊTE
WHERE   P#_Pdépart = 'P1'
UNION
SELECT  F#_Fdroite
FROM    ARÊTE
WHERE   P#_Pterminaison = 'P1'

```

Idée de solution : Il faut déterminer d'abord toutes les faces gauches des arêtes qui partent du sommet P1 (à savoir F1 et F2), et les combiner ensuite aux faces droites des arêtes aboutissent à P1 (F3).

Sections 3.2, 3.4

7. a) *Que signifie l'intégrité référentielle ?*

Une base de données relationnelle respecte la règle de l'intégrité référentielle si chaque valeur d'une clé étrangère existe comme valeur de la clé primaire correspondante (dans la table référencée).

Section 2.5

b) *Programmez une opération d'insertion INSERT (en langage SQL) dans la table VILLE, qui transgresse la règle d'intégrité référentielle.*

PAYS			VILLE		
Nom	Capitale	Code	Désignation	Population	État
Suisse	Berne	CH	Berne	35000	CH
Allemagne	Berlin	D	Berlin	1860000	D
Italie	Rome	I	Florence	40000	I
			Bâle	60000	CH
			Münich	1290000	D
			Zürich	900000	CH
			Rome	1140000	I
			Fribourg	35000	CH

Réponse :

```

INSERT INTO VILLE
VALUES ('Paris',1400000,'F')

```

Remarque : La valeur «F» (code de la France) n'existe pas (encore) dans la table correspondante PAYS.

Sections 2.5, 3.4

8. a) *On désire obtenir une liste de toutes les villes d'Italie (voir exercice 7. Formulez la requête d'interrogation en SQL.*

```
SELECT    Designation
FROM      PAYS, VILLE
WHERE     Code=Etat AND Nom='Italie'
```

Remarque : La requête produit une table résultat contenant les villes Florence et Rome.

Section 3.4

- b) *Exprimez la requête en utilisant les opérateurs de l'algèbre relationnelle (expression algébrique).*

La table résultat s'obtient en évaluant l'expression algébrique suivante :

$$\pi_{\text{Désignation}} (\sigma_{\text{Nom}='Italie'} (\text{PAYS} \times_{\text{Code}=\text{État}} \text{VILLE}))$$

Section 3.2

9. a) *Exprimez en SQL une requête qui produit la table résultat suivante à partir des tables PAYS et VILLE de l'exercice 7b.*

Désignation	Population	État
Berlin	1860000	D
Münich	1290000	D
Rome	1140000	I

Réponse :

```
SELECT    *
FROM      VILLE
WHERE     Population > 1000000
```

ou

```
SELECT  Designation, Population, Etat
FROM    VILLE
WHERE   Population > 1000000
```

Section 3.4

- b) *Quelle est l'expression en algèbre relationnelle qui produit la même table résultat qu'en 9a.*

La même table résultat s'obtient en évaluant l'expression algébrique suivante :

$$\sigma_{\text{Population} > 1000000}(\text{VILLE})$$

Section 3.2

10. a) *On désire obtenir tous les noms de pays et de villes dans une table unique. Sous quelle condition est-il possible de formuler cette requête ?*

Les tables $\pi_{\text{Nom}}(\text{PAYS})$ et $\pi_{\text{Désignation}}(\text{VILLE})$ doivent être compatibles avec l'union, c'est-à-dire qu'elles doivent être de même dimension et définies dans les mêmes domaines.
Section 3.2

- b) *Exprimez la requête en SQL.*

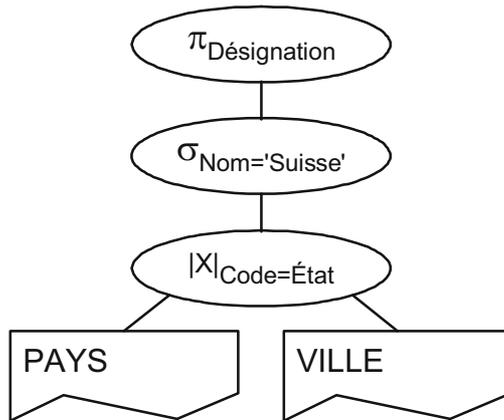
```
SELECT  Nom
FROM    PAYS
UNION
SELECT  Désignation
FROM    VILLE
```

Remarque : Nous admettons ici que les noms des pays et les désignations des villes sont définis dans un même domaine. Sous cette hypothèse, l'union (opérateur UNION en SQL) est possible.

Section 3.4

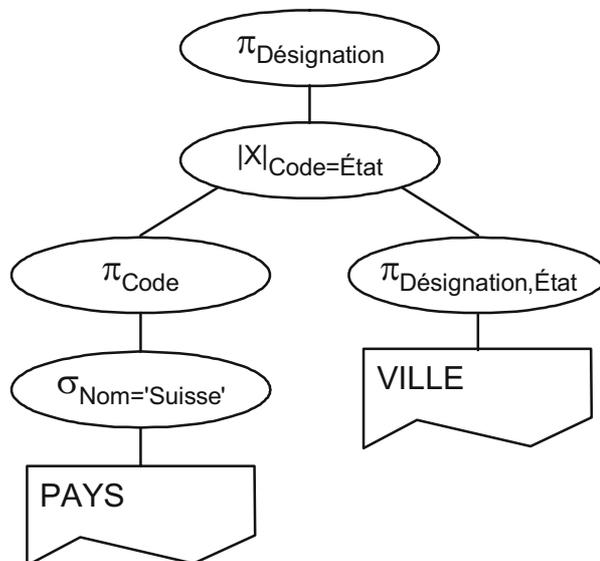
11. a) *Exprimez l'expression algébrique suivante sous la forme d'un arbre d'interrogation :*

$$\pi_{\text{Désignation}} (\sigma_{\text{Nom}='Suisse'} (\text{PAYS} \bowtie_{\text{Code}=\text{État}} \text{VILLE}))$$



Section 4.2

- b) *Construisez un arbre d'interrogation optimisé.*



Section 4.2

12. a) *Pourquoi l'adoption d'un modèle à couches multiples est-elle pertinente pour définir l'architecture des systèmes de bases de données relationnelles ?*

L'architecture des systèmes de bases de données repose sur un principe fondamental qui énonce que les modifications

et les extensions doivent pouvoir se faire localement. Comme dans l'implémentation des systèmes d'exploitation ou d'autres composants logiciels, les systèmes de bases de données relationnelles sont aussi bâtis en plusieurs niveaux indépendants qui communiquent entre eux au travers d'interfaces prédéfinies.

Section 4.6

- b) *Quelles sont les principales fonctions attribuées à la couche supérieure, c'est-à-dire au niveau de l'interface ensembliste ? (présentez au moins trois fonctions)*

La couche supérieure remplit les fonctions de traduction et d'optimisation des requêtes en exécutant les tâches suivantes : traduction des requêtes et résolution des noms, contrôle et autorisation d'accès, optimisation algébrique, vérification des chemins d'accès, contrôle d'intégrité, et éventuellement, génération de codes.

Section 4.6

13. a) *Que signifie une base de données XML ? À quoi peut-elle servir ?*

Une base de données XML contient des hyperdocuments (documents XML) et repose sur un schéma XML qui permet de décrire et de gérer des données semi-structurées (texte, images, graphiques, etc.). Les bases de données XML sont nécessaires, entre autres, aux applications orientées Web dans les entreprises (commerce électronique ; *e-commerce*, en anglais) ou dans les administrations (gouvernement électronique ; *e-government*, en anglais)

Sections 5.1, 5.2

- b) *Comment préserve-t-on les investissements du passé lors d'un changement de système de bases de données ?*

Il existe plusieurs variantes de migration, telles que la conversion assistée par ordinateur des bases de données et des systèmes applicatifs, la transformation des interfaces de langage, la coexistence temporaire avec maintenance

parallèle (synchrone ou asynchrone) des ensembles de données.

Sections 5.3, 5.4, 5.5

14. a) *Quelles sont les faiblesses reconnues aujourd'hui dans l'exploitation des bases de données relationnelles ? Citez au moins trois points critiques.*

En bureautique, dans la conception assistée par ordinateur et la fabrication de circuits ou de composants électroniques, dans les applications orientées Web avec hyperdocuments, ou encore dans les systèmes d'information géographiques, nous constatons des points faibles à trois niveaux suivants :

Description des objets : Les données sont hautement structurées de différentes manières. À côté des données formatées et structurées, il existe aussi des données semi-structurées telles que texte, images et graphiques. Outre les types de données prédéfinis dans les systèmes de bases de données, l'utilisateur doit pouvoir créer de nouveaux types.

Section 6.4

Gestion des transactions : La transaction est l'unité de base dans le contrôle de la cohérence et pour la reprise après panne. Dans le domaine de l'ingénierie, nous observons qu'une transaction dure en général des jours, voire des semaines dans un projet de développement. Or, de telles transactions risquent de ne pas pouvoir être annulées complètement en cas d'erreur ou de panne. En d'autres termes, l'avenir doit nous offrir la possibilité de traiter des transactions de longue durée.

Sections 4.3, 4.5

Archivage : Un système de bases de données doit permettre la gestion des versions. En outre, il doit pouvoir gérer des données temporelles (par exemple, une série chronologique ou une suite de mesures) dans le but de traiter des requêtes d'interrogation sur le passé et le futur.

Section 6.3

- b) *Quelles sont les extensions et les améliorations en perspective ?*

Dans l'avenir, d'une part les bases de données relationnelles bénéficieront des extensions imposées par les besoins de la pratique, et d'autre part de nouvelles générations de bases de données arriveront sur le marché. Les technologies dites post-relationnelles supporteront des systèmes de bases de données réparties, temporelles, relationnelles-objet, multidimensionnelles, floues, déductives.

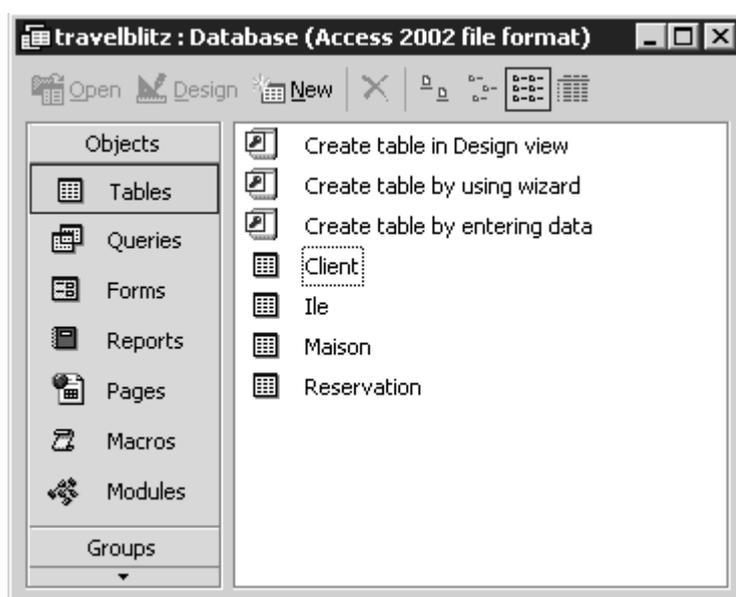
Sections 6.2 à 6.7

La mise en œuvre d'une base de données avec Access

En suivant ce guide pas à pas, vous réaliserez vous-même un projet de base de données simple dans le domaine du tourisme. Votre première tâche consiste à développer un modèle entité-association. Ensuite, vous le transformerez en une structure de tables constituant votre base de données qui sera implémentée avec Access. Après la saisie des données de l'application, vous formulerez vos requêtes pour extraire les informations désirées de votre base de données.

Access est un logiciel de bases de données qui vous permet de gérer vos tables avec une interface graphique. Tous vos ordres sont traduits en instructions SQL pour l'exécution. Access vous offre aussi la possibilité de programmer directement en SQL, ce qui est très utile pour créer des requêtes complexes.

Les objets d'une base de données Access sont gérés dans différents onglets affichés à l'ouverture de la base considérée, comme le montre la figure suivante.



La «fenêtre de base de données»

Il s'agit d'une base de données nommée *travelblitz* avec ses onglets correspondant aux sept types d'objets. Le premier onglet contient les quatre tables de la base de données.

Une base de données Access comporte au minimum les types d'objets suivants :

- Les *tables* (onglet *Tables*) constituent la pierre angulaire de toute base de données Access. Elles contiennent toutes les informations dans une base de données sous forme de tuples (enregistrements).
- Les *requêtes* (onglet *Queries*) vous permettent d'extraire des informations d'une base de données.

La figure ci-dessus contient d'autres onglets qui renferment des objets nécessaires à l'exploitation efficace d'une base de données :

- Les *formulaires* (onglet *Forms*) vous permettent de saisir, d'afficher et de gérer de manière commode les données à l'aide de masques de saisie.
- Les *états* (onglet *Reports*) vous permettent de présenter les données sous une forme intelligible et agréable à lire (pour établir des factures, par exemple).
- Les *pages* (onglet *Pages*) sont à maints égards analogues aux formulaires. En outre, vous pouvez afficher les pages aussi bien dans Access que dans la fenêtre de votre navigateur Web.
- Les *macros* (onglet *Macros*) et les *modules* (onglet *Modules*) vous permettent d'automatiser des procédures de traitement complexes de votre base de données.

Dans ce guide, les figures et les commandes ont été développées avec la version anglaise d'Access 2002 ; mais il ne vous sera pas difficile de les mettre en correspondance avec d'autres versions du logiciel.

Étude de cas : travelblitz

Votre mission consiste à mettre en œuvre une application de base de données pour l'agence de voyage *travelblitz*, spécialisée dans la location de maisons de vacances situées sur des îles grecques. L'agence de voyage prévoit d'étendre ses offres de location à de nouvelles îles dans le futur. À l'heure actuelle, les données sur les clients et les maisons de vacances sont gérées dans une traditionnelle cartothèque, ce qui restreint les possibilités d'extraction des informations. Ainsi, il faut par exemple beaucoup de temps pour savoir quelle maison est libre dans une période déterminée (durant les trois premières semaines de juillet, par exemple) avec un loyer inférieur à 400 francs suisses la semaine. L'agence de voyage décide de remédier à cette situation par l'implantation d'une base de données pour mieux servir sa clientèle.

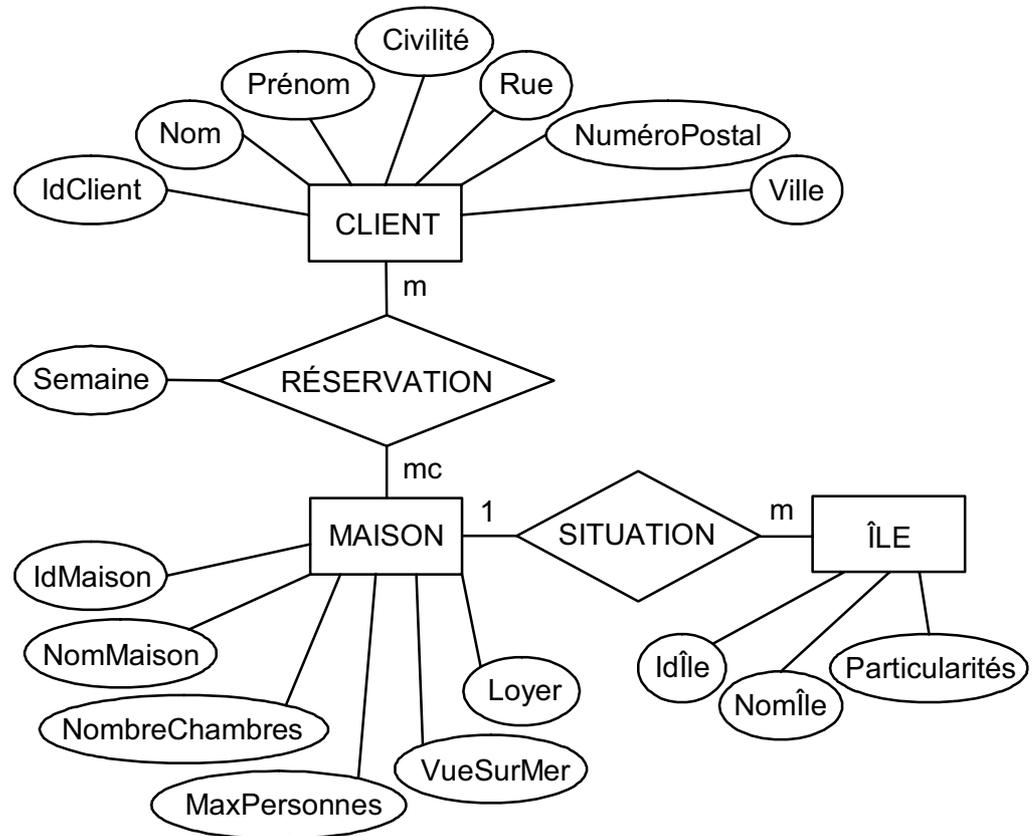
Étape 1 : développer un modèle entité-association

Pour simplifier, votre modèle entité-association repose sur les hypothèses suivantes :

1. La saison dure de la semaine 10 à la semaine 40, soit du début avril à la fin septembre. Le loyer reste fixe durant toute la saison ; les maisons sont louées à la semaine.
2. A chaque réservation, il faut saisir les informations sur la maison, le client et le numéro de la semaine. Si un client loue une maison sur plusieurs semaines consécutives, la location doit être éclatée en plusieurs réservations (une par semaine).
3. Chaque période de réservation, c'est-à-dire la semaine en question, est indiquée par un numéro compris entre 10 et 40. Une nouvelle base de données est créée chaque année.
4. Pour le moment, la base de données ne contient pas d'informations sur les factures, les délais de paiement, etc.

À des fins d'exercice, les hypothèses énoncées visent à réduire le plus possible la taille du modèle de données. Pour construire le

modèle entité-association, vous devez maintenant réfléchir aux questions suivantes : Quels sont les ensembles d'entités et les ensembles de liens à créer ? Quels sont leurs attributs respectifs ? Quelles sont les clés d'identification à définir pour les ensembles d'entités ?



Pour la gestion des maisons de vacances, le modèle entité-association se construit à partir de trois ensembles d'entités, CLIENT, MAISON et ÎLE. L'ensemble de liens RÉSERVATION traduit l'attribution des maisons aux clients ; la liaison est de type complexe-complexe. Vous constatez que l'attribut Semaine est un attribut de liaison typique, car il détermine la réservation d'une maison de vacances par un client dans le temps. Enfin, vous exprimez l'appartenance hiérarchique des maisons de vacances aux îles par l'ensemble de liens SITUATION.

Étape 2 : concevoir le schéma de base de données

Vous vous posez maintenant les questions suivantes : Comment se présente un schéma de base de données relationnelle pour la

gestion des maisons de vacances ? Quelles sont les règles de transformation à appliquer pour convertir le modèle entité-association précédent en tables ?

En vous référant à la section 2.3.2, vous créez pas à pas la structure des tables à partir de votre modèle entité-association :

1. En vertu de la règle de conversion 1, vous devez créer une table distincte pour chaque ensemble d'entités. Pour simplifier, donnez à la table un nom identique à celui de l'ensemble d'entités correspondant. Vous obtenez ainsi les tables suivantes :
 - CLIENT (*IdClient*, Nom, Prénom, Civilité, Rue, NuméroPostal, Ville)
 - MAISON (*IdMaison*, NomMaison, NombreChambres, MaxPersonnes, VueSurMer, Loyer)
 - ÎLE (*IdÎle*, NomÎle, Particularités)

Par convention, écrivez les clés d'identification en italique pour les mettre en évidence.

2. En vertu de la règle de conversion 3, vous devez absolument définir une table distincte pour l'ensemble de liens de type complexe-complexe RÉSERVATION. Elle servira à stocker tous les liens générés par la location des maisons. Outre les clés étrangères qui identifient les maisons (*IdMaison*) et les clients (*IdClient*), la table renferme aussi l'attribut de liaison *Semaine* pour indiquer la période de location, plus précisément, le numéro de la semaine louée.
 - RÉSERVATION (*IdMaison*, *IdClient*, *Semaine*)

Mettez en évidence l'attribut *IdMaison* et le numéro de la semaine qui composent la clé d'identification de la table en question. Elle permet d'éviter de manière sûre la double réservation ou surréservation d'une chambre donnée.

3. L'ensemble de liens SITUATION qui relie les maisons de vacances aux îles est de type simple-complexe. Vous avez deux

possibilités de le traduire dans le schéma de base de données : soit comme une table distincte (règle de conversion 2), soit par l'ajout d'une clé étrangère dans la table des maisons de vacances (règle de conversion 4). Choisissez la deuxième variante en agrandissant la table MAISON comme suit :

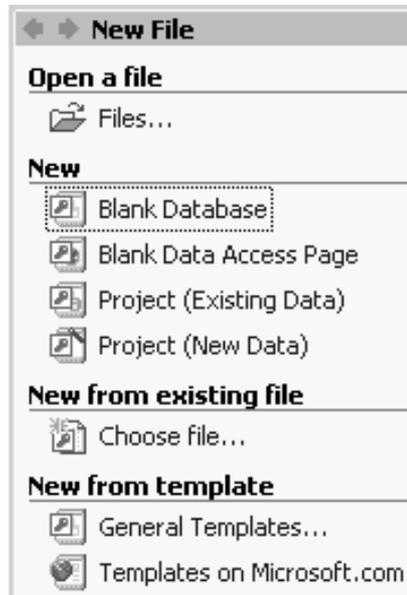
- MAISON (*IdMaison*, NomMaison, NombreChambres, MaxPersonnes, VueSurMer, Loyer, IdÎle_Situation)
4. En conclusion, vous obtenez les quatre tables suivantes dont vous devez encore vérifier les propriétés par rapport à la troisième forme normale :
- CLIENT (*IdClient*, Nom, Prénom, Civilité, Rue, NuméroPostal, Ville)
 - MAISON (*IdMaison*, NomMaison, NombreChambres, MaxPersonnes, VueSurMer, Loyer, IdÎle_Situation)
 - ÎLE (*IdÎle*, NomÎle, Particularités)
 - RÉSERVATION (*IdMaison*, *IdClient*, *Semaine*)

À l'exception de la table CLIENT, toutes les autres sont en troisième forme normale. En effet, l'attribut Ville dépend de l'attribut IdClient par transitivité, via NuméroPostal. C'est pourquoi, vous auriez dû définir une table supplémentaire LOCALITÉ contenant deux attributs, NuméroPostal et Ville. Cependant, pour des raisons pratiques, n'appliquez pas cette décomposition dans le présent exercice.

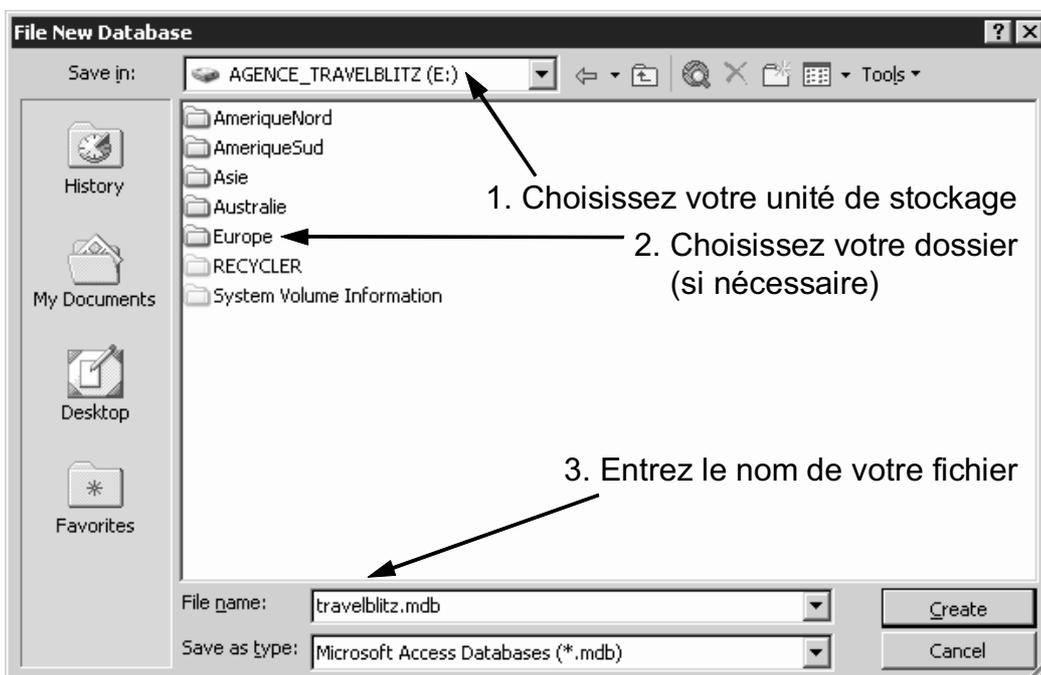
Étape 3 : implanter le schéma de base de données en Access

Démarrez Access et indiquez au système que vous désirez partir avec une nouvelle base de données vide (*Blank Database*) :

Création d'une nouvelle base de données



Access a besoin de savoir où stocker le fichier qui contiendra votre base de données. À cet effet, vous lui précisez l'unité de stockage, le dossier et le nom du fichier dans la fenêtre *File New Database* :



Une fenêtre de base de données s'ouvre aussiôt. Vous l'avez déjà vue au début de ce guide. Naturellement, tous les onglets sont vides au départ.

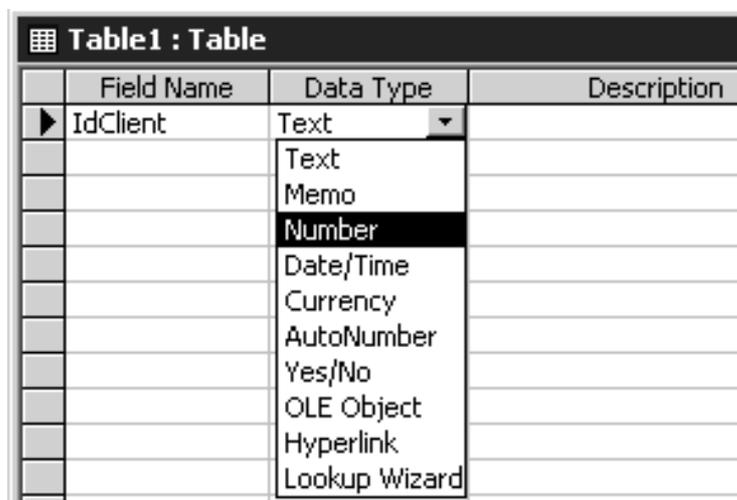
Vous êtes maintenant prêt à définir le schéma de base de données. En premier lieu, vous créez la table des clients en suivant la procédure suivante :

1. Dans l'onglet *Tables*, cliquez sur le bouton *New*. Choisissez ensuite le *mode Création (Design View)*. Access affiche alors une fenêtre de création de table. Vous entrez ligne par ligne les attributs de la table à créer en spécifiant les propriétés suivantes :

- Nom,
- Type de donnée,
- Description concise de l'attribut.

Le nom et la description de l'attribut sont saisis manuellement. Le type de donnée est choisi dans une liste déroulante : dans la colonne *Data Type*, cliquez sur la flèche  pour ouvrir une liste de choix, puis sélectionnez un type de donnée approprié à l'attribut :

Définition d'une table dans Access (choisir le type de donnée)



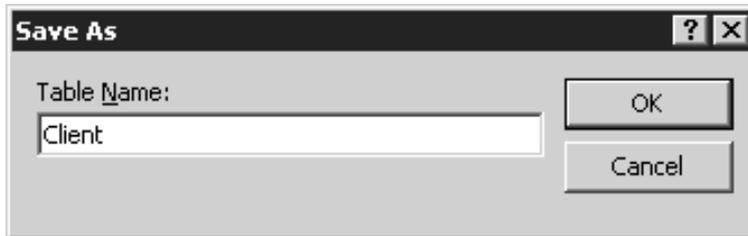
Field Name	Data Type	Description
IdClient	Text	
	Text	
	Memo	
	Number	
	Date/Time	
	Currency	
	AutoNumber	
	Yes/No	
	OLE Object	
	Hyperlink	
	Lookup Wizard	

Les types de données les plus importants dans Access sont : Texte, Numérique, Monétaire, Valeur de vérité (Oui/Non) et Date/Heure.

2. Pour définir une clé primaire, marquez l'attribut clé en cliquant sur le sélecteur d'enregistrement  placé en regard de l'attribut en question, puis choisissez *Edit/Primary Key*. Si la clé primaire se compose de plusieurs attributs, marquez d'abord un attribut

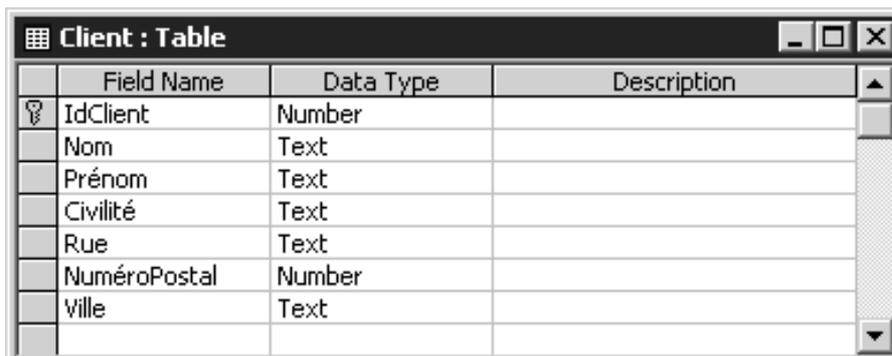
clé, puis, tout en maintenant la touche CTRL enfoncée, marquez les autres attributs membres de la clé.

3. Quand vous fermez la fenêtre de création de table, Access affiche une boîte de dialogue dans laquelle vous introduisez le nom de la nouvelle table :



Introduire le nom d'une table

La fenêtre de création de table suivante contient la définition complète de la table CLIENT :



	Field Name	Data Type	Description
🔑	IdClient	Number	
	Nom	Text	
	Prénom	Text	
	Civilité	Text	
	Rue	Text	
	NuméroPostal	Number	
	Ville	Text	

Définition de la table des clients

Il vous reste maintenant à créer les tables MAISON, ÎLE et RÉSERVATION par la même procédure.

Par la suite, vous définirez les contraintes d'intégrité structurelle qui ont été étudiées dans les sections 2.5 et 3.8. En Access, il existe trois types de règles d'intégrité :

- les contraintes de domaine (traitées à l'étape 4),
- les règles de validation de tuples (traitées à l'étape 5) et
- l'intégrité référentielle (traitée à l'étape 6)

Ces règles d'intégrité peuvent être définies indépendamment les unes des autres. Vous allez introduire maintenant des contraintes de

domaine et de tuple, ainsi que la contrainte d'intégrité référentielle pour les attributs clés étrangères.

Étape 4 : spécifier les contraintes de domaine

Comment restreindre l'ensemble des valeurs possibles d'un attribut ? Par exemple, pour l'attribut Semaine dans la table RÉSERVATION, il faut absolument introduire des nombres compris entre 10 et 40, car l'agence de voyage ne loue pas ses maisons de vacances durant les autres semaines. Comment activer cette contrainte d'intégrité en Access ?

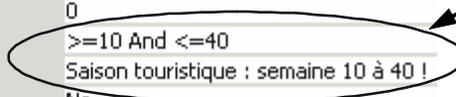
Pour spécifier une contrainte de domaine imposée à l'attribut Semaine, ouvrez la table RÉSERVATION en mode Création, puis cliquez sur le sélecteur de ligne  en regard de l'attribut concerné pour le marquer. Des informations détaillées sur l'attribut marqué s'affichent dans la partie inférieure de la fenêtre. Introduisez une règle de validation pour cet attribut dans le champ *Validation Rule*. Plus tard, lorsque l'utilisateur introduit un numéro de semaine qui enfreint cette règle d'intégrité, le logiciel de base de données rejettera l'opération en affichant un message d'erreur. Le champ *Validation Text* vous permet de personnaliser ce message par un texte intelligible à l'utilisateur. Si vous laissez ce champ vide, le message d'erreur sera un texte standard, pas très parlant, tel que «Valeur interdite».

Contrainte de
domaine d'un
attribut

Reservation : Table			
	Field Name	Data Type	Description
	IdMaison	Number	
	IdClient	Number	
	Semaine	Number	Semaine réservée

Field Properties	
General	
Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	>=10 And <=40
Validation Text	Saison touristique : semaine 10 à 40 !
Required	No
Indexed	No

Règle de validation
avec message d'erreur
pour l'attribut
«Semaine»



Définissez la contrainte d'intégrité de l'attribut *Semaine* et formulez un message d'erreur le plus parlant possible.

Vous admettez les mots suivants comme titre de civilité : "Monsieur", "Madame", "Herr", "Frau". Comment formulez-vous la contrainte d'intégrité correspondante en Access ?

Parmi les contraintes de domaine imposées à un attribut, vous pouvez encore spécifier deux autres propriétés : *Field Size* et *Required* :

- Avec la propriété *Field Size* vous affinez la définition du type de donnée d'un attribut. Ainsi, vous pouvez fixer la longueur maximale des valeurs d'un attribut de type texte, ou déclarer le type de nombre associé à un attribut numérique (Integer pour des nombres entiers, Single ou Double pour des nombres en virgule flottante).
- La propriété *Required* vous permet de rendre obligatoire la saisie d'une valeur non nulle de l'attribut concerné. Le contenu d'un attribut caractérisé par *Required=Yes* ne peut pas être vide.

Remarque : Vous pouvez réaliser simultanément les étapes 3 et 4. En d'autres termes, au fur et à mesure que vous définissez les attributs, vous déclarez leurs propriétés dans la partie inférieure de la fenêtre de création de table.

Étape 5 : définir les règles de validation de tuples

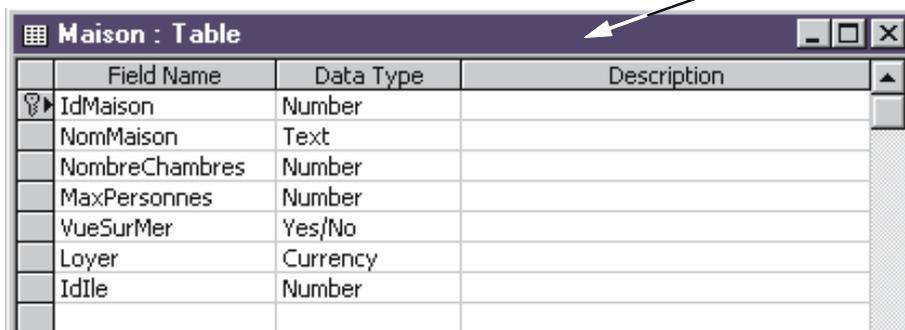
Cette catégorie de contraintes d'intégrité implique plusieurs attributs dont les valeurs sont liées entre elles dans un même tuple. Une règle de ce type énonce par exemple que le loyer de chaque maison doit se baser sur un tarif minimum de 100 francs suisses par chambre. Cette contrainte établit un lien entre deux attributs, *NombreChambres* et *Loyer*, selon l'expression suivante :

$$\text{Loyer} \geq \text{NombreChambres} * 100$$

Pour spécifier une règle de validation de tuples, ouvrez la table en mode Création, puis cliquez sur sa barre de titre bleue avec le bouton droit :

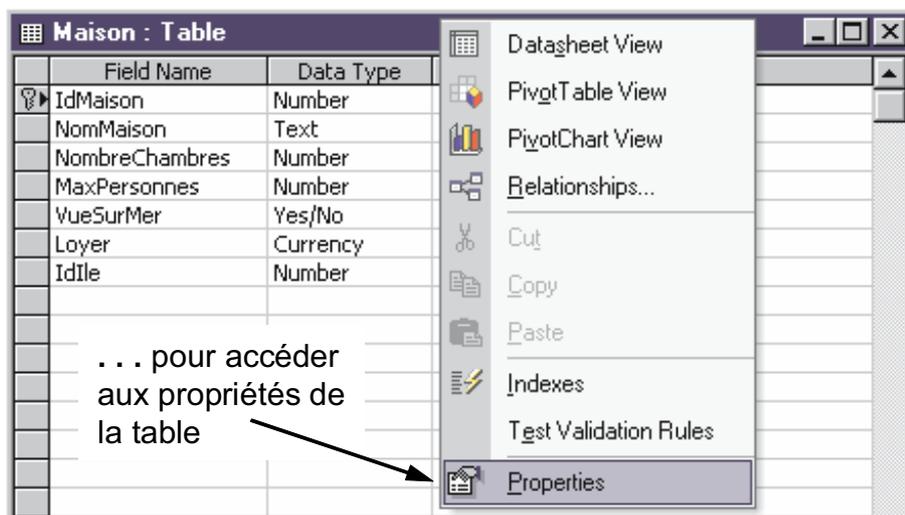
Ouverture d'un menu contextuel ...

En mode Création, pour définir les propriétés d'une table (par exemple, une règle de validation de tuples), pointez la souris sur la barre de titre de la table et cliquez avec le bouton *droit*



Field Name	Data Type	Description
IdMaison	Number	
NomMaison	Text	
NombreChambres	Number	
MaxPersonnes	Number	
VueSurMer	Yes/No	
Loyer	Currency	
IdIle	Number	

Un menu contextuel s'affiche. Sélectionner *Properties* :

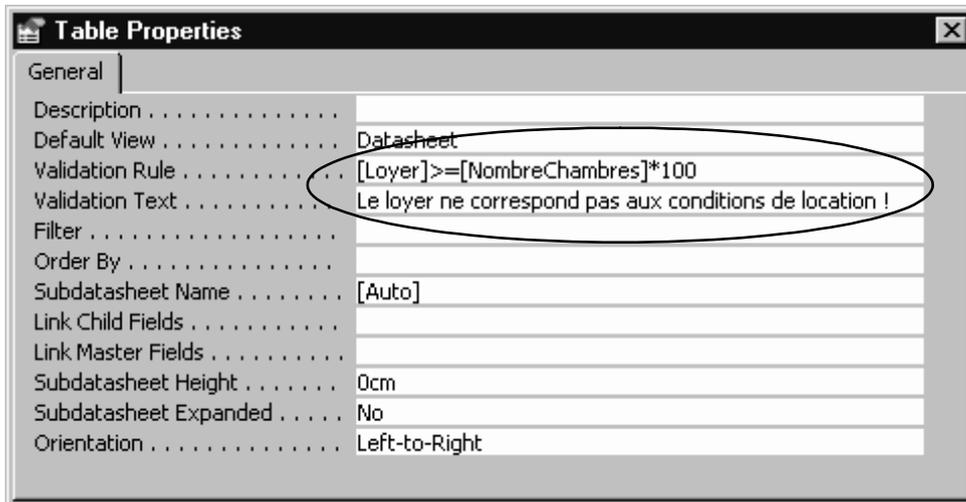


Field Name	Data Type
IdMaison	Number
NomMaison	Text
NombreChambres	Number
MaxPersonnes	Number
VueSurMer	Yes/No
Loyer	Currency
IdIle	Number

... pour accéder aux propriétés de la table

Une fenêtre apparaît avec les propriétés de la table considérée. Dans le champ *Validation Rule*, vous entrez une règle que chaque tuple de la table doit respecter. Le champ *Validation Text* vous permet de personnaliser le message d'erreur qui s'affichera lorsqu'un utilisateur viole cette règle lors de la saisie d'un tuple dans la base de données.

Entrez maintenant la règle d'intégrité définie précédemment pour la table MAISON :



Règle de validation de tuples dans la table MAISON (avec message d'erreur personnalisé)

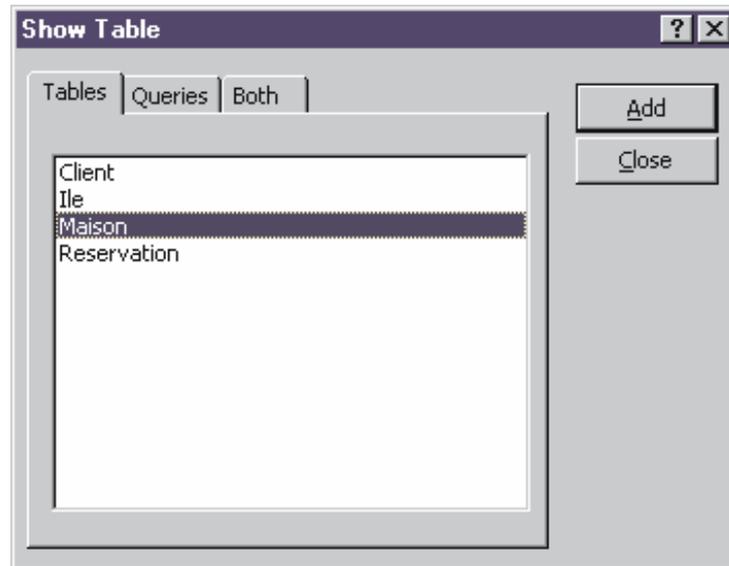
Étape 6 : définir les contraintes d'intégrité référentielle

Vous souvenez-vous de la règle d'intégrité référentielle expliquée dans les sections 2.5 et 3.8 ? Elle empêche qu'un attribut déclaré comme clé étrangère ne reçoive une valeur à laquelle ne correspond aucun tuple dans la table référencée. Dans la base de données de *travelblitz*, vous devez prévenir de telles erreurs de saisie pour chaque attribut clé étrangère en spécifiant des contraintes d'intégrité référentielle appropriées.

Du point de vue conceptuel, chaque lien défini par une clé étrangère sera représenté graphiquement ci-après par un trait qui la connecte à la clé primaire correspondante dans la table référencée. Définissez tout d'abord la contrainte d'intégrité référentielle pour la clé étrangère *IdÎle* dans la table des maisons de vacances. Pour ce faire, suivez les opérations 1 à 3 suivantes :

1. Choisissez la commande *Tools/Relationships*. La fenêtre *Show Table* vous propose une liste des tables pouvant participer à la définition d'une règle d'intégrité référentielle :

La fenêtre
Show Table
pour désigner les
tables dans une
contrainte
d'intégrité
référentielle



2. Choisissez les tables MAISON et ÎLE, cliquez le bouton *Add*, puis le bouton *Close* pour terminer. Les deux tables apparaissent schématiquement dans une fenêtre intitulée *Relationships* (les clés primaires s'affichent en gras) :

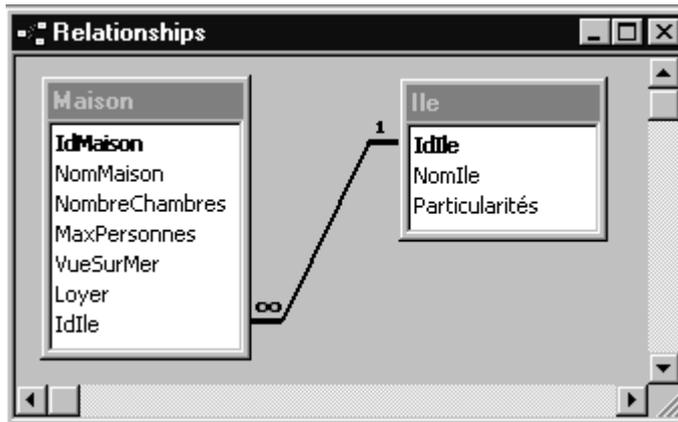
Placement des
tables avant de
définir la contrainte
d'intégrité
référentielle ...



Agrandissez le cadre contenant la table des maisons pour visualiser tous ses attributs comme dans la figure ci-dessus. Glissez la table des îles vers la droite pour augmenter l'espace séparant les deux boîtes (pour déplacer une boîte, cliquez sur la barre de titre et glissez la boîte vers l'endroit désiré tout en pressant le bouton gauche de la souris).

3. Placez à présent le pointeur de la souris sur l'attribut *Maison.IdÎle*, cliquez avec le bouton gauche et, en le maintenant enfoncé, glissez le pointeur vers l'attribut *Île.IdÎle*. À ce moment-là, dans la boîte de dialogue qui apparaît, cochez la case en

regard de l'option *Enforce Referential Integrity*, puis cliquez sur le bouton *Create*. Les deux attributs mentionnés sont alors connectés par un trait continu :



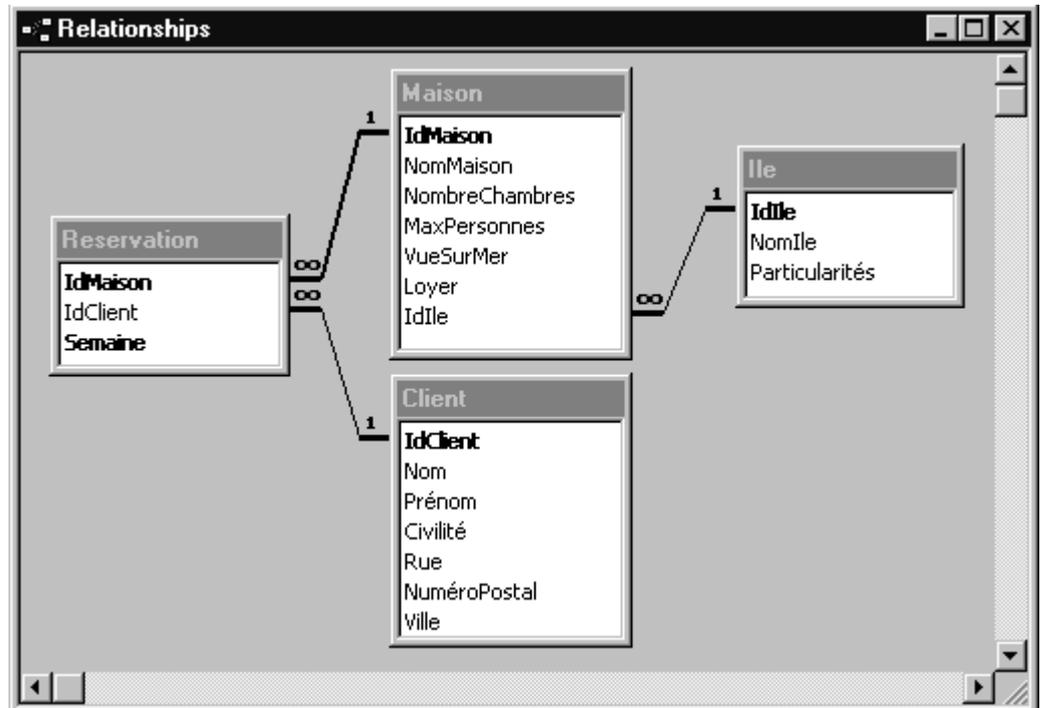
... ensuite

(Au cas où le trait de liaison ne s'affiche pas comme dans la figure ci-dessus, vous avez fait probablement une erreur. Vérifiez si la règle d'intégrité référentielle a été correctement définie. À cette fin, cliquez sur le trait de liaison avec le bouton droit et choisissez *Edit Relationship*. Si cela ne vous permet pas de découvrir l'erreur, vérifiez si les deux attributs ont le même type de donnée, et faites la correction si nécessaire. Si l'erreur persiste, supprimez le trait de liaison et recommencez au début.)

4. Pour terminer la définition des liens par clés étrangères, choisissez la commande *File/Close* ou cliquez sur le bouton Fermer dans le coin supérieur droit de la fenêtre *Relationships*. Sauvegardez les résultats de votre travail.

Définissez maintenant les contraintes d'intégrité référentielle associées aux autres attributs déclarés comme clés étrangères. Appliquez de nouveau la commande *Tools/Relationships*. (Pour réafficher la liste des tables, cliquez avec le bouton droit sur une zone vide dans la fenêtre *Relationships*, sélectionnez *Show Table* dans le menu contextuel, procédez ensuite comme auparavant.)

Si vos manipulations se déroulent correctement, la fenêtre *Relationships* se présente finalement comme suit :



Vous venez de terminer la définition du schéma de base de données. La saisie des données peut maintenant débuter.

Étape 7 : introduire les données

Ouvrez en premier la table CLIENT. Pour ce faire, vous devez d'abord sélectionner l'onglet *Tables* dans la fenêtre Base de données. Il existe deux possibilités d'ouvrir une table : double-cliquer sur la table considérée, ou sélectionner la table et cliquer sur le bouton *Open*.

Une table, vide pour l'instant, s'affiche avec une seule ligne destinée à recevoir les données. Dans Access, le chiffre zéro s'affiche par défaut à l'emplacement des attributs numériques et signale ainsi à l'utilisateur qu'il doit y taper des nombres.

Introduisez maintenant quelques valeurs de donnée dans la table CLIENT :

Saisie des
données dans
Access

Client : Table							
	IdClient	Nom	Prénom	Civilité	Rue	NuméroPostal	Ville
✍	1	Meier	Ursula			0	
*	0					0	

Record: 1 of 1

Voici quelques instructions pratiques pour la saisie des données :

1. Pour passer au champ de donnée suivant, tapez la touche Tabulation ou Entrée. Pour retourner au champ précédent, tapez simultanément la touche Majuscule (\hat{U} , appelée aussi touche *Shift*) et la touche Tabulation.
2. Les touches fléchées (flèche haut, flèche bas, flèche gauche, flèche droite) vous permettent d'atteindre n'importe quel endroit d'une table.
3. Pour annuler une entrée, vous avez besoin de la touche *Esc* (Escape) : tapez cette touche une fois pour rétablir l'ancien contenu de la cellule courante, deux fois pour rétablir le contenu d'un tuple entier.
4. Pour supprimer un enregistrement, marquez la ligne correspondante avec le sélecteur d'enregistrement, puis enfoncez la touche *Del* (Delete).
5. Pour introduire des valeurs logiques (par exemple, pour l'attribut *VueSurMer*), utilisez la souris ou la barre d'espace.
6. Pour fermer une table, choisissez *File/Close* ou cliquer sur .

Introduisez les données suivantes dans les quatre tables indiquées :

IdClient	Nom	Prénom	Civilité	Rue	NuméroPostal	Ville
1	Meier	Ursula	Frau	Lindenstrasse 13	4051	Basel
2	Aeby	Paul	Herr	Rosenweg 26	3001	Bern
3	Gendre	Hélène	Madame	Grand-Rue 11	1700	Fribourg
4	Zumsteg	Irene	Frau	Brückenstrasse 23	3005	Bern
5	Wyss	Beat	Herr	Wallisellenstrasse 243	8050	Zürich

CLIENT

Idlle	Nomlle	Particularités
1	Rhodes	Planche à voile
2	Samos	Golf, Randonnées
3	Crète	Sites historiques, Rochers

ÎLE

MAISON

IdMaison	NomMaison	NombreChambres	MaxPersonnes	VueSurMer	Loyer	IdIle
1	Paphos	3	5	<input checked="" type="checkbox"/>	SFr. 500.00	1
2	Arethoussa	2	4	<input type="checkbox"/>	SFr. 400.00	2
3	Malia	4	6	<input checked="" type="checkbox"/>	SFr. 750.00	1
4	Atrium	3	4	<input checked="" type="checkbox"/>	SFr. 450.00	3

RÉSERVATION

IdMaison	IdClient	Semaine
1	2	28
1	2	29
1	4	31
1	4	32
1	4	33
2	5	17
2	1	31
2	1	32
4	3	25
4	3	26
4	3	27

Si les données sont saisies exactement comme c'est indiqué ci-dessus, aucune contrainte d'intégrité référentielle ne sera violée. En outre, grâce à la clé primaire formée de deux attributs dans la table RÉSERVATION, Access n'admettra aucun doublon causé par la double réservation d'une même chambre à la même période.

Vérifiez si la base de données garantit aussi les règles d'intégrité définies en effectuant les tests suivants :

- Introduire un numéro de semaine invalide dans la table RÉSERVATION.
- Changer "Madame" en "Mademoiselle" dans la formule de civilité d'une cliente.
- Réduire sensiblement le loyer d'une maison (voir la règle de validation des loyers).
- Associer à une maison le numéro d'une île inexistante.
- Supprimer la maison Arethoussa dans la base de données (Pourquoi cette opération est-elle rejetée ?).

La base de données est maintenant mise en exploitation pour gérer les affaires de votre agence de voyage. Par exemple, vous devez

enregistrer un client, Monsieur Ernst Bircher, domicilié au 10 Seestrasse, 6004 Luzern, qui réserve la maison de vacances Malia à la fin août pendant trois semaines (semaines 33 à 35). Traitez cette transaction avec votre base de données.

L'agence de voyage vient d'acquérir Pegasus, une nouvelle maison de vacances sur l'île de Crète. Elle dispose de 5 chambres pouvant héberger 8 personnes au maximum. Son emplacement n'offre pas la vue sur mer. Enregistrez Pegasus dans la base de données.

Étape 8 : interroger la base de données avec QBE

Vous vous intéressez à une multitude de questions : quelles sont les maisons de vacances situées sur l'île de Crète ? Quelle est la maison réservée par la cliente Ursula Meier ? Quelles sont les maisons libres durant les semaines 31 à 33 ? Naturellement, vous pouvez y répondre en consultant les tables ci-dessus. Dans la réalité, lorsque la base de données atteint une certaine taille, l'usage des langages de requête s'impose.

Dans Access, vous pouvez formuler une requête soit en mode QBE avec son interface graphique (voir la section 3.4.3), soit en programmant directement des instructions SQL (voir la section 3.4.1 et la prochaine étape 9).

Vous désirez établir une liste de toutes les maisons, triées dans l'ordre croissant des loyers :

NomMaison	Loyer
Arethoussa	SFr. 400.00
Atrium	SFr. 450.00
Paphos	SFr. 500.00
Pegasus	SFr. 650.00
Malia	SFr. 750.00

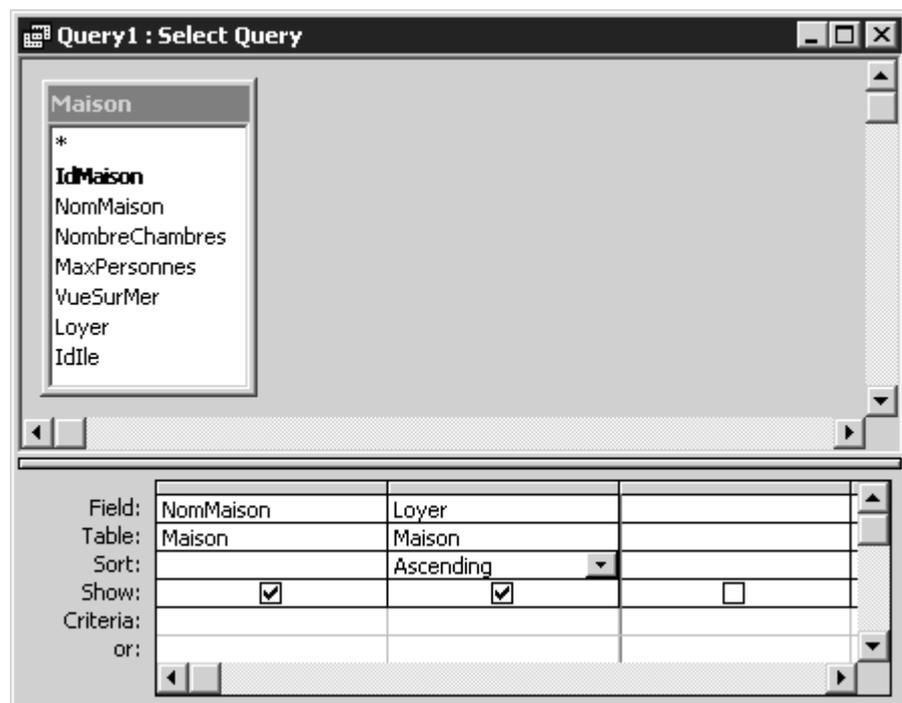
Une requête de sélection simple

En QBE, votre requête est formulée puis exécutée en quatre étapes :

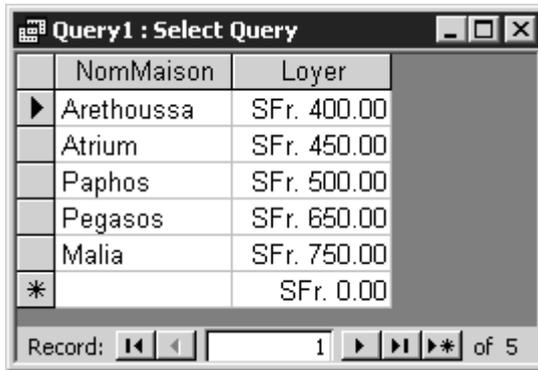
1. Dans la fenêtre Base de données, passez à l'onglet *Queries* et cliquez sur le bouton *New*. Choisissez le mode Création (*Design View*).

2. Dans la fenêtre *Show Table*, sélectionnez les tables que vous voulez interroger et cliquez sur le bouton *Add*, puis sur le bouton *Close*. Une fenêtre de création de requête s'affiche afin que vous puissiez y formuler votre requête en mode QBE.
3. Vous activez une ou plusieurs options suivantes pour chaque attribut dans la fenêtre de création de requête :
 - Afficher l'attribut (*Show*),
 - Trier les enregistrements (*Sort*) dans l'ordre croissant ou décroissant des valeurs de cet attribut,
 - Sélectionner les enregistrements par filtrage d'après un critère de recherche (*Criteria*).

... formulée en
QBE (mode
Création)



4. Pour exécuter votre requête, passez en mode Feuille de données par la commande *View/Datasheet View*. Access produit alors le résultat suivant :



	NomMaison	Loyer
▶	Arethoussa	SFr. 400.00
	Atrium	SFr. 450.00
	Paphos	SFr. 500.00
	Pegasos	SFr. 650.00
	Malia	SFr. 750.00
*		SFr. 0.00

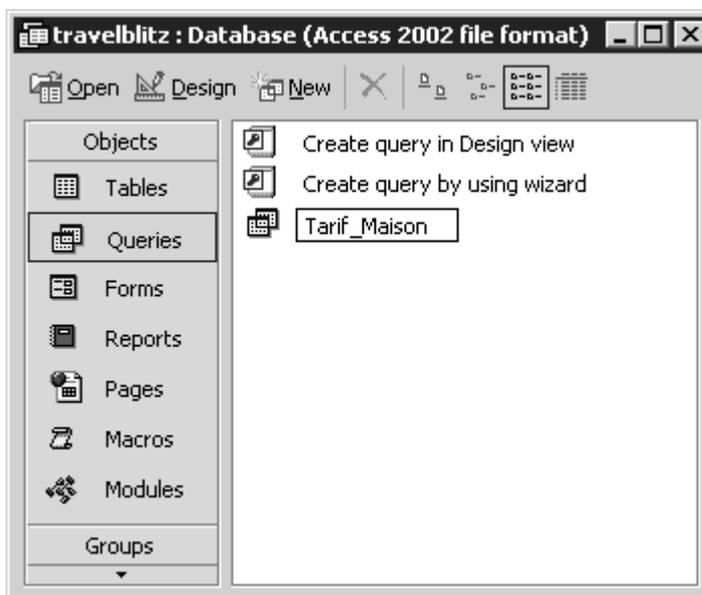
*Résultat de la
requête en mode
Feuille de données*

(Remarque : L'enregistrement vide à la fin de la fenêtre résultat signifie qu'en principe, vous pouvez profiter de cette requête pour introduire un nouvel enregistrement dans la table de base. Cependant, cette pratique est déconseillée pour entrer de nouvelles données.)

Si vous désirez modifier une requête, retournez au mode Création par la commande *View/Design View*, faites les modifications, puis vérifiez le résultat en mode Feuille de données.

5. Si vous envisagez de réutiliser la requête dans le futur, vous devez la sauvegarder (voir la section suivante).

La sauvegarde se fait soit par la commande *File/Save*, soit en fermant la requête. Dans le deuxième cas, Access vous demande automatiquement si vous désirez sauvegarder votre requête. Une fois sauvegardée, la requête figure dans l'onglet *Queries* :



*L'onglet Queries
contient des
requêtes
considérées
comme tables
virtuelles*

À tout moment, vous pouvez exécuter une requête en la sélectionnant puis en cliquant sur le bouton *Open*. Le bouton *Design* vous permet d'ouvrir une requête en mode Création.

Remarquez que le résultat de l'exécution d'une requête se présente comme une table. Toutefois, les données affichées ne sont pas conservées sous forme de table. En revanche, elles sont générées à chaque exécution. Par conséquent, lorsque les données d'une table sont mises à jour, toutes les requêtes qui interrogent la table modifiée produisent automatiquement des résultats différents.

Étape 9 : exécuter des requêtes en SQL

Avant d'exécuter une requête, Access la transforme en commande SQL que vous pouvez aussi examiner. Pour ce faire, ouvrez la requête en mode Création ou Feuille de données, puis choisissez la commande *View/SQL View*.

Exécutez maintenant la requête *Tarif_Maison* que vous venez de créer, puis passez en mode SQL :

*La requête
Tarif_Maison
exprimée en SQL*



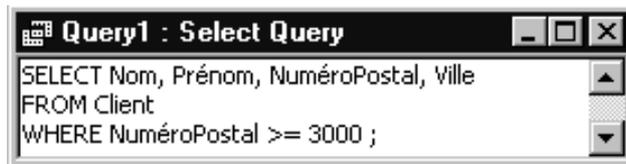
Access préfixe chaque nom d'attribut par le nom de la table correspondante. Dans la plupart des cas, vous n'avez pas besoin d'apporter cette précision et la simple commande SQL suivante suffit :

```
SELECT      NomMaison, Loyer
FROM        Maison
ORDER BY   Loyer
```

Vous désirez obtenir une liste de tous les clients de la Suisse orientale (NuméroPostal \geq 3000) avec noms, prénoms, numéros postaux et villes.

Pour créer cette requête, choisissez l'onglet *Queries*, puis la commande *View/SQL View* pour travailler en mode SQL. Editez

maintenant la commande `SELECT` appropriée dans la fenêtre de création de requête SQL :



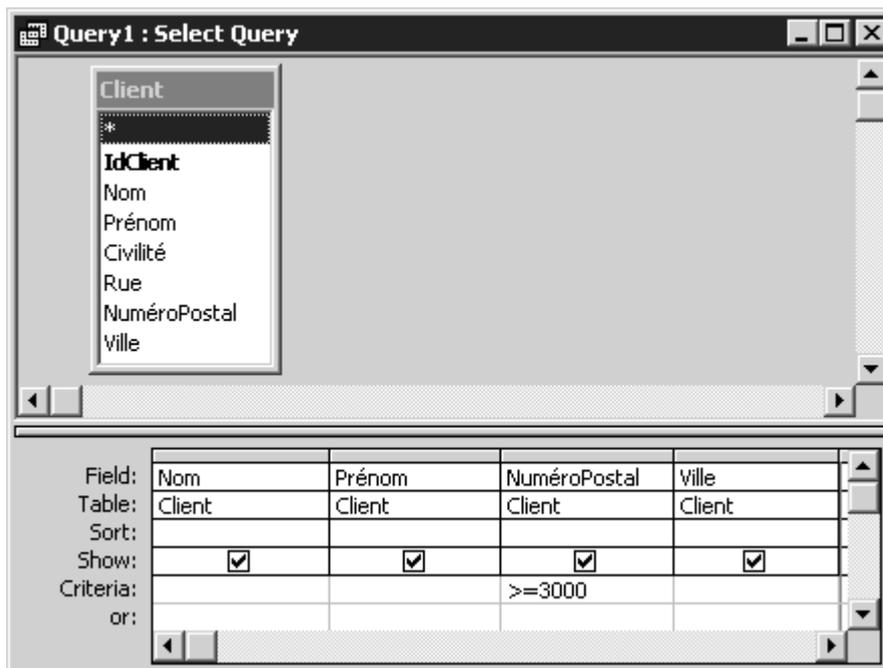
Exemple d'une requête SQL ...

Par la commande *View/Datasheet View*, Access exécute votre requête et produit le résultat suivant :

	Nom	Prénom	NuméroPostal	Ville
▶	Meier	Ursula	4051	Basel
	Aeby	Paul	3001	Bern
	Zumsteg	Irene	3005	Bern
	Wyss	Beat	8050	Zürich
	Bircher	Ernst	6004	Luzern
*				

... qui produit les résultats désirés

Pour voir la formulation équivalente de votre requête en QBE, passez en mode Création par la commande *View/Design View* :



... la même requête formulée en QBE

En comparant la commande SQL à la formulation de la même requête en QBE, vous constatez qu'il faut, dans les deux cas, introduire

les mêmes éléments constitutifs de la requête en question. Par conséquent, pour créer des requêtes simples, choisissez entre SQL et QBE le langage avec lequel vous vous sentez à l'aise. En revanche, vous devez recourir à SQL pour définir des requêtes complexes (par exemple, des commandes SQL emboîtées, voir l'étape 10).

Étape 10 : créer des requêtes complexes (avec jointures ou instructions SQL imbriquées)

Vous avez souvent besoin d'extraire des données provenant de plusieurs tables simultanément. Par exemple, vous désirez avoir une vue d'ensemble de toutes les maisons de vacances avec les noms des îles où elles se trouvent :

*Requête
nécessitant des
données extraites
de deux tables*

NomMaison	NomIle	NombreChambres	MaxPersonnes	VueSurMer	Loyer
Arethoussa	Samos	2	4	<input type="checkbox"/>	SFr. 400.00
Atrium	Crète	3	4	<input checked="" type="checkbox"/>	SFr. 450.00
Malia	Rhodes	4	6	<input checked="" type="checkbox"/>	SFr. 750.00
Paphos	Rhodes	3	5	<input checked="" type="checkbox"/>	SFr. 500.00
Pegasos	Crète	5	8	<input type="checkbox"/>	SFr. 650.00

Le résultat ci-dessus est issu de deux tables, MAISON et ÎLE, liées par l'attribut *IdIle* :

```

SELECT      NomMaison, NomIle, NombreChambres,
            MaxPersonnes, VueSurMer, Loyer
FROM        Maison, Ile
WHERE       Maison.IdIle = Ile.IdIle
ORDER BY   NomMaison

```

Créez cette requête et sauvegardez-la sous le nom `Maison_sur_Ile`.

Comment réalisez-vous la jointure de plus de deux tables ? Par exemple, vous désirez établir une liste de toutes les réservations, contenant les noms des clients et des maisons réservées :

NomMaison	Semaine	Nom
Arethoussa	17	Wyss
Arethoussa	31	Meier
Arethoussa	32	Meier
Atrium	25	Gendre
Atrium	26	Gendre
Atrium	27	Gendre
Malia	33	Bircher
Malia	34	Bircher
Malia	35	Bircher
Paphos	28	Aeby
Paphos	29	Aeby
Paphos	31	Zumsteg
Paphos	32	Zumsteg
Paphos	33	Zumsteg

*Requête impliquant
une double jointure*

Votre requête nécessite trois tables, MAISON, RÉSERVATION et CLIENT :

```

SELECT      NomMaison, Semaine, Nom
FROM        Maison, Reservation, Client
WHERE       Maison.IdMaison = Reservation.IdMaison
            AND Client.IdClient = Reservation.IdClient
ORDER BY   NomMaison, Semaine

```

Créez cette requête et sauvegardez-la sous le nom Plan_Réserveation.

Dans une requête imbriquée, une commande SQL renferme d'autres commandes. Vous désirez obtenir par exemple une liste des maisons de vacances sur l'île de Crète. Tout d'abord, il faut extraire le numéro d'identification de Crète par une commande SQL interne ; ce numéro est ensuite utilisé dans une commande SQL externe :

```

SELECT  NomMaison
FROM    Maison
WHERE   IdIle = ( SELECT IdIle FROM Ile WHERE NomIle='Crète' )

```

instruction SQL externe
instruction SQL interne

Créez et testez cette requête en y insérant à chaque fois un nom d'île différent.

SQL permet de formuler de puissantes requêtes imbriquées, notamment lorsqu'une commande SQL interne traite non pas une seule valeur de comparaison, mais plusieurs valeurs successives provenant des tuples issus d'une requête externe. Pour connaître par exemple les maisons de vacances libres durant les semaines 31 à 33, vous formulerez la requête SQL imbriquée suivante :

```

SELECT    IdMaison, NomMaison
FROM      Maison
WHERE     NOT EXISTS
          ( SELECT    *
            FROM      Reservation
            WHERE     Reservation.IdMaison =
                    Maison.IdMaison
                    AND Semaine >= 31
                    AND Semaine <= 33 )

```

Access vous fournit le résultat suivant que vous pouvez vérifier par rapport à celui de la précédente requête Plan_Réservation.

	IdMaison	NomMaison
▶	4	Atrium
	5	Pegasos
*	0	

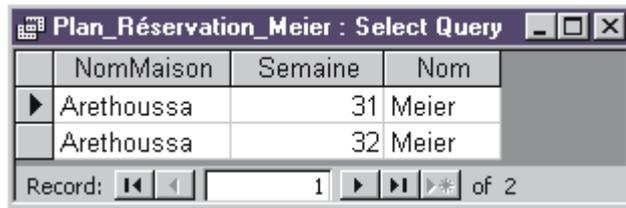
Nous vous présentons encore deux techniques pour exploiter d'autres potentialités du langage SQL. La première consiste à interroger des requêtes considérées elles-mêmes comme des tables :

```

SELECT    *
FROM      Plan_Réservation
WHERE     Nom = 'Meier'

```

Vous obtenez comme résultat tous les enregistrements du client Meier extraits de la requête Plan_Réservation :



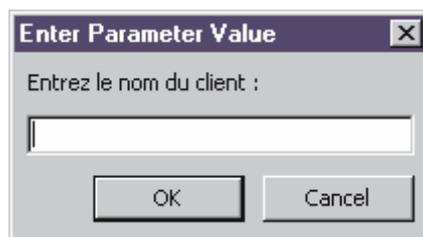
NomMaison	Semaine	Nom
Arethoussa	31	Meier
Arethoussa	32	Meier

La seconde technique consiste à créer en Access des requêtes avec valeurs de comparaison variables. Il s'agit de valeurs que vous devez seulement introduire au lancement d'une requête. Par cette technique, vous pouvez généraliser la requête ci-dessus pour qu'elle puisse s'appliquer à n'importe quel client.

Pour ce faire, remplacez la valeur de comparaison Meier par un message entre crochets, invitant l'utilisateur à entrer le nom d'un client :

```
SELECT      *
FROM        Plan_Réserveation
WHERE       Nom = [Entrez le nom du client :]
```

Au lancement de cette requête, le texte entre crochets apparaît dans une boîte de dialogue dans laquelle vous introduirez la valeur de comparaison manquante :



Access insère cette valeur dans votre requête d'interrogation avant d'exécuter la commande SQL.

Grâce à cette technique, vous êtes en mesure de modifier les deux commandes imbriquées ci-dessus pour qu'elles soient applicables à n'importe quelle île ou à n'importe quelle période de la saison touristique.

Glossaire

Administrateur de données

L'administrateur de données est chargé de la gestion des définitions de données et de fonctions utilisées dans l'entreprise à l'aide d'un système de dictionnaire de données.

Agrégation

L'agrégation est le regroupement des ensembles d'entités comme un tout. Une structure d'agrégation peut être hiérarchique ou en réseau.

Algèbre relationnelle

L'algèbre relationnelle constitue le cadre formel des langages de base de données relationnels. Elle définit les opérateurs suivants : l'union, la différence, le produit cartésien, la projection et la sélection.

Anomalie

Les anomalies sont des écarts par rapport à la réalité, qui surviennent dans une base de données lors des opérations d'insertion, de mise à jour et de suppression .

Arbre

Un arbre est une structure de données dans laquelle chaque nœud, sauf le nœud racine, possède un seul nœud prédécesseur, et chaque feuille est connectée à la racine par un chemin unique.

Association

L'association d'un ensemble d'entités à un autre définit la liaison des deux ensembles dans cette direction. Le type attribué à chaque association exprime le degré de cette liaison orientée.

Base de données floue

Une base de données floue repose sur la logique floue qui lui permet de prendre en charge des faits incomplets, vagues ou imprécis.

Calcul relationnel

Le calcul relationnel repose sur la logique des prédicats. Il utilise les combinaisons logiques de prédicats et les quantificateurs («quel que soit ...» ou «il existe ...»).

Clé

Une clé est une combinaison minimale d'attributs qui identifie de manière unique chaque tuple dans une table.

Contrainte d'intégrité

Les contraintes d'intégrité contiennent la spécification formelle des clés, des attributs et des domaines. Elles visent à garantir la cohérence des données.

Dictionnaire de données

Un système de dictionnaire de données sert à décrire et documenter les éléments de données, les structures de la base de données, les transactions, etc., ainsi que leurs liaisons.

Entité

Les entités sont des objets du monde réel ou dans notre pensée. Elles se caractérisent par leurs attributs et se regroupent en ensembles d'entités.

Entrepôt de données

Un entrepôt de données est un système de bases de données multidimensionnel permettant l'analyse de données représentées dans un cube multidimensionnel.

Forme normale

Les formes normales sont des règles qui permettent de détecter les dépendances à l'intérieur des tables en vue d'éliminer les informations redondantes et les anomalies qui en résultent.

Généralisation

La généralisation est un processus visant à regrouper les attributs communs à plusieurs ensembles d'entités dans un ensemble d'entités supérieur ; dans une hiérarchie de généralisation, les sous-ensembles d'entités sont appelés spécialisations.

Gestion de curseurs

La gestion de curseurs permet de traiter, à l'aide d'un pointeur, un ensemble de tuples enregistré par enregistré.

Hachage

Le hachage est l'organisation fragmentée du stockage de données, basée sur une fonction de hachage qui calcule, à partir d'une clé, l'adresse correspondante d'un enregistrement de données.

Indépendance des données

Un système de base de données garantit l'indépendance des données grâce à ses fonctions qui permettent de séparer les données des programmes d'application.

Index

Un index est une structure de données physique qui contient les adresses internes des enregistrements de données, associées aux attributs désignés dans une table.

Jointure

La jointure est une opération de base de données qui génère une table résultat en rapprochant deux tables par leur attribut commun.

Langage de manipulation de données

Un langage relationnel de manipulation de données permet d'effectuer des opérations ensemblistes sur une base de données pour insérer, modifier et supprimer les entrées dans les tables.

Langage de requête

Un langage de requête relationnel permet le traitement ensembliste des tables, basé sur des critères de sélection ; dans

l'approche ensembliste, le résultat d'une requête consiste toujours en un ensemble de tuples stockés dans une table.

Migration

La migration des programmes de bases de données est le passage, assisté par ordinateur, d'un système de bases de données à un autre par la conversion des données et des applications du système traditionnel vers le système cible.

Modèle de données

Un modèle de données décrit de manière formelle et structurée les données et leurs liaisons dans un système d'information.

Modèle entité-association

Le modèle entité-association est un modèle de données qui définit des classes de données (ensembles d'entités) et leurs liaisons. Les ensembles d'entités sont représentés graphiquement par des rectangles, et les ensembles de liens par des losanges.

Modèle relationnel

Le modèle relationnel est un modèle de données dans lequel les données et leurs liaisons sont toutes deux représentées sous forme de tables.

Optimisation

L'optimisation d'une requête de base de données consiste à transformer l'expression algébrique correspondante (optimisation algébrique) et à exploiter de manière efficace les structures d'accès et de stockage en vue de réduire le coût de traitement.

Orienté objet

Dans l'approche orientée objet, les données sont encapsulées avec leurs méthodes. En outre, le concept de l'héritage s'applique aux propriétés des classes de données.

Protection des données

La protection des données vise à prévenir l'accès non autorisé aux données et leur usage illicite.

Protocole de validation à deux phases

Dans une base de données répartie, le protocole de validation à deux phases assure que toutes les transactions locales se terminent avec succès et que la mise à jour des données s'effectue correctement. Sinon, il défait les transactions, annulant ainsi leur effet sur la base de données.

Protocole de verrouillage à deux phases

Le protocole de verrouillage à deux phases empêche une transaction d'acquiescer un autre verrou après le premier déverrouillage d'un objet de la base de données.

Redondance

La redondance désigne le stockage multiple d'un même fait dans une base de données.

Reprise

La reprise consiste à restaurer l'état cohérent d'une base de données après panne.

Schéma de base de données

Un schéma de base de données relationnelle est la spécification formelle d'une base de données et des tables. Il définit tous les attributs clés et non clés ainsi que les contraintes d'intégrité.

Sécurité des données

La sécurité des données englobe des moyens techniques et des outils logiciels destinés à protéger un système d'information contre la corruption, la destruction et la perte de données.

Sélection

La sélection est une opération de base de données qui extrait des tuples dans une table d'après un critère spécifié par l'utilisateur.

SQL

SQL (Structured Query Language) est le plus important langage relationnel de requête et de manipulation de données. Il est normalisé par l'ISO (International Organization for Standardization).

Synchronisation

Dans un environnement multi-utilisateur, la synchronisation est la coordination des accès simultanés à une base de données. En mode synchronisation pessimiste, le système élimine le plus tôt possible les conflits entre les transactions exécutées en parallèle. En mode synchronisation optimiste, les transactions conflictuelles sont annulées le plus tard possible.

Système de gestion de base de données

Un système de gestion de base de données se compose d'un module de stockage et d'un module de gestion. Le module de stockage permet d'enregistrer les données et leurs liaisons. Le module de gestion offre des fonctionnalités et des langages nécessaires à la maintenance et la gestion des données.

Système hérité

Dans le contexte des bases de données, nous parlons de système hérité (système légué, système patrimoine) lorsque ses structures de données présentent des faiblesses accumulées dans le passé. Cela concerne aussi des ensembles de données qui ne peuvent pas être automatiquement transformés lors d'un changement de système de bases de données.

Table

Une table (relation) est un ensemble de tuples (enregistrements de données) formés d'un nombre déterminé d'attributs. Un attribut ou une combinaison d'attributs identifie de manière unique les tuples dans une table.

Transaction

Une transaction est une suite d'opérations caractérisée par les propriétés d'atomicité, de cohérence, d'isolation et de durabilité. La gestion des transactions permet à plusieurs utilisateurs simultanés de travailler sans conflit dans un système de bases de données.

Utilisateur final

L'utilisateur final d'une application est rattaché à un département fonctionnel de l'entreprise et possède des connaissances de base en informatique.

Valeur nulle

La valeur nulle représente une valeur de donnée qui n'est pas encore connue à un moment donné dans le système de bases de données.

Verrou mortel

Un verrou mortel est un interblocage des transactions dans un environnement multi-utilisateur, c'est-à-dire une situation où les transactions se bloquent mutuellement. Un système de gestion de bases de données doit être capable de détecter et de résoudre les verrous mortels.

XML

Le langage de balisage XML (eXtensible Markup Language) permet de décrire ou de représenter de manière hiérarchique des données et des documents-textes semi-structurés.

Lexique anglais-français

access path	chemin d'accès
after image	image après
aggregation	agrégation
association	association, liaison
atomicity	atomicité, unité indivisible
attribute	attribut, caractéristique, propriété
before image	image avant
B-tree	arbre B
B*-tree	arbre B*
built-in function	fonction prédéfinie, fonction intégrée
candidate key	clé candidate
cascaded deletion	suppression en cascade
checkpoint	point de reprise, point de sauvegarde
commit	valider les changements effectués dans la base de données
concurrency control	synchronisation
conditional	conditionnel
consistency	cohérence
corporate-wide data architecture, enterprise data architecture	architecture de données d'entreprise
create	créer
cursor	curseur

database management system	système de gestion de bases de données
database schema	schéma de base de données
data definition language	langage de définition de données
data dictionary system	système de dictionnaire de données
data manipulation language	langage de manipulation de données
data mining	fouille de données, forage de données
data model	modèle de données
data protection	protection des données
data security	sécurité des données
data warehouse	entrepôt de données, destiné aux applications décisionnelles
date	date
deadlock	verrou mortel, interblocage
declare	déclarer
deductive database	base de données déductive
delete	supprimer
descriptive language	langage descriptif
design	conception, concevoir
difference	différence
distributed database	base de données répartie
domain	domaine
durability	durabilité
enduser	utilisateur final
entity	entité
entity-relationship model	modèle entité-association
entity set	ensemble d'entités
equi-join	équijointure
exclusive lock	verrou exclusif

fetch	chercher
foreign key	clé étrangère
functional dependency	dépendance fonctionnelle
fuzzy database	base de données floue
generalization	généralisation
grant	accorder
grid file	fichier grilles
hash function	fonction de hachage
hashing	hachage
identification key	clé d'identification
index	index
information system	système d'information
insert	insérer
integrity	intégrité, absence d'incohérence
intersection	intersection
is-a structure	structure de généralisation (structure «est un»)
isolation	isolation
join	jointure, joindre
key	clé
key hashing	transformation de clé
knowledge base	base de connaissances
lock	verrou, verrouiller
locking protocol	protocole de verrouillage
log	journal, journaliser
log file	fichier journal
loop	boucle

manipulation language	langage de manipulation de données
multi-dimensional data structure	structure de données multi-dimensionnelle
multi-valued dependency	dépendance multivaluée
nested join	jointure imbriquée
normal form	forme normale
null value	valeur nulle
object-relational database	base de données relationnelle-objet
optimistic concurrency control	mode de synchronisation optimiste
optimization	optimisation
page	page
part-of structure	nomenclature (structure «membre de»)
performance	performance
pessimistic concurrency control	mode de synchronisation pessimiste
point query	requête singulière
precedence graph	graphe de précédence
primary key	clé primaire
procedural language	langage procédural
projection	projection
query language	langage de requête, langage d'interrogation
query tree	arbre d'interrogation
range query	requête d'intervalle
record	enregistrement
recovery	reprise après panne

redundancy	redondance
referential integrity	intégrité référentielle
relation	relation, table
relational algebra	algèbre relationnelle
relational calculus	calcul relationnel
relationship	lien, liaison
repeating group	groupe répétitif
restart	redémarrer
restricted deletion	suppression restreinte
retrieve	extraire
revoke	révoquer
role	rôle
rollback	annuler, défaire
save	sauvegarder
selection	sélection
serializability	sérialisabilité
snapshot	cliché instantané
sort-merge join	jointure par tri-fusion
synchronization	synchronisation
temporal database	base de données temporelle
time	temps
transaction	transaction
transitive dependency	dépendance transitive
transitive closure	fermeture transitive
tree	arbre
two-phase commit protocol	protocole de validation à deux phases
two-phase locking protocol	protocole de verrouillage à deux phases
union	union
unlock	déverrouiller

update

changer, actualiser, mettre à jour

view

vue

Bibliographie

- Akoka J., Comyn-Wattiau I. (2001) Conception des bases de données relationnelles en pratique : Concepts, méthodes et cas corrigés. Vuibert
- Astrahan M. M. et al. (1976) System R : A Relational Approach to Data Base Management. ACM Transactions on Database Systems, Vol.1, No. 2, pp. 97-137
- Balzert H. (Hrsg.) (1993) CASE-Systeme und Werkzeuge. Bibliographisches Institut
- Balzert H. (1999) Lehrbuch der Objektmodellierung - Analyse und Entwurf. Spektrum Akademischer Verlag
- Bancilhon F. et al. (1992) Building an Object-Oriented Database System : The Story of O2. Morgan Kaufmann
- Banos D., Mouyssinat M. (1991) De MERISE aux bases de données - SGBD hiérarchique, réseau, relationel. Eyrolles
- Bayer R. (1972) Symmetric Binary B-Trees : Data Structures and Maintenance Algorithms. Acta Informatica, Vol. 1, No. 4, pp. 290-306
- Bernstein P.A. et al. (1987) Concurrency Control and Recovery in Database Systems. Addison-Wesley
- Berson A., Smith S. (1997) Data Warehousing, Data Mining and OLAP. McGraw-Hill
- Bertino E., Martino L. (1993) Object-Oriented Database Systems. Addison-Wesley
- Biethahn et al. (2000) Ganzheitliches Informationsmanagement (Band II : Daten- und Entwicklungsmanagement). Oldenbourg
- Booch G. (1993) Object-Oriented Analysis and Design with Applications. Benjamin Cummings

- Bordogna G., Pasi G. (Eds.) (2000) Recent Issues on Fuzzy Databases. Physica-Verlag
- Bosc P., Liétard L., Pivert O., Rocacher D. (2004) Gradualité et imprécision dans les bases de données - Ensembles flous, requêtes flexibles et interrogation de données mal connues. Éditions Ellipses
- Bosc P., Kacprzyk J. (Eds.) (1995) Fuzzyness in Database Management Systems. Physica-Verlag
- Brodie M.L., Stonebraker M. (1995) Migrating Legacy Systems - Gateways, Interfaces & The Incremental Approach. Morgan Kaufmann
- Brouard F., Soutou C. (2005) SQL. Collection Synthex : Synthèse de cours et exercices corrigés. Pearson Education France
- Castano S. et al. (1995) Database Security. Addison-Wesley
- Cattell R.G.G. (1997) Bases de données orientées objets. International Thomson Publishing
- Celko J. (1999) Joe Celko's Data & Databases - Concepts in Practice. Morgan Kaufmann
- Celko J. (2000) SQL AVANCÉ - Programmation et techniques avancées. Vuibert
- Ceri S., Pelagatti G. (1985) Distributed Databases - Principles and Systems. McGraw-Hill
- Chen G. (1992) Design of Fuzzy Relational Databases Based on Fuzzy Functional Dependency. Ph.D. Dissertation Nr. 84, Leuven Belgium
- Chen G. (1998) Fuzzy Logic in Data Modeling - Semantics, Constraints and Database Design. Kluwer Academic Publishers
- Chen P.P.-S. (1976) The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No. 1, pp. 9-36
- Clifford J., Warren D.S. (1983) Formal Semantics for Time in Databases. ACM Transactions on Database Systems, Vol. 8, No. 2, pp. 214-254
- Clocksin W.F., Mellish C.S. (1994) Programming in Prolog. Springer

- Coad P., Yourdon E. (1991) Object-Oriented Design. Yourdon Press
- Codd E.F. (1970) A Relational Model for Large Shared Data Banks. Communications of the ACM, Vol. 13, No. 6, pp. 377-387
- Connolly T., Begg C. (2005) Database Systems - A Practical Approach to Design, Implementation and Management. Addison-Wesley
- Cremers A.B. et al. (1994) Deduktive Datenbanken - Eine Einführung aus der Sicht der logischen Programmierung. Vieweg
- Dadam P. (1996) Verteilte Datenbanken und Client/Server-Systeme. Springer
- Date C.J. (1986) Relational Database - Selected Writings. Addison-Wesley
- Date C.J. (2004) Introduction aux bases de données. Vuibert
- Darwen H., Date C.J. (1997) A Guide to the SQL Standard. Addison-Wesley
- Delobel C. et al. (1991) Bases de données : des systèmes relationnels aux systèmes à objets. InterEditions
- Dietrich S.W., Urban S.D. (2005) An Advanced Course in Database Systems - Beyond Relational Databases. Pearson Prentice Hall
- Dippold R., Meier A., Ringgenberg A., Schnider W., Schwinn K. (2001) Unternehmensweites Datenmanagement - Von der Datenbankadministration bis zum modernen Informationsmanagement. Vieweg
- Dittrich K.R. (Ed.) (1988) Advances in Object-Oriented Database Systems. Lecture Notes in Computer Science, Vol. 334, Springer
- Dürr M., Radermacher K. (1990) Einsatz von Datenbanksystemen - Ein Leitfaden für die Praxis. Springer
- Dutka A.F., Hanson H.H. (1989) Fundamentals of Data Normalization. Addison-Wesley
- Eilers H. et al. (1986) SQL in der Praxis. Addison-Wesley
- Elmasri R., Navathe S.B. (2004) Conception et architecture des bases de données. Pearson Education France

- Etzion O., Jajodia S., Sripada S. (Eds.) (1998) *Temporal Databases - Research and Practice*. Lecture Notes in Computer Science, Springer
- Ferstl O.K., Sinz E.J. (1991) Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). *Wirtschaftsinformatik*, Jhrg. 33, Nr. 6, S. 477-491
- Findler N.V. (Ed.) (1979) *Associative Networks - Representation and Use of Knowledge by Computers*. Academic Press
- Gadia S.K. (1988) A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, Vol. 13, No. 4, pp. 418-448
- Gallaire H. et al. (1984) *Logic and Databases : A Deductive Approach*. *ACM Computing Surveys*, Vol. 16, No. 2, pp. 153-185
- Garcia-Molina H., Ullman J.D., Widom J. (2000) *Database System Implementation*. Prentice-Hall
- Gardarin G. (2000) *Bases de données - Objet & relationnel*. Eyrolles
- Gardarin G. (2003) *Bases de données*. Eyrolles
- Gardarin G., Valduriez P. (1989) *Relational Databases and Knowledge Bases*. Addison-Wesley
- Gemünden G., Schmitt M. (1991) Datenmanagement in deutschen Grossunternehmen - Theoretischer Ansatz und empirische Untersuchung. *Information Management*, Jhrg. 6, Nr. 4, S. 22-34
- Geppert A. (2002) *Objektrelationale und objektorientierte Datenbankkonzepte und -systeme*. dpunkt
- Gillenson M.L. (1990) Physical Design Equivalences in Database Conversion. *Communications of the ACM*, Vol. 33, Nr. 8, pp. 120-131
- Gluchowski P., Gabriel R., Chamoni P. (1997) *Management Support Systeme - Computergestützte Informationssysteme für Führungskräfte und Entscheidungsträger*. Springer
- Goglin J.-F. (1998) *La construction du datawarehouse : du datamart au dataweb*. Hermes

- Gouarné J.-M. (1998) Le projet décisionnel : enjeux, modèles et architectures du Data Warehouse. Eyrolles
- Gray J., Reuter A. (1993) Transaction Processing - Concepts and Techniques. Morgan Kaufmann
- Härder T. (1978) Implementierung von Datenbanksystemem. Hanser
- Härder T., Rahm E. (1999) Datenbanksysteme - Konzepte und Techniken der Implementierung. Springer
- Härder T., Reuter A. (1983) Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys, Vol. 15, Nr. 4, pp. 287-317
- Heinrich L.J. (1999) Informationsmanagement - Planung, Überwachung und Steuerung der Informationsinfrastruktur. Oldenbourg
- Hernandez M.J., Viescas J.L. (2001) Introduction aux requêtes SQL. Eyrolles
- Heuer A. (1997) Objektorientierte Datenbanken. Addison-Wesley
- Heuer A., Saake G. (2000) Datenbanken - Konzepte und Sprachen. MITP
- Hitz M., Kappel G. (2002) UML@Work - Von der Analyse zur Realisierung. dpunkt
- Hoffer J.A., Prescott M.B., McFadden F.R. (2004) Modern Database Management. Pearson Prentice Hall
- Hughes J.G. (1991) Object-Oriented Databases. Prentice-Hall
- Hüsemann F.C. (2002) Datenbankmigration - Methodik und Softwareunterstützung. VDI Verlag
- Inmon W.H. (2002) Building the Data Warehouse. Wiley
- Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P. (2000) Fundamentals of Data Warehouses. Springer
- Jensen C.S., Clifford J., Gadia S.K., Seger A., Snodgrass R.T. (1992) A Glossary of Temporal Database Concepts. In : SIGMOD RECORD, Vol. 21, Nr. 3. ACM Press
- Jones A., Stephens R.K., Plew R.R., Garrett R.F., Kriegel A. (2005) SQL Functions - Programmer's Reference. Wiley

- Kacprzyk J., Zadrozny S. (1995) FQUERY for Access - Fuzzy Querying for a Windows-Based DBMS. In : Bosc and Kacprzyk (1995), pp. 415-433
- Kazakos W., Schmidt A., Tomczyk P. (2002) Datenbanken und XML - Konzepte, Anwendungen, Systeme. Springer
- Kemper A., Eickler A. (2001) Datenbanksysteme - Eine Einführung. Oldenbourg
- Kerre E.E., Chen G. (1995) An Overview of Fuzzy Data Models. In : Bosc and Kacprzyk (1995), pp. 23-41
- Kimball R., Reeves L., Ross M., Thornthwaite W. (2000) Concevoir et déployer un data warehouse - Guide de conduite de projet. Eyrolles
- Kim W. (1990) Introduction to Object-Oriented Databases. The MIT Press
- Klettke M., Meyer H. (2003) XML & Datenbanken - Konzepte, Sprachen und Systeme. dpunkt
- Kuhlmann G., Müllmerstadt F. (2000) SQL - Der Schlüssel zu relationalen Datenbanken. Rowohlt
- Kurbel K., Strunz H. (Hrsg.) (1990) Handbuch der Wirtschaftsinformatik. C.E. Poeschel
- Lang S.M., Lockemann P.C. (1995) Datenbankeinsatz. Springer
- Lans R.F. van der (1988) Das SQL-Lehrbuch. Addison-Wesley
- Lausen G., Vossen G. (1996) Objekt-orientierte Datenbanken : Modelle und Sprachen. Oldenbourg
- Lentzner R. (2004) SQL3 avec Oracle, MySQL, Microsoft SQL Server et Access. Dunod
- Lockemann P.C., Schmidt J.W. (Hrsg.) (1993) Datenbank-Handbuch. Springer
- Loeser H. (2001) Web-Datenbanken - Einsatz objekt-relationaler Datenbanken für Web-Informationssysteme. Springer
- Lorie R.A. et al. (1985) Supporting Complex Objects in a Relational System for Engineering Databases. In : Kim W. et al. (Ed.) Query Processing in Database Systems. Springer, pp. 145-155

- Ma Z. (2005) Fuzzy Database Modeling with XML. Advances in Database Systems, Vol. 29. Springer Science & Business Media
- Maier D. (1983) The Theory of Relational Databases. Computer Science Press
- Maier D., Warren D.S. (1988) Computing with Logic - Logic Programming with Prolog. Benjamin/ Cummings
- Martin J. (1986) Einführung in die Datenbanktechnik. Hanser
- Martin J. (1990) Information Engineering - Planning and Analysis. Prentice-Hall
- Martin J., Odell J.J. (1992) Object-Oriented Analysis and Design. Prentice-Hall
- Martiny L., Klotz M. (1989) Strategisches Informationsmanagement. Oldenbourg
- Maurer W.D., Lewis T.G. (1975) Hash Table Methods. ACM Computing Surveys, Vol. 7, Nr. 1, pp. 5-19
- Meier A. (1987) Erweiterung relationaler Datenbanksysteme für technische Anwendungen. Informatik-Fachberichte, Bd. 135, Springer
- Meier A. (1994) Ziele und Aufgaben im Datenmanagement aus der Sicht des Praktikers. Wirtschaftsinformatik, Jhrg. 36, Nr. 5, S. 455-464
- Meier A. (1997) Datenbankmigration - Wege aus dem Datenchaos. Praxis der Wirtschaftsinformatik, Jhrg. 34, Nr. 194, S. 24-36
- Meier A. (Hrsg.) (2000) WWW & Datenbanken. Praxis der Wirtschaftsinformatik, Jhrg. 37, Heft 214, dpunkt
- Meier A., Dippold R. (1992) Migration und Koexistenz heterogener Datenbanken - Praktische Lösungen zum Wechsel in die relationale Datenbanktechnologie. Informatik-Spektrum, Jhrg. 15, Nr. 3, S. 157-166
- Meier A., Graf H., Schwinn K. (1991) Ein erster Schritt zu einem globalen Datenmodell. Information Management, Jhrg. 6, Nr. 2, S. 42-48

- Meier A., Haltinner R., Widmer-Itin B. (1993) Schutz der Investitionen beim Wechsel eines Datenbanksystems. *Wirtschaftsinformatik*, Jhrg. 35, Nr. 4, S. 331-338
- Meier A., Johner W. (1991) Ziele und Pflichten der Datenadministration. *Theorie und Praxis der Wirtschaftsinformatik*, Jhrg. 28, Nr. 161, S. 117-131
- Meier A., Savary C., Schindler G., Veryha Y. (2001) Database Schema with Fuzzy Classification and Classification Query Language. *Proc. of the International Congress on Computational Intelligence - Methods and Applications, CIMA 2001, University of Wales, Bangor U.K.*
- Meier A., Wüst T. (1995) Objektorientierte Datenbanksysteme - Ein Produktvergleich. *Theorie und Praxis der Wirtschaftsinformatik*, Jhrg. 32, Nr. 183, S. 24-40
- Meier A., Wüst T. (2003) Objektorientierte und objektrelationale Datenbanken - Ein Kompass für die Praxis. *dpunkt*
- Miranda S. (2002) Bases de données - Architecture, modèles relationnels et objets, SQL3. *Dunod*
- Morin A., Bosc P., Hebrail G., Lebart L. (2002) Bases de données et statistique. *Dunod*
- Mucksch H., Behme W. (Hrsg.) (1996) Das Data-Warehouse-Konzept - Architektur, Datenmodelle, Anwendungen. *Gabler*
- Nanci D., Espinasse B. (2001) Ingénierie des systèmes d'information : Merise - Deuxième génération. *Vuibert*
- Nievergelt J., Hinterberger H., Sevcik K.C. (1984) The Grid File : An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, Vol. 9, No. 1, pp. 38-71
- Olle T.W. et al. (1988) Information Systems Methodologies - A Framework for Understanding. *Addison-Wesley*
- Ortner E. et al. (1990) Entwicklung und Verwaltung standardisierter Datenelemente. *Informatik-Spektrum*, Jhrg. 13, Nr. 1, S 17-30
- Österle H. et al. (1991) Unternehmensführung und Informationssystem - Der Ansatz des St. Galler Informationssystem-Managements. *Teubner*

- Özsu M.T., Valduriez P. (1991) Principles of Distributed Database Systems. Prentice-Hall
- Panny W., Taudes (2000) Einführung in den Sprachkern SQL-99. Springer
- Paredaens J. et al. (1989) The Structure of the Relational Database Model. Springer
- Petry F.E. (1996) Fuzzy Databases - Principles and Applications. Kluwer Academic Publishers
- Pistor P. (1993) Objektorientierung in SQL3 : Stand und Entwicklungstendenzen. Informatik-Spektrum, Jhrg. 16, Nr. 2, S. 89-94
- Pons O., Vila M.A., Kacprzyk J. (Eds.) (2000) Knowledge Management in Fuzzy Databases. Physica-Verlag
- Pucheral P. (2004) Bases de données avancées - Modèles, systèmes et usages. Numéro spécial de la revue RSTI - Technique et science informatiques. Hermès/Lavoisier
- Rahm E. (1994) Mehrrechner-Datenbanksysteme. Addison-Wesley
- Rahm E., Vossen G. (Hrsg.) (2003) Web & Datenbanken - Konzepte, Architekturen, Anwendungen. dpunkt
- Ramakrishnan R., Gehrke J. (2003) Database Management Systems. McGraw-Hill
- Reingruber M.C., Gregory W.W. (1994) The Data Modeling Handbook - A Best-Practice Approach to Building Quality Data Models. Wiley
- Reuter A. (1981) Fehlerbehandlung in Datenbanksystemen. Hanser
- Riordan R.M. (2005) Designing Effective Database Systems. Addison-Wesley
- Rothnie J.B. et al. (1980) Introduction to a System for Distributed Databases. ACM Transactions on Database Systems, Vol. 5, No. 1, pp. 1-17
- Rumbaugh J. et al. (1991) Object Oriented Modelling and Design. Prentice-Hall
- Saake G. et al. (1997) Objektdatenbanken. International Thomson Publishing

- Sauer H. (1992) *Relationale Datenbanken - Theorie und Praxis inklusive SQL-2*. Addison-Wesley
- Scheer A.-W. (1991) *Architektur integrierter Informationssysteme - Grundlagen der Unternehmensmodellierung*. Springer
- Schek H.-J., Scholl M.H. (1986) The Relational Model with Relation-Valued Attributes. *Information Systems*, Vol. 11, No. 2, pp. 137-147
- Schindler G. (1998) *Fuzzy Datenanalyse durch kontextbasierte Datenbankabfragen*. Deutscher Universitäts-Verlag
- Schlageter G., Stucky W. (1983) *Datenbanksysteme : Konzepte und Modelle*. Teubner
- Schöning H. (2003) *XML und Datenbanken - Konzepte und Systeme*. Hanser
- Shenoi S., Melton A., Fan L.T. (1992) Functional Dependencies and Normal Forms in the Fuzzy Relational Database Model. *Information Sciences*, Vol. 60, pp. 1-28
- Silberschatz A., Korth H.F., Sudarshan S. (2005) *Database System Concepts*. McGraw-Hill
- Silverston L. (2001a) *The Data Model Resource Book, Revised Edition, Volume 1 - A Library of Universal Data Models for All Enterprises*. Wiley
- Silverston L. (2001b) *The Data Model Resource Book, Revised Edition, Volume 2 - A Library of Universal Data Models by Industry Types*. Wiley
- Simsion G.C., Witt G.C. (2005) *Data Modeling Essentials*. Morgan Kaufmann
- Smith J.M., Smith D.C.P. (1977) Database Abstractions : Aggregation and Generalization. *ACM Transactions on Database Systems*, Vol. 2, No. 2, pp. 105-133
- Snodgrass R.T. (1987) The Temporal Query Language TQuel. *ACM Transactions on Database Systems*. Vol. 12, No. 2, pp. 247-298
- Snodgrass R.T. et al. (1994) A TSQL2 Tutorial. *SIGMOD-Record*, Vol. 23, No. 3, pp. 27-33

- Stein W. (1994) Objektorientierte Analysemethoden - Vergleich, Bewertung, Auswahl. Bibliographisches Institut
- Stephens R.K., Plew R.R. (2001) Conception de bases de données. CampusPress
- Stonebraker M. (Ed.) (1986) The Ingres Papers. Addison-Wesley
- Stonebraker M. (1996) Object-Relational DBMS's - The Next Great Wave. Morgan Kaufmann
- Takahashi Y. (1995) A Fuzzy Query Language for Relational Databases. In : Bosc and Kacprzyk (1995), pp. 365-384
- Taylor A.G. (2003) SQL For Dummies. Wiley
- Tsichritzis D.C., Lochovsky F.H. (1982) Data Models. Prentice-Hall
- Türker C. (2003) SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML. dpunkt
- Ullman J. (1982) Principles of Database Systems. Computer Science Press
- Ullman J. (1988) Principles of Database and Knowledge-Base Systems. Computer Science Press
- Vetter M. (1998) Aufbau betrieblicher Informationssysteme mittels pseudo-objektorientierter, konzeptioneller Datenmodellierung. Teubner
- Vossen G. (2000) Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. Oldenbourg
- Vossen G., Witt K.-U. (1988) Das SQL/DS-Handbuch. Addison-Wesley
- Wedekind H. (1991) Datenbanksysteme - Eine konstruktive Einführung in die Datenverarbeitung in Wirtschaft und Verwaltung. Spektrum Akademischer Verlag
- Weikum G. (1988) Transaktionen in Datenbanksystemen - Fehler-tolerante Steuerung paralleler Abläufe. Addison-Wesley
- Weikum G., Vossen G. (2002) Transactional Information Systems - Theory, Algorithms and the Practice of Concurrency Control and Recovery. Morgan Kaufmann

- Wiederhold G. (1983) Database Design. McGraw-Hill
- Williams K. et al. (2001) XML et les bases de données. Wrox Press et les Éditions Eyrolles
- Williams R. et al. (1982) R* : An Overview of the Architecture. In : Scheuermann P. (Ed.) Improving Database Usability and Responsiveness. Academic Press, pp. 1-27
- Witten I.H., Frank E. (1999) Data Mining. Morgan Kaufmann
- Zehnder C.A. (2002) Informationssysteme und Datenbanken. vdf Hochschulverlag
- Zloof M.M. (1977) Query-by-Example : A Data Base Language. IBM Systems Journal. Vol. 16, No. 4, pp. 324-343

Index

A

Absence de données incohérentes

119

Accès

chemin d'accès 63, 107, 132,
146

structure d'accès 107, 147

Agrégation 25, 37, 62, 194, 196

Algèbre relationnelle 69, 82, 109

Annulation d'une transaction 119,
143

Anomalie 42

ANSI 5

Approche

optimiste 129

pessimiste 124

Arbre 108, 132

Arbre B 133

arbre B* 135

Arbre d'interrogation 109, 185

optimisé 113

Arbre multiple 132, 146

orienté feuilles 135

Architecture du système 145

Association

conditionnelle 23, 33

multiple 24, 34

multiple conditionnelle 24, 35

simple 23, 33, 37

Atomicité 119

Attribut 1, 20, 42

Autonomie 184, 187

B

Base de connaissances 214

Base de données 8

base de connaissances 210

expert en bases de données 67

floue 203

multidimensionnelle 197

relationnelle 8

relationnelle-objet 192

répartie 182

temporelle 188

Base de méthodes 213

C

Calcul des adresses. *Voir*

Hachage

Calcul relationnel 81, 87

CASE 29, 63

Clé 1, 43, 132

artificielle 3

candidate 30, 50

composée 46

d'identification 2

de substitution 194, 197

étrangère 20, 31, 101

multidimensionnelle 138

- Clé
 primaire 4, 30, 37, 54
 Cohérence 62, 119, 167, 171
 COMMIT 185
 Compatibilité avec l'union 72
 Conception de bases de données
 9, 61
 Contrainte d'intégrité 53, 62, 100
 Copie d'archive 144
 CREATE 85, 97, 100
 Curseur
 gestion de curseur 107, 146
 notion de curseur 68, 92
 CURSOR 92
 Cycle 124
- D**
- Data mining 202, 203, 214
 Data warehouse 202
 DATE 189
 Degré d'une liaison 24
 DELETE 86
 Dépendance 41, 44, 47
 fonctionnelle 44, 47
 multivaluée 44, 50
 transitive 44, 47
 Différence 73
 Division 71, 79
 Domaine 1, 54
 Donnée
 administrateur de données 67
 analyse de données 12, 17, 57
 architecte de données 67
 architecture de données 11, 57
 enregistrement de données 2,
 135, 146
 indépendance des données 10
 intégrité des données 10, 53,
 100
 langage de définition de
 données. *Voir* Langage
 langage de manipulation de
 données. *Voir* Langage
 DROP 86
 Duplication 171
 Durabilité 120
- E**
- Ensemble d'entités 17, 30, 37, 62
 Ensemble de liens 18, 31, 62
 Entité 20, 30
 Entrepôt de données. *Voir* Data
 warehouse
 Exploration de gisements de
 données. *Voir* Data mining
- F**
- Fait 199, 210, 214
 Fermeture transitive 213, 214
 FETCH 93
 Fichier grille. *Voir* Grille
 Fonction d'appartenance 206
 Fonction prédéfinie 85
 Fonctionnement dans
 l'environnement multi-
 utilisateur 10, 118
 Forme normale 41, 63, 160, 195
 cinquième forme normale 43,
 52, 167
 deuxième forme normale 43,
 46, 194
 forme normale de Boyce-
 Codd 43, 50

Forme normale
forme normale de projection-
jointure 52
première forme normale 43,
45, 160, 163, 195
quatrième forme normale 43,
52
troisième forme normale 43,
44, 48

Fouille de données. *Voir* Data
mining

Fragment 182, 187
horizontal 183
vertical 184

G

Généralisation 25, 37, 62, 194,
196
Gestion de fichiers 146
Gestion de la mémoire tampon
146
Gestion des enregistrements 131,
146
GRANT 98
Graphe de précedence 123
Grille
fichier grille 139, 146
index grille 139
Groupe répétitif 45, 160

H

Hachage 107, 135, 146
dynamique 137
par la méthode de la division
136

I

Index 116
Indicateur 199
INSERT 86
Intégrité 10, 53, 100
référentielle 54, 100
Interblocage 124
Interface
ensembliste 145
orientée enregistrement 146
Intersection 70, 73
ISO 5, 83, 176
Isolation 119

J

Jointure 71, 77, 84, 90, 114
dépendance de jointure 52,
167
évaluation d'une jointure 114
implicite 193, 195
par boucles imbriquées 115
par tri-fusion 117
prédicat de jointure 77, 84
stratégie de jointure 106, 114
Journal 122, 128
Journal de transaction 143

L

Langage
de classification floue 205
de définition de données 9
de manipulation de données 5,
67, 86
de requête 5, 67, 81
descriptif 6

- Langage
 - ensembliste 4, 71, 93
 - immersé 92
 - naturel 7
 - procédural 6
 - relationnel complet 81
- Langage d'interrogation. *Voir*
 - Langage de requête
- Liaison 21, 33
 - complexe-complexe 34, 194
 - simple-complexe 33, 35
 - simple-simple 33
- LOCK 125

- M**

- Mémoire
 - allocation de la mémoire 147
 - gestion de la mémoire 107, 146
 - structure de stockage 107, 147
- Méthode 196
- Méthode de hachage. *Voir*
 - Hachage
- Migration 158, 164, 167
- Minimalité 3
- Modèle de données 17, 58
 - relationnel 17
- Modèle entité-association 18, 29, 158
- Modèle multi-couches 146
- Modèle relationnel 4, 41
- Modélisation des données 17

- O**

- Objet structuré 193

- Opérateur
 - binaire 110
 - de division 79
 - de jointure 71, 77, 114, 167
 - de projection 70, 75, 113, 166
 - de sélection 70, 76, 82, 85, 113
 - ensembliste 69, 71, 82
 - générique 196
 - relationnel 70, 75, 82
 - unaire 110
- Optimisation 111

- P**

- Page
 - accès aux pages de données 141
 - allocation de pages 147
- Parallélisme 121, 129, 186
- Patrimoine d'applications
 - héritées. *Voir* Système hérité
- Performance 10, 63, 194
- Point de reprise 143
- Principe
 - ACID 120
 - de récursion 214
 - de sérialisabilité 120
 - du tout ou rien 106, 119
- Privilège 98
- Procédure de redémarrage 120, 143
- Procédure de restauration 120, 143
- Produit cartésien 70, 77
- Projection 70, 75, 84, 89, 113
- Protection des données 96

- Protocole**
 de validation en deux phases 184, 187
 de verrouillage à deux phases 125
- Q**
- QBE 81, 83, 89, 90, 91
 QUEL 81, 83, 87
- R**
- READ 121, 127, 130
 Récursion 214
 Redémarrage. *Voir* Procédure de redémarrage
 Redondance 42, 171
 Règle 210, 214
 Règle de conversion 30, 33, 37, 158
 Règle de duplication 170, 171
 Relation. *Voir* Table
 Reprise sur panne. *Voir* Procédure de restauration
 Requête
 d'intervalle 141, 142
 singulière 141
 Restart. *Voir* Procédure de redémarrage
 REVOKE 98
 Rôle 35
 ROLLBACK 143
- S**
- Schéma de base de données 18, 30
 relationnelle 18, 30, 62
- Sécurité des données 96
 SELECT 5, 84
 Sélection 4, 70, 76, 85
 Sérialisabilité 120, 124, 130
 Sous-ensemble d'entités 25
 SQL 4, 83
 Stratégie de remplacement des pages 147
 Structure 26, 195
 EST UN (IS A) 26, 196
 MEMBRE DE (PART OF) 28, 195
 Structure de données
 multidimensionnelle 138, 147, 203
 physique 62, 147
 Suppression
 en cascade 56, 102
 restreinte 56, 102
 Synchronisation
 optimiste 129
 pessimiste 124
 Système d'information 17, 57, 63
 orienté web 151
 Système de bases de données
 floues 209
 multidimensionnelles 202
 post-relationnelles 181
 relationnelles 105
 relationnelles-objet 197
 réparties 187
 système à bases de connaissances 214
 temporelles 191
 Système de dictionnaire de données 67

- Système de gestion de bases de données. *Voir* Système de bases de données
 - Système expert 214
 - Système hérité 149, 168, 174
- T**
- Table 1, 4
 - dérivée 211
 - table système 8, 67
 - Temps 188
 - transactionnel 189
 - valide 188
 - TIME 189
 - Traduction 107, 108, 145
 - Traitement des erreurs 108, 142
 - Transaction 106, 118, 146
 - Transformation de clés. *Voir*
 - Hachage
 - Type d'associations 22, 62, 161
- U**
- Unicité 3, 54
 - Union 70, 72
 - UNLOCK 125
 - UPDATE 86
 - Utilisateur
 - final 69
 - occasionnel 8
- V**
- Valeur nulle 94
 - Variable linguistique 206
 - Verrou exclusif 124
- Verrouillage
 - protocole de verrouillage à deux phases 125
 - taille des unités de verrouillage 128
 - VIEW 97
 - Vue 97, 178, 212
- W**
- WRITE 121, 127, 130
- X**
- XML 153
 - XQuery 156
- Z**
- Zone de débordement 136