CHAPITRE 0

BASES DE DONNÉES RELATIONNELLES (LICENCE PROFESSIONNELLE PARIS6 - IRÈNE GUESSARIAN)

Ce polycopié s'inspire de plusieurs documents dont la lecture est recommandée :

Le Cours de S. Grigorieff en Licence d'Informatique à Paris7, et les TD de Jean-François Antoniotti, Yan Jurski, Ines Klimann, qu'ils ont bien voulu me communiquer : qu'ils en soient chaleureusement remerciés.

Le Livre de J. Ullmann, Data Base and Knowledge-base system, Computer Science Press, et ses notes de cours, http ://WWW-DB.Stanford.EDU/˜ullman. Nous recommandons vivement la visite de ce site WEB remarquablement organisé et bien documenté.

Le Livre de H. Garcia-Molina, J. Ullmann, J. Widom, Data Base systems : the complete book, où le lecteur trouvera nombre d'exercices dont ce polycopié s'inspire et toutes informations utiles sur pratiquement tous les domaines des bases de données ; nous recommandons très fortement ce livre comme ouvrage de référence. Prentice-Hall.

Le Livre de J. Ullmann, J. Widom, A first course in Data Base systems, Prentice-Hall.

Le polycopié de S. Grin, Introduction aux Bases de Données : modèle relationnel et SQL, http ://deptinfo.unice.fr/˜grin.

Pour aller plus loin, on pourra lire avec profit l'excellent livre de S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley.

CHAPITRE 1

SGDB

1.1 Introduction

Avant les bases de données (DATA BASE), chaque application écrite pour un organisme travaillait avec ses propres fichiers ; une même information, par exemple le numéro de téléphone d'un client, est alors enregistrée dans plusieurs fichiers différents. Ceci cause des délais de mise à jour, peut amener les diverses applications à travailler sur des données contradictoires, et multiplie les possibilités d'erreurs de mises à jour.

Au contraire, quand la gestion des données est “centralisée” (centralisée logiquement mais peut être répartie physiquement sur plusieurs sites), chaque donnée n'est enregistrée qu'en un seul endroit de la base, ce qui diminue les risques d'erreurs de mises à jour, et supprime le problème d'avoir des informations contradictoires sur une même donnée dans des fichiers différents. Cela facilite donc le maintien de l'intégrité des données ; la sécurité est aussi accrue grace à des fonctionnalités de récupération après pannes.

Les SGDB (Système de gestion de Bases de Données, en anglais Data Base Management System ou DBMS) évolués offrent aussi des instructions très puissantes pour traiter les données (un ordre SELECT de SQL peut correspondre à plusieurs dizaines de lignes de programme C par exemple. Enfin, les SGDB récents permettent l'interrogation de la base de données par des utilisateurs non informaticiens dans des langages non procéduraux.

1.2 Fonctionalités d'un SGDB

Un SGDB est un *ensemble de programmes qui permettent de stocker des données dans une base et de récupérer des informations de cette base.*

Un SGDB doit permettre de

- décrire les données qui seront stockées. Par exemple, on veut
  - une table donnant pour chaque étudiant : numéro de la carte d'étudiant, nom, prénom, adresse, diplome préparé.
  - une table donnant pour chaque enseignant : numéro de SS, nom, prénom, adresse, UFR de rattachement.
  - une table donnant pour chaque cours : horaire, salle, UV.
- gérer les données,
  - manipuler les données (ajouter, modifier, supprimer des informations),

- assurer la cohérence (ou intégrité) des données (contraintes de domaine, d'existence),
- assurer la confidentialité des données (mots de passe, autorisations,...),
- gérer les problèmes d'accès multiples aux données (blocages, transaction parallèles),
- prévoir des procédures de reprise en cas de panne (copies de sauvegarde, journaux).

– interroger les données au moyen de requêtes (QUERY en anglais) (selections, tris, agrégats, etc.)

– permettre l'écriture d'applications indépendantes de l'implantation physique des données (codage, supports d'enregistrement), et aussi indépendantes que possible de l'implantation logique des données (index, décomposition en "fichiers logiques").

1.2.1 Architecture d'un SGDB

On distingue 3 niveaux dans la description d'une base de données

externe : accessible aux utilisateurs de la base, et aux programmes d'applications (chaque utilisateur a une vision partielle – ou vue – de la base, et peut utiliser son propre langage de requêtes),

conceptuel : c'est le SGDB propre, qui définit la structure globale des données, intègre les différentes visions externes de la base,

interne : l'implantation de la base de données sur les ordinateurs (stockage des données sur disque, etc.).

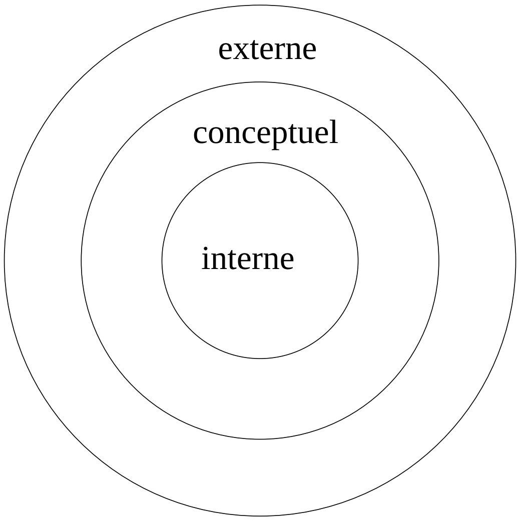

Figure 1.1 Les 3 niveaux d'une base de données

Les fonctions d'un SGBD sont

– : de faire l'interface avec le niveau interne, c'est-à-dire

- le stockage des données et meta-données sur les différents supports (disques,etc.). Les données sont l'information brute à laquelle on s'intéresse, les meta-données sont les informations sur la structuration des données : catalogue des tables, des attributs, index permettant la recherche rapide non séquentielle des données selon certains attributs. Les index sont souvent implantés sous la forme d'arbres équilibrés (toutes les feuilles à la même profondeur) et leurs nœuds internes peuvent avoir plusieurs centaines de fils.

- de gérer l'accès aux fichiers via un gestionnaire de fichiers qui connait les blocs disques où se trouvent les fichiers, et une mémoire tampon (buffer) où transitent les échanges d'information entre la mémoire centrale et les disques, et ce par secteur ou bloc entier.

– : de faire l'interface avec le niveau externe, c'est-à-dire de traiter les requêtes pour

- les optimiser en réorganisant les opérations faites dans la requête, en utilisant les éventuels index, etc.
- traduire les requêtes et les mises à jour exprimées dans divers langages de haut niveau (SQL, QBE, etc.) en une suite d'actions sur les données stockées.

En gros, l'utilisateur indique l'information qu'il souhaite obtenir et le SGDB trouve la manière d'obtenir cette information.

– : enfin d'assurer la gestion et l'organisation du niveau conceptuel :

- implanter les commandes de l'administrateur de la base de données pour décider de la structuration de la base (par exemple comment structurer les données en tables, relations, etc.)
- gérer les transactions à effectuer sur la base : une transaction est un ensemble de requêtes à exécuter séquentiellement comme une seule unité (par exemple : opération sur carte bancaire : donner les billets et enregistrer le débit). Les transactions doivent être
  - *atomiques*, c'est-à-dire s'effectuer en tout ou rien (on ne délivre pas de billets sans enregistrer le débit),
  - *consistantes*, c'est-à-dire ne pas violer les contraintes de la base de données (par exemple on ne délivre pas de billets si le compte est déjà débiteur)
  - *isolées*, c'est-à-dire le fait que plusieurs transactions puissent s'effectuer en même temps ne doit avoir aucun effet indésirable comme par exemple attribuer la dernière place disponible dans un train à deux voyageurs différents (ceci est en général assuré par des mécanismes de blocage (lock en anglais) qui garantissent l'exclusion mutuelle).
  - *durables*, c'est-à-dire résistantes aux pannes : l'effet sur la base de la transaction ne doit pas être perdu après la fin de la transaction (en général on écrit le résultat de la transaction sur les disques au fur et à mesure de leur exécution).

CHAPITRE 2

LE MODÈLE ENTITÉ–RELATION

La conception d'une base de données commence par une analyse des informations que la base doit contenir et des divers types de relations entre ces informations. On commencera donc par décrire la structure de la base par un schéma abstrait, le plus souvent sous la forme de diagrammes entité-relation, c'est-à-dire des graphiques représentant les divers types de données et la manière dont ils sont reliés. Ensuite on implémentera ce schéma : nous décrirons le modèle le plus simple et le plus utilisé pour implémenter les schémas de bases, à savoir le modèle relationnel, où toutes les données et leurs connexions sont représentées par des tables ou relations ; ce modèle est le *modèle relationnel*.

2.1 Les éléments du modèle

- les *entités*, ou objets qu'on peut identifier : étudiant, enseignant, UFR, consommateur, film, acteur, livre, etc. (à peu près tout peut être une entité...)
- les *types* ou ensembles d'entités ayant quelques similarités ou points communs : les étudiants, les enseignants, les films, les livres, etc.
- les *attributs* : des propriétés des entités, qui sont en général une application de l'ensemble d'entités dans les réels, les entiers ou les chaînes de caractères ; par exemple, nom, numéro de carte d'étudiant, diplôme préparé sont des attributs du type étudiant.

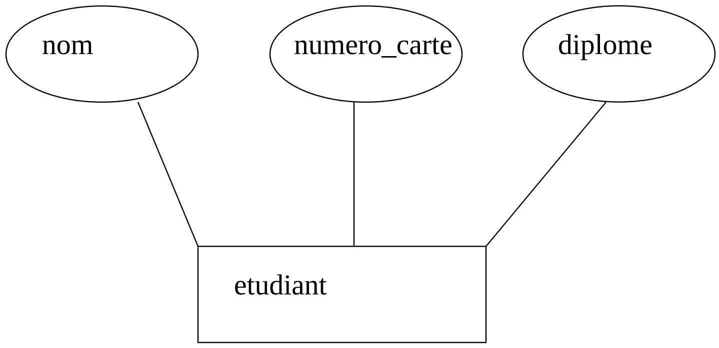

FIGURE 2.1 Un type étudiant et ses attributs

- les *relations* ou *associations* : un lien entre des types d'entités, traduit souvent ce que décrit un verbe ; par exemple l'étudiant Dupont étudie le cours de Bases de données traduit la relation “étudier” entre les types étudiants et cours.

Un attribut peut aussi être une propriété d'une relation, par exemple "prix" est un attribut de la relation vendre dans "le fournisseur Librairie du Campus vend le livre de Ullmann au prix de N euros".

- les *sous-types* : un sous-ensemble d'un type, qui comporte moins d'éléments, hérite de toutes les propriétés de son sur-type, mais peut avoir des propriétés supplémentaires (exprimées par des attributs), qui ne sont pas partagées par le sur-type ; par exemple, on peut envisager un sous-type "étudiant étranger" avec des attributs "Bourse Erasmus", "Université d'origine", un sous-type "étudiant travailleur", avec des attributs "entreprise d'origine", "statut du détachement".
- les *clés* : une clé est un attribut (ou un groupe d'attributs) dont la valeur permet d'identifier de manière unique chaque entité. Il y a une clé principale pour chaque type d'entité : par exemple le numéro de Sécurité sociale pour les individus.

La notion de clé permet de vérifier automatiquement une certaine cohérence de la base de données : par exemple le SGDB refusera d'enregistrer deux individus ayant le même numéro de Sécurité sociale. Elle permet aussi d'accélérer les recherches dans la base : à la clé principale on associera un index permettant un accès rapide aux données (par exemple par des méthodes de hash-coding).

- les *cardinalités* et *fonctionalités* des relations binaires. Une relation R entre les types d'entités E et F est dite fonctionnelle (en anglais one-many) si son graphe est celui d'une fonction. On peut donner une information plus précise en donnant la cardinalité des relations : les cardinalités sont parmi $\{(0,1)\ ,\ (1,1)\ ,\ (1,N)\ ,\ (0,N)\}$, ou N signifie "plusieurs". La relation $R\colon E \longrightarrow F$ donne à E la cardinalité :
  - $(0,1)$ si pour toute entité f dans F il y a au plus une entité e dans E telle que (e,f) soit dans R (many-one en anglais)
  - $(1,1)$ si pour toute entité f dans F il y a exactement une entité e dans E telle que (e,f) soit dans R
  - $(1,N)$ si pour toute entité f dans F il y a au moins une entité e dans E telle que (e,f) soit dans R
  - $(0,N)$ sinon

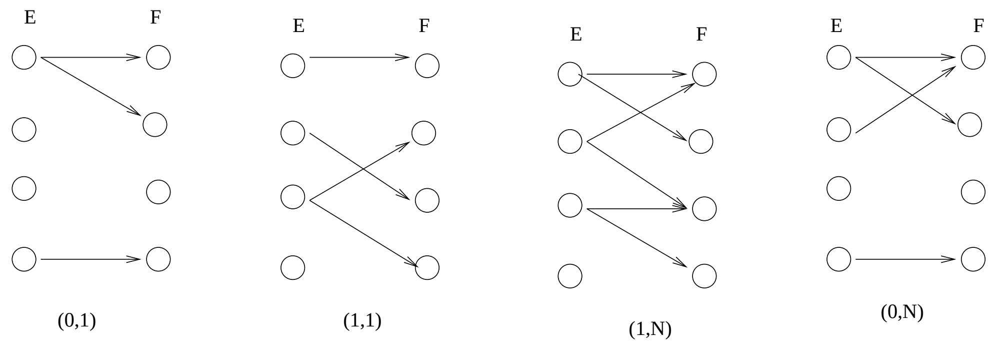

FIGURE 2.2 Cardinalité de E

La terminologie ci-dessus se généralise aux relations $R\colon E \longrightarrow F_1 \times \cdots \times F_n$: par exemple, on dira que R donne à E la cardinalité $(1,N)$ si pour toute entité $(f_1,\ldots,f_n)$ dans $F_1 \times \cdots \times F_n$ il y a au moins une entité e dans E telle que $(e,f_1,\ldots,f_n)$ soit dans R.

Enfin notons qu'il peut exister des relations non binaires : par exemple, supposons qu'il y a plusieurs groupes de TD correspondant au même cours de sociologie ; alors, la relation "l'étudiant Durand suit le TD de l'enseignant Dupont", la relation "suivre un TD" est une relation à trois attributs "étudiant, cours, enseignant_TD", que l'on peut considérer soit comme une relation étudiant$\longrightarrow$ (cours, enseignant_TD), soit comme une relation (étudiant, cours)$\longrightarrow$ enseignant_TD, etc.

Remarque 2.1 Les terminologies sur les cardinalités varient grandement : le système Merise a une terminologie totalement différente (on voit les cardinalités dans l'autre sens), nous la résumons ci-dessous : la relation $R: E \longrightarrow F$ donne à E la cardinalité :

- $(0, 1)$ si pour toute entité e dans E il y a au plus une entité f dans F telle que (e, f) soit dans R (many-one en anglais)
- $(1, 1)$ si pour toute entité e dans E il y a exactement une entité f dans F telle que (e, f) soit dans R
- $(1, N)$ si pour toute entité e dans E il y a au moins une entité f dans F telle que (e, f) soit dans R
- $(0, N)$ sinon

Dans le cas d'une relation binaire $R: E \longrightarrow F$ il suffit d'inverser les rôles de E et F, par contre dans le cas d'une relation ayant 3 arguments ou plus, les cardinalités au sens Merise n'ont rien de commun avec les cardinalités données précédemment.

La terminologie anglaise est aussi différente (et voit les cardinalités un peu dans le même sens que Merise) ; elle est résumée par le diagramme ci-dessous (cf. Ullmann) : $R: E \longrightarrow F$ est

- many-one ssi pour tout e dans E il existe au plus un f dans F tel que (e, f) soit dans R,
- one-many ssi pour tout f dans F il existe au plus un e dans E tel que (e, f) soit dans R, ou encore R^{-1} est many-one,
- one-one ssi elle est à la fois one-many et many-one,
- many-many dans les autres cas.

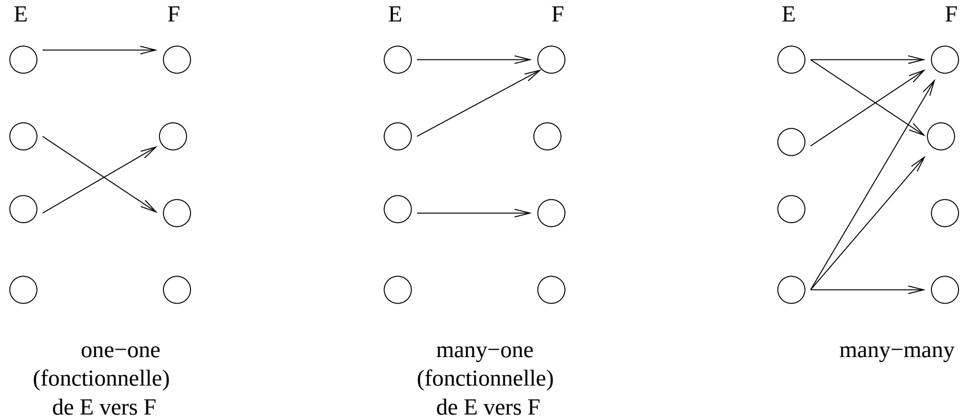

Figure 2.3 Cardinalités en anglais

Exercice 2.1 1. Vérifier qu'une relation $R: E \longrightarrow F$ est telle que $R^{-1}: F \longrightarrow E$ soit fonctionnelle ssi R donne à E les cardinalités $(0, 1)$ ou $(1, 1)$.

2. Vérifier qu'une relation $R{:}\,E \longrightarrow F$ donne à E la cardinalité $(0,1)$ ssi la relation $R^{-1}F \longrightarrow E$ est many-one. ♢

Exercice 2.2 On considère la relation à trois attributs "étudiant, cours, enseignant_TD", "suivre un TD" dont le diagramme est représenté comme suit :

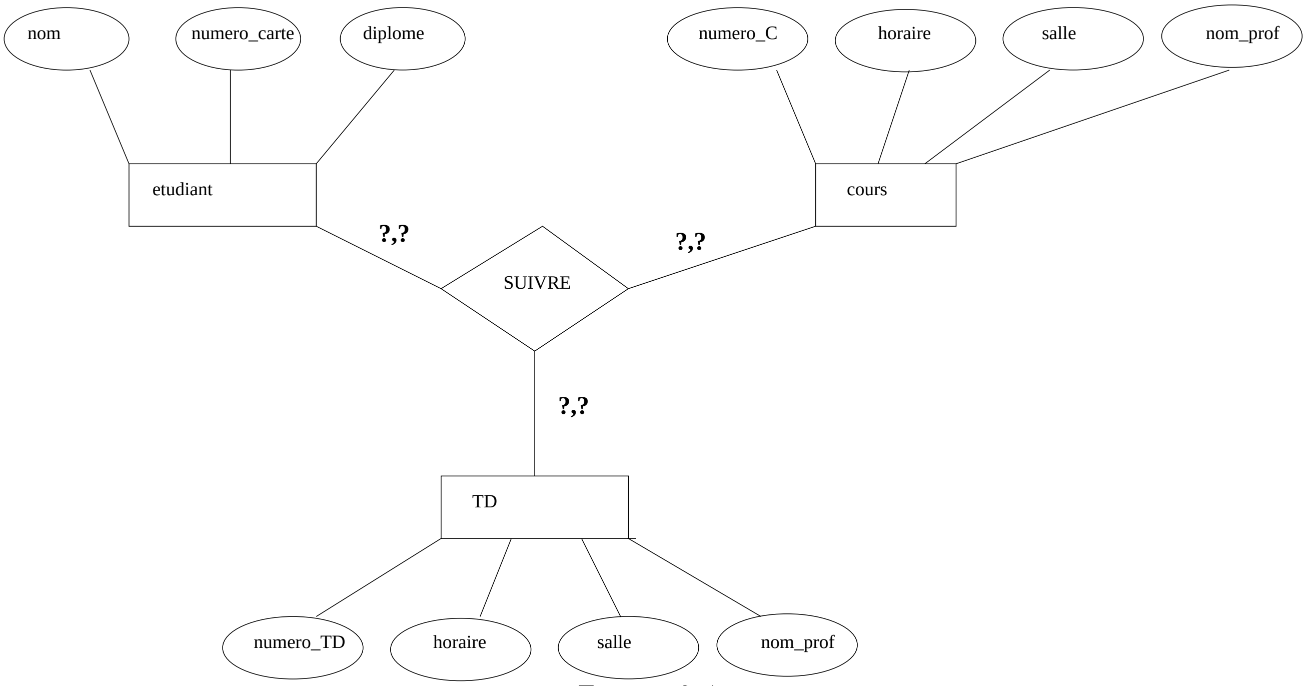

Figure 2.4

1. Quels traits peut-on remplacer par des flèches ?

2. Compléter le diagramme en remplaçant les " ?" par les cardinalités.

3. Trouver les clés principales. ♢

2.2 Représentation du modèle

La représentation du modèle se décompose en un *schéma de la base* que l'on conçoit le plus souvent sous la forme d'une représentation graphique qui donne les éléments essentiels de la structure de la base, puis de diverses *instances* de la base, qui donnent les éléments réels qui sont dans la base de données.

– les *diagrammes entité–relation.*

Les diagrammes entité–relation sont aux bases de données relationnelles ce que la notion d'organigramme est pour les programmes : ils donnent une représentation graphique synthétique de l'organisation de la base. Les conventions utilisées par Chen (l'inventeur du modèle) et Ullmann sont les suivantes :

– chaque type d'entité est représenté par un rectangle

– chaque attribut est représenté par un ovale relié au type d'entité ou bien à la relation

– chaque relation est représentée par un losange relié par un trait aux types d'entités qu'elle met en jeu

– chaque inclusion de sous-type dans un type est représentée par un triangle

- la fonctionalité d'une relation est représentée en transformant en flèche le trait la reliant au type (ou à l'attribut) vers lequel elle est fonctionnelle.
- la clé principale d'un type d'entités est représentée en soulignant les attributs qui la constituent.

Certains auteurs regroupent les attributs dans le rectangle du type d'entité, par exemple Merise ou UML (Unified Modeling Language). D'atres utilisent des ovales à la place des losanges. Nous utiliserons principalement la notation de Chen et Ullmann, mais aussi parfois les autres notations.

- les *instances* d'un schéma.

Une instance d'un schéma est la valeur "réelle" des entités et relations qui constituent le schéma. On représentera par exemple une instance d'un type d'entité par une table ayant

- une colonne pour chacun des attributs qui lui sont reliés
- une ligne pour chaque ensemble d'entités (une entité pour chacune des colonnes) qui appartiennent au type.

Exemple 2.2 Par exemple une instance correspondant au type étudiant de la figure 2.1 est représentée par la table :

étudiant

nom	*numero carte*	*diplôme*
Dupont	10598	Licence
Durant	21345	Maitrise
Martin	15967	DEUG

On représentera de même une instance d'une relation par une table ayant

- une colonne pour chacun des types d'entités qui sont reliés par la relation
- une ligne pour chaque ensemble d'entités (une entité pour chacune des colonnes) qui appartiennent à la relation.

Exemple 2.3 Par exemple une instance correspondant à la relation "suivre un cours" est représentée par la table :

suivre un cours

nom étudiant	*cours*
Dupont	Bases de données Licence
Durant	Logique Maitrise
Martin	Pascal DEUG

2.3 Exemples de modélisation d'une Base de données par des diagrammes entité-relation

Les exemples qui suivent sont tirés du polycopié de S. Grigorieff et du livre de Garcia-Molina, Ullmann et Widom. On veut représenter une base de données contenant des informations sur des films, leurs acteurs, et les studios de production. Plusieurs choix sont possibles pour les types d'entités et les relations. Le choix doit être fait en fonction des données du problème et des souhaits du commanditaire de la base, après étude faite par l'administrateur et le concepteur de la Base. Garcia-Molina, Ullmann et Widom proposent les diagrammes suivants pour les entités et les relations.

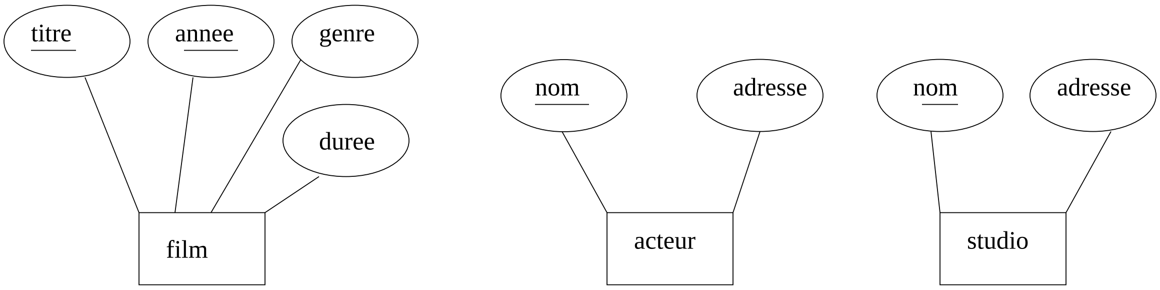

FIGURE 2.5 Les types d'entités de la base Cinema

On peut considérer diverses relations entre ces types d'entités, par exemple deux relations binaires, "jouer" et "appartient", et on suppose qu'un film donné appartient à un seul studio, que chaque studio possède au moins un film, que dans chaque film il y a au moins un acteur, et enfin qu'un acteur joue dans plusieurs (ou peut être aucun film), ce qui est modélisé par le diagramme :

FIGURE 2.6 Des relations binaires pour la base Cinema

On peut aussi considérer une relation d'arité quatre, Contrat(film, acteur, studio1,studio2), où studio1 est le studio qui "possède" l'acteur, et studio2 est le studio qui produit le film et qui fait un contrat avec studio1 pour que l'acteur puisse jouer dans son film.

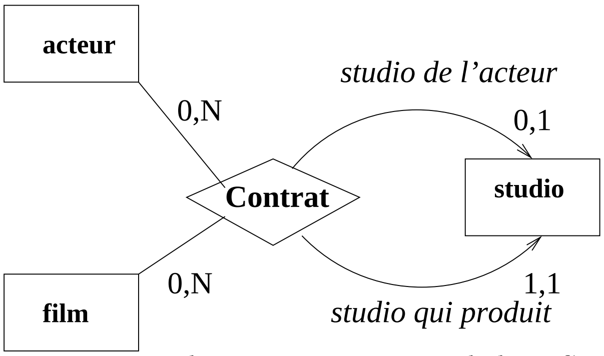

FIGURE 2.7 Une relation quaternaire pour la base Cinema

Les cardinalités sont fonctionnelles vers studio car on suppose qu'un seul studio produit un film donné, et qu'au plus un studio "possède" un acteur donné (de fait on n'utilise qu'un des 3 autres types d'entités pour savoir que la relation est fonctionnelle, mais cela ne peut pas être traduit sur le diagramme, mais seulement par la notion de dépendance fonctionnelle (voir Ullmann)). Par contre, les cardinalités de acteur et film sont $(0, N)$ car étant donnés un film, et deux studios, il peut y avoir plusieurs (ou même aucun) acteur(s) qui sont sous contrat pour jouer dans ce film. De même un acteur peut avoir plusieurs (peut-être aucun) contrats pour jouer dans plusieurs films.

Les conventions de Ullmann sont de mettre une flèche du côté de l'entité E si la relation est fonctionnelle de tous les autres attributs vers E, et un arc de cercle du côté de l'entité E si (i) la relation est fonctionnelle vers E et (ii) de plus on impose la contrainte que, étant donné un n-uplet de tous les autres attributs, il existe *au moins* un élément de E en relation avec ce n-uplet.

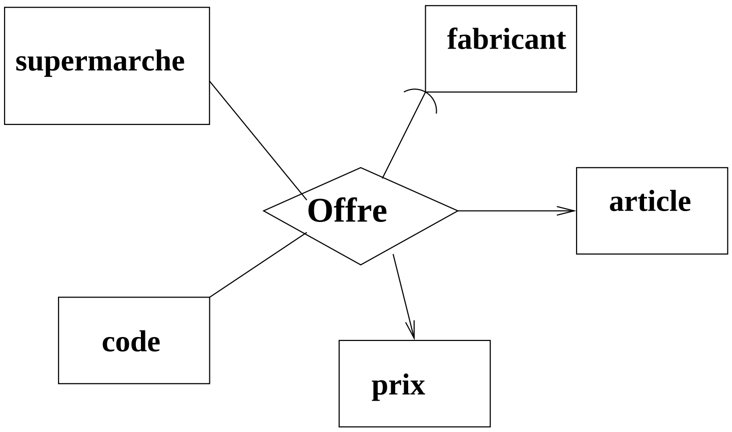

FIGURE 2.8 Fonctionalités et flèches

Il y a des flèches vers "article" et "prix" car on suppose qu'un supermarché donné a un seul article venant d'un fabricant donné et avec un code barre donné, et de même, on suppose que le prix de cet article est le même dans tout le magasin (ce n'est hélas pas toujours le cas, du moins en ce qui concerne le prix affiché, si ce n'est le prix payé). Il y a un arc de cercle du côté "fabricant" car on suppose que pour chaque article offert avec son code et son prix, il y a au moins un fabricant qui fournit l'article (sinon on supprime l'article correspondant, ce qui est une contrainte d'implantation de la base, dite *referential integrity*).

EXERCICE 2.3 Complétez les cardinalités de la figure 2.8. ♢

EXERCICE 2.4 1. A quelle cardinalité sur E correspond un arc de cercle vers E ?

2. Serait-il naturel d'avoir des arcs de cercle dans la figure 2.7 et la figure 2.6 ? Si oui lesquelles ? ♢

On peut aussi ajouter des attributs à une relation, par exemple la relation contrat de la figure 2.7 peut avoir un attribut "salaire". Enfin on peut transformer une relation d'arité supérieure ou égale à 3 en plusieurs relations binaires, en remplaçant la relation par un type d'entité du même nom, relié par des relations binaires à chacun des types d'entités qui la constituent. Par exemple, la relation contrat de la figure 2.7 (affectée d'un attribut "salaire") peut être transformée en un type d'entité contrat, relié par 4 relations binaires à acteur, film et studio : pour chaque quadruplet (film, acteur, studio1,studio2) de la relation Contrat, le type d'entité contrat contiendra une entité e, qui sera en relation avec acteur par la relation acteur_du, etc.

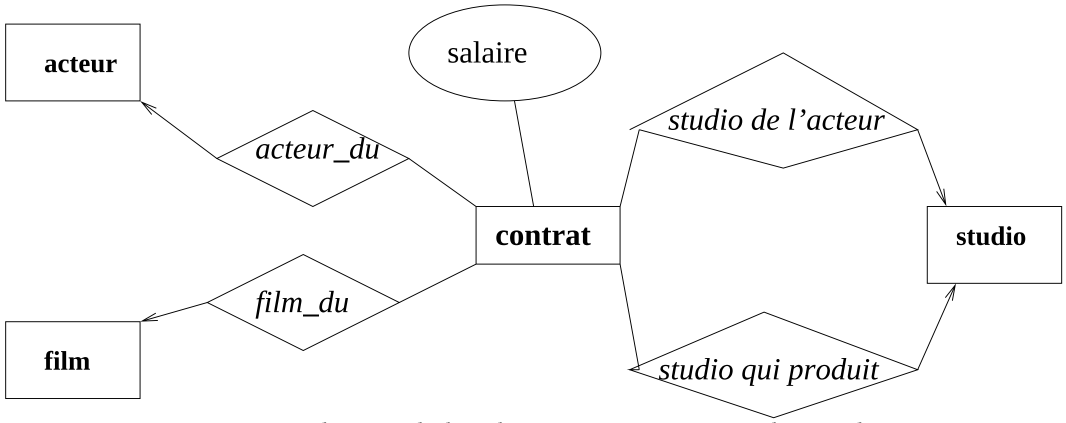

FIGURE 2.9 Traduction de la relation quaternaire en relations binaires

EXERCICE 2.5

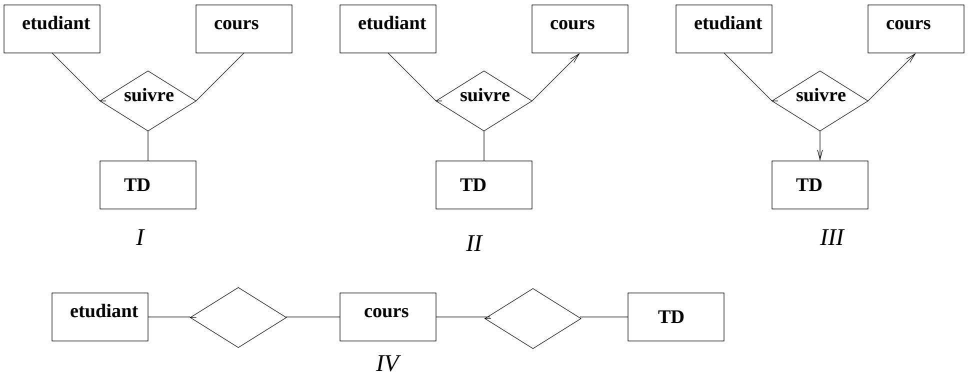

FIGURE 2.10

1. Expliquer la différence entre les diagrammes I et II.
2. Quelle contrainte ajoute le diagramme III ?
3. Quelle est la limitation du diagramme IV par rapport aux autres ? ♢

EXERCICE 2.6 Supposons une société immobilière dont l'activité consiste à louer des locaux commerciaux (stand de galerie commerciale, salle de réception, ...). La représentation du diagramme suit les conventions UML.

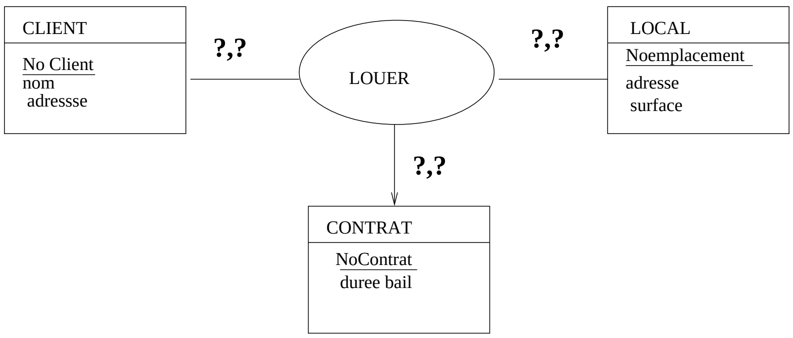

FIGURE 2.11

Remplacez les points d'interrogation par des cardinalités. ♢

EXERCICE 2.7 On suppose avoir une relation R reliant deux types d'entités $E1$ et $E2$; on suppose que R a un unique attribut A. Sous quelle condition A peut-il devenir un attribut de $E1$ sans changer la conception de la base ? ♢

2.4 Type d'entités faible ou dépendant

Parfois, la clé d'un type d'entités E peut provenir (partiellement ou totalement) non pas de ses atttributs propres, mais des clés d'autres types d'entités $F_1, \ldots, F_n$ auxquels E est relié par des relations binaires *fonctionnelles* $R_1, \ldots, R_n$. On dit alors que E est un *type d'entité faible*, et graphiquement, on met un double rectangle autour de E et un double losange autour de chacune des relations $R_1, \ldots, R_n$ qui relie E à un type contenant un clé qui est une clé du type faible E. Il est essentiel que :

- chaque R_i soit fonctionnelle de E vers F_i, ce qui permet de connaitre de façon déterministe l'unique f_i dans F_i associé à un élément e de E (R_i doit donc être many-one de E vers F_i, ou bien donner à F_i les cardinalités (0,1) ou (1,1)
- chaque entité de E soit effectivement reliée à une entité de F_i, c'est-à-dire que R_i doit donc donner à F_i la cardinalité (1,1) (ou encore on peut transformer la flèche en arc de cercle)
- les attributs de F_i qui servent de clé pour E sont les attributs qui forment la clé de F_i

Il peut se faire que l'un des F_i soit lui aussi un type d'entité faible.

Les deux sources principales d'entités faibles sont

- les sous-types : par exemple (cf. Garcia-Molina, Ullmann et Widom), si on considère que chaque studio a des équipes avec des numéros, la clé d'une équipe sera formée de son numéro et du nom du studio auquel elle appartient, voir la figure 2.12.
- les transformations de relations d'arité supérieure à deux en un type d'entité et plusieurs relations binaires. Le type d'entité ainsi introduit artificiellement hérite alors sa clé des attributs des anciens types. Par exemple, le type d'entité "contrat" introduit dans la figure 2.9 pour traduire la relation binaire de la figure 2.7, peut être précisé dans la figure 2.13 ci-dessous.

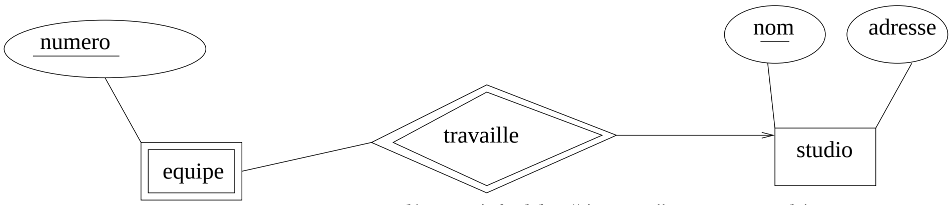

FIGURE 2.12 Le type d'entité faible "équipe" avec ses clés

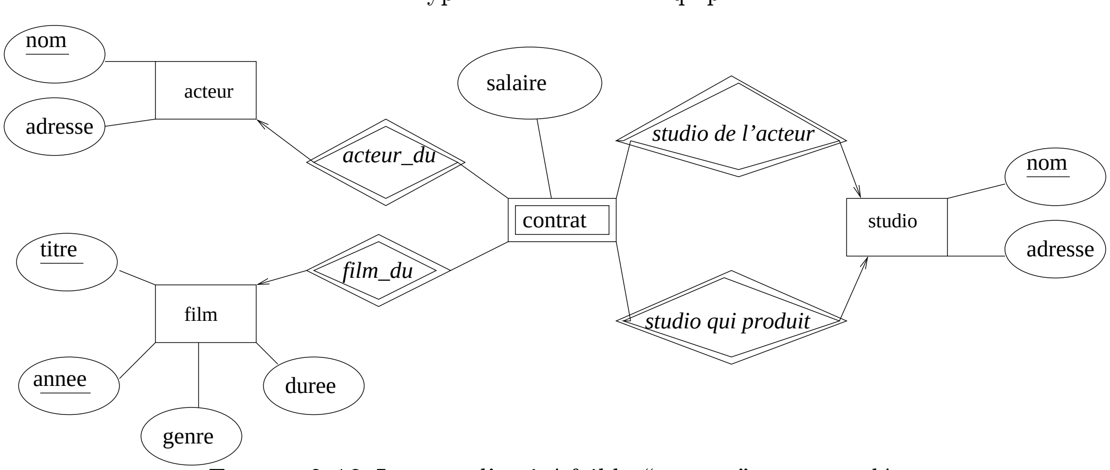

FIGURE 2.13 Le type d'entité faible "contrat" avec ses clés

EXERCICE 2.8 Dessiner un diagramme entité-relation pour une base de donnée des étudiants athlètes de haut niveau de paris6. Il faudra inclure les informations suivantes : pour chaque étudiant le nom, le numéro de la carte d'étudiant, le diplôme préparé, pour chaque cours (UFR, numéro du cours). Il faudra aussi pouvoir déterminer quel sport pratique l'étudiant une saison donnée, quels cours il suit, ses notes en cours, quels professeurs enseignent quels cours. On soulignera les clés de chaque type d'entité, on indiquera les relations fonctionnelles par des flèches, et les types d'entité faibles seront indiqués par un double encadré.

On supposera que

- un étudiant athlète de haut niveau peut suivre autant de cours qu'il veut
- un étudiant athlète de haut niveau peut suivre chaque cours autant de fois qu'il veut (jusqu'à ce qu'il obtienne le module)
- chaque professeur peut enseigner plusieurs cours chaque semestre
- un étudiant athlète de haut niveau ne peut peut pratiquer que un seul sport par saison (pour lui laisser le temps de travailler ses cours). ♢

2.4.1 Quelques heuristiques et conseils de conception de bases de données

- éviter d'introduire une relation là où un attribut suffirait ; par exemple on pourrait être tenté de définir une relation "enseigne" : le professeur Dupond enseigne le cours d'Archi, alors qu'il suffit d'ajouter un attribut "professeur" au type d'entité cours (qui est alors un type avec deux attributs, le nom du cours et le professeur qui enseigne le cours).
- Pour décider si une entité va devenir un type d'entité ou être un simple attribut, on peut donner les heuristiques suivantes : un type d'entité doit être plus qu'un simple nom, par exemple avoir des attributs qui ne sont pas des clés, ou être relié par plusieurs relations à divers types d'entités, ou bien être du côté non fonctionnel d'une relation fonctionnelle. Si par contre une entité intervient dans une seule relation, et est du côté fonctionnel pour cette relation, alors il est plus judicieux d'en faire un attribut des autres composantes de la relation.
- éviter dans la mesure du possible d'utiliser les types d'entités faibles : on essaiera de ne créer des types d'entités faibles que lorsqu'il est impossible de leur créer une clé propre, ou bien s'il n'y a pas un administrateur global qui puisse créer ces clés propres.
- si on souhaite imposer une contrainte, il faut l'imposer dès la conception du schéma. Par exemple si une relation R doit être fonctionnelle de F vers E, on interdira la possiblité d'ajouter dans R un couple $(f, e2)$ dès l'instant où il y a déjà un couple $(f, e1)$ dans R : il suffira pour ce faire de décider que la clé de F est aussi la clé pour la table de R.
- éviter les redondances (cf. exemple 3.3).

EXEMPLE 2.4 On considèrera dans la suite l'exemple suivant, emprunté au cours de S. Grigorieff et au livre de Ullman, et qui modélise un supermarché.

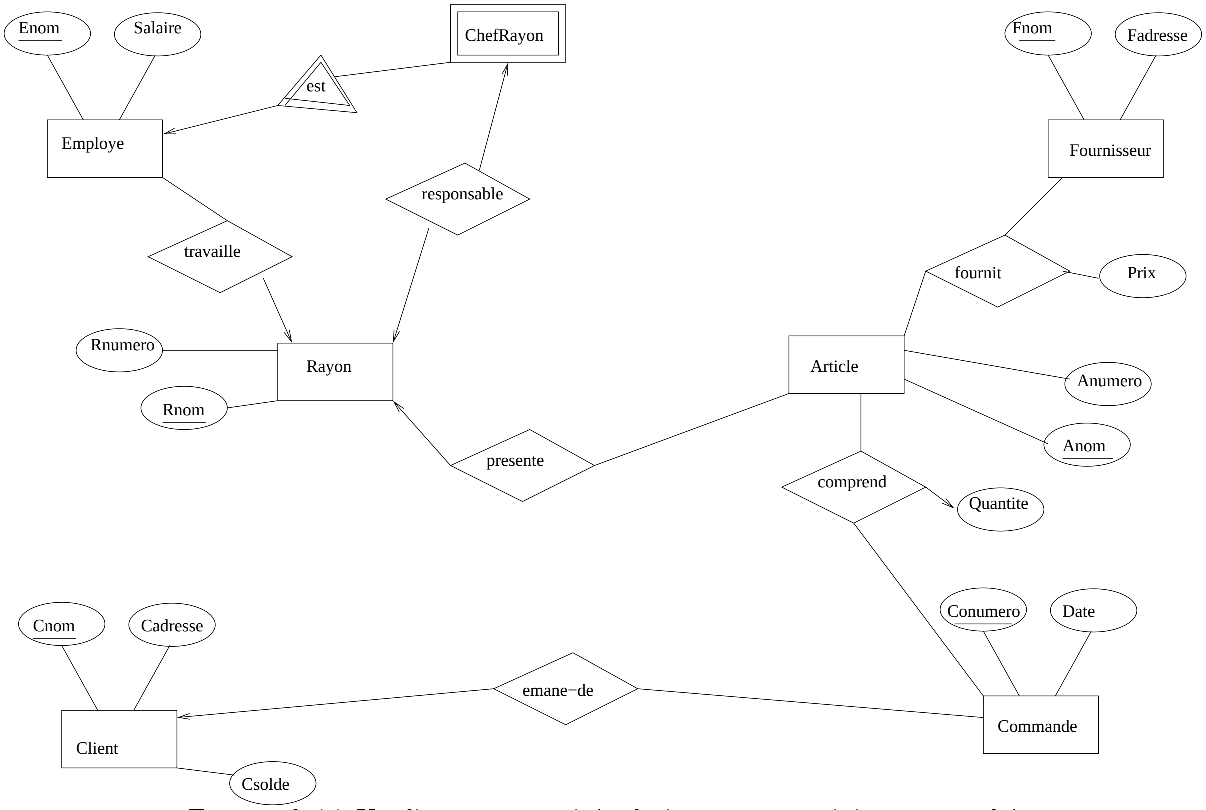

FIGURE 2.14 Un diagramme entité-relation pour un mini-supermarché

EXERCICE 2.9 Une centrale d'achats commune à tous les hypermarchés de la marque X dispose de plusieurs entrepôts dans lesquels sont stockés des articles. Chaque hypermarché passe ses commandes auprès de la centrale en indiquant le ou les entrepôts et le ou les articles concernés.

1. Proposez une modélisation de ce système.

2. Changez la modélisation du système en supposant, de plus, que chaque entrepôt dessert une zone géographique (ces zones forment une partition du territoire) et que les hypermarchés doivent nécessairement commander à l'entrepôt de la zone à laquelle ils appartiennent (toujours par l'intermédiaire de la centrale d'achats).

3. Changez la modélisation du système en supposant que chaque entrepôt dessert une zone géographique, et que, de plus, chaque entrepôt est spécialisé (produits frais, textiles, droguerie) ♢

EXERCICE 2.10 (Ullmann) Modéliser une Base de données pour une banque : les clients ont des comptes de divers types (épargne, courant) et on veut connaître le solde des comptes. Les clients ont un numéro de SS, une adresse, un téléphone et on veut savoir à qui appartient un compte.

1. Proposez une modélisation de ce système.

2. Changez la modélisation du système en supposant, de plus, que

- chaque compte a au plus un client
- chaque client a au plus un compte
- un client peut avoir plusieurs adresses (attention, un attribut ne peut pas contenir un ensemble d'entités, seulement une seule entité).
- un client peut avoir plusieurs téléphones à une même adresse . On étudiera d'abord le cas des téléphones fixes, puis le cas où on permet des mobiles. ♢

CHAPITRE 3

LE MODÈLE RELATIONNEL

Proposé par Codd en 1970, il est maintenant quasi-universellement adopté grace à sa simplicité et il est à la base du langage de requêtes SQL (toutefois SQL utilise les “bags” ou multi-ensembles, alors que le modèle relationnel utilise les ensembles). Les buts sont

- indépendance des programmes d’application et des sessions interactives par rapport à la représentation interne des données (en particulier de l’ordre d’implantation des données dans les fichiers).
- Base théorique solide (logique et calcul ensembliste) pour traiter des problèmes de cohérence et redondance des données (qui sont traités par de multiples pointeurs dans les autres approches).

Dans le modèle relationnel, l’utilisateur voit toutes les relations comme des tables bidimensionnelles, les noms de colonnes sont les attributs et les lignes sont les n-uplets de la relation.

3.1 Généralités

3.1.1 Les tables du modèle relationnel

L’utilisateur perçoit le modèle comme des *tables* bidimensionnelles et rien d’autre. chaque tables consiste en

- un *schéma relationnel* ou *schéma de relation* : un nom pour la table, un nom et un type (appelés attribut et domaine) pour chaque colonne de la table. Des tables distinctes ont toujours des noms différents. Les noms des colonnes doivent être deux à deux distincts, mais deux colonnes peuvent avoir le même type. Un même nom de colonne peut apparaître dans plusieurs tables. Les types sont des scalaires (non structurés) : chaîne de caractères (de longueur fixe ou variable mais bornée), nombre, date, heure, avec éventuellement des contraintes (être un nombre compris entre m et M par exemple).
- des *contraintes de fonctionnalité* : un attribut A (ou en ensemble E d’attributs) est une *clé* de la table si deux lignes distinctes doivent différer sur A (ou sur au moins un des attributs de E) ; chaque ligne est donc uniquement (fonctionnellement) déterminée par sa restriction sur les attributs de la clé. Si la clé est formée par un groupe d’attributs E, aucun sous-ensemble strict de E ne doit pouvoir être une clé (E doit donc être minimal). Parmi les clés possibles, on en distingue une qui est appelée la *clé principale*

ou *primaire* (en anglais *primary key*), les autres clés sont appelées les *clés candidates* (en anglais *candidate keys*). Le choix de la clé est un décision du concepteur de la base et est imposé à l'utilisateur. Dans le cas d'entités faibles, on peut avoir plusieurs clés dites *clés étrangères*, qui font intervenir d'autres tables (dont elles sont les clés primaires).

- un *contenu* ou *instance du schéma* : des lignes dont chaque élément respecte le type de la colonne auquel il appartient. Dans une table donnée, deux lignes distinctes n'ont jamais le même contenu, par contre deux colonnes distinctes peuvent fort bien être identiques. (Ceci n'est pas valable pour SQL qui peut engendrer des tables comportant des lignes identiques).
- l'ordre des lignes et des colonnes sont ignorés, car ils ne traduisent que l'apparence de la table. Comme il n'y a pas d'ordre entre les attributs, l'ordre des colonnes n'apporte aucune information sur les données, et donc on ne peut pas l'utiliser (on ne peut donc référencer une colonne que par son nom et pas par son numéro de colonne). De même, l'ordre des lignes traduit au maximum l'évolution temporelle de la base, qui n'a pas à être connue de l'utilisateur (si ce n'est pas le cas il faut rajouter explicitement un attribut d'estampillage "date de création ou de mise à jour"). Il n'y a aucun moyen de référencer directement une ligne, si ce n'est en utilisant des opérateurs qui recherchent les sous-tables formées des lignes ayant un certain *contenu* (lesquelles peuvent éventuellement se réduire à une seule ligne). Certains SGDB offrent la possiblité de trier une table, mais cette possibilité ne fait pas partie du modèle relationnel et est une sur-couche ajoutée.
- l'utilisateur dispose d'opérateurs qui engendrent de nouvelles tables à partir des anciennes.

EXEMPLE 3.1 Une table pour la relation Contrat de la figure 2.7. Par exemple une instance correspondant à la figure 2.7 est représentée par la table Contrat :

film	*acteur*	*studiofilm*	*studioacteur*
Whispers	Vanessa	Pathé	Universal
The end	Grace	Universal	Pathé
East-West	Sharon	United Artists	MetroGoldwynMayer
Modern times	Charlie	United Artists	United Artists

3.1.2 Modélisation des tables du modèle relationnel par des relations.

Soit T est une table ayant n attributs $A1, A2, \ldots, An$, ou en d'autres termes n colonnes de noms $A1, A2, \ldots, An$ dont les domaines associés sont les types $D1, D2, \ldots, Dn$. On associe à T une relation R qui est la partie du produit cartésien $D1 \times D2 \times \cdots \times Dn$ constituée des n-uplets $(d1, d2, \ldots, dn)$ tels que T contienne une ligne dont les éléments des colonnes sont exactement $(d1, d2, \ldots, dn)$. La relation R est donc constituée des listes $(d1, d2, \ldots, dn)$ de longueur n qui sont des lignes de la table T. En d'autres termes, les lignes de la table sont les n-uplets de la relation, et les colonnes de la table sont les projections de la relation R sur les domaines associés aux attributs. Un groupe d'attributs E est une clé si et seulement si la relation associée R est fonctionnelle en ses composantes correspondant aux attributs de E, c'est-à-dire que la donnée des attributs de E détermine un unique n-uplet de R.

La représentation d'une table par une relation rend compte de la contrainte que l'ordre sur les lignes est irrelevant : en effet, il n'y a pas d'ordre sur les n-uplets d'une relation.

Cependant, cette représentation ne rend pas compte de l'indépendance par rapport à l'ordre des colonnes, en effet, la relation R contient implicitement un ordre sur ses composantes qui est l'ordre $D1, D2, \ldots, Dn$: le produit cartésien n'est pas commutatif ($A \times B$ est différent de $B \times A$, même si $A = B$ ...).

3.1.3 Modélisation des tables du modèle relationnel par des fonctions.

On peut contourner cette difficulté en modélisant une table non plus comme un ensemble de n-uplets, mais comme un ensemble de fonctions de domaine $\{A1, A2, \ldots, An\}$, qui envoient chacun des Ai sur une entité du domaine associé Di. Rappelons que dans une table, les noms d'attributs sont deux à deux distincts même si les domaines associés sont identiques (par exemple, dans l'exemple 3.1, les attributs *studiofilm* et *studioacteur* ont le même type d'entité comme domaine associé).

On associe à T l'ensemble F des fonctions f de domaine $\{A1, A2, \ldots, An\}$, qui envoient l'attribut Ai dans son domaine associé Di, les lignes de la table sont les n-uplets $\{f(A1), f(A2), \ldots, f(An)\}$ pour f variant dans F, et la colonne de la table d'attribut Ai est formée par l'ensemble des valeurs prises sur l'attribut Ai par les fonctions de F.

Dans cette formalisation, E est une clé pour T si l'ensemble F de fonctions associées à T satisfait la condition suivante : deux fonctions distinctes de F doivent différer sur au moins un attribut de E.

Cette formalisation rend compte des contraintes d'indépendance de l'ordre des lignes et de l'ordre des colonnes. On utilisera indifféremment ces deux formalisations. Toutefois, la formalisation la plus utilisée est la formalisation par des relations.

3.2 Traduction d'un diagramme entité-relation en tables du modèle relationnel.

On peut donner une traduction automatique très simple, que l'on améliorera ensuite. Pour approfondir l'optimisation de cette traduction, que l'on n'abordera pas dans ce cours, on a besoin de la théorie de la normalisation et des dépendances fonctionnelles.

- Un type d'entité non faible est représenté par la relation de schéma relationnel suivant :
  - les attributs de la relation sont les noms des attributs du type d'entités dans le diagramme entité-relation
  - le domaine d'un attribut de la relation est le type associé à cet attribut dans le diagramme entité-relation

    par exemple le type client de la figure 2.14 est traduit par la relation Client(Cnom,Cadresse,Csolde) dont les domaines sont l'ensemble des chaînes de caractères pour les deux premières composantes et l'ensemble des nombres (réels avec 2 chiffres après la virgule) pour la troisième composante.
- Un sous-type E d'un type F est aussi traduit par une relation dont les attributs (et domaines) sont ceux spécifiques à F, auxquels on ajoute ceux de la clé principale de E. Par exemple, le type “employé” et son sous-type “ChefRayon” de la figure 2.14 sont traduits par les relations Employe(Enom,salaire) et ChefRayon(Enom).

- Un type d'entité faible est représenté par la relation dont le schéma est formé des attributs (et domaines associés) de ce type d'entité, auxquels on ajoute ceux des attributs des types d'entités auxquels il est relié et qui constituent sa clé. Par exemple, le type "Equipe" de la figure 2.12 est traduit par le schéma $equipe(numero, nom)$ où nom est l'attribut clé de type *studio*.
- Une relation entre les types $E1, \ldots, En$ est traduite par la relation dont les attributs (et domaines associés) sont ceux des clés principales des types $E1, \ldots, En$, ceci après renommage éventuel des attributs pour que tous ces attributs aient des noms différents. Par exemple, la relation contrat de la figure 2.9 est traduite par le schéma contrat(nom.acteur , titre.film , annee.film , nom.studio.acteur , nom.studio.qui.produit). Une *clé* F pour la relation R est un (ensemble d')attribut(s) F qui
  1. détermine tous les autres attributs de R
  2. est telle que aucun sous-ensemble strict de F n'a la propriété 1

     (si la propriété 2. n'est pas vérifiée, on parle de surclé).

Par exemple, dans la figure 2.14 l'attribut "Conumero" est une clé de la relation "emane-de" car cette relation est fonctionnelle de Commande vers Client.

Exemple 3.2 En utilisant cette méthode naïve de traduction, le diagramme de la figure 2.14 se traduit par les 13 relations suivantes, dans lesquelles les attributs de la clé primaire sont en gras, les attributs des clés candidates sont en italique, et les autres attributs en caractères romains.

1. Employe(**Enom**,salaire)
2. ChefRayon(**Enom**)
3. Rayon(**Rnom**,*Rnumero*)
4. Fournisseur(**Fnom**, Fadresse)
5. Article(**Anom**,*Anumero*)
6. Commande(**Conumero**,date)
7. Client(**Cnom**, Cadresse,Csolde)
8. travaille(**Enom**,Rnom)
9. responsable(**Enom**,*Rnom*)
10. presente(**Anom**,Rnom)
11. fournit(**Fnom**,**Anom**,prix)
12. comprend(**Conumero**,**Anom**,Quantite)
13. emane-de(**Conumero**, Cnom)

Exercice 3.1 On suppose avoir une relation R ternaire reliant trois types d'entités $E1$, $E2$, $E3$, qui ont chacune un attribut qui est une clé, respectivement $A1$, $A2$, $A3$. La relation R n'a pas d'attribut.

1. Dessiner les huit diagramme entité-relation correspondant possibles lorsqu'on met des flèches dans les diagrammes.

2. Pour chaque diagramme obtenu à la question 1., écrire une relation pour R où on soulignera les attributs qui forment une clé de R.

3. L'un des diagrammes de la question 1. a des flèches vers $E1$ et $E2$ et pas de flèche vers $E3$. Spécifier une application réelle où ce diagramme est un bon modèle de la situation. Il y aura trois entités et une relation ternaire qui devra être fonctionnelle vers $E1$ et $E2$ et pas vers $E3$. Les entités et la relation peuvent avoir autant d'attributs que nécessaire. ◇

EXERCICE 3.2 Traduire le diagramme de l'exercice 2.8 en relations en utilisant la méthode précédente. ◇

EXERCICE 3.3 On trouve dans un SGBD relationnel les relations ci-dessus. Les clefs primaires sont en gras, les clefs étrangères ne sont pas indiquées.

IMMEUBLE(**Adresse**, NbEtages, DateConstruction, Syndic)

APPART(**Adresse, Numero**, Code.Type, Superficie, Etage)

PERSONNE(**Nom**, Age, Code.Profession)

OCCUPE(**Adresse, Numero.Appart, Nom**, DateArrivee, DateDepart)

POSSEDE(**Adresse, Nom, Numero.Appart**, QuotePart)

TYPEAPPART(**Code**, Libelle)

PROFESSION(**Code**, Libelle)

1. Identifier les clefs étrangères dans chaque relation.
2. Construire un schéma entité-relation correspondant. ◇

3.2.1 Quelques Heuristiques pour améliorer la traduction d'un diagramme entité-relation en tables du modèle relationnel.

On essaiera de supprimer les informations redondantes.

EXEMPLE 3.3 Par exemple, dans le cas des types d'entités faibles, les relations qui supportent le type faible (entourées d'un double losange) sont redondantes. Dans la figure 2.12, par exemple, la relation equipe(numero, nom) correspondant au type d'entité "equipe" , et la relation travaille(numero, nom) correspondant à la relation faible "travaille" ont exactement les mêmes attributs, et correspondent au même schéma de relation.

On essaiera de simplifier au maximum la base de données. Par exemple, *si deux tables ont la même clé, il sera préférable de les joindre en une seule table* dont les attributs sont la réunion des attributs des deux tables de départ et qui est calculable à partir des deux tables de départ. On peut retrouver les deux tables de départ par projection de la nouvelle table. Par définition de la clé, chacune des lignes de ces deux tables se retrouve dans une et une seule ligne de la table obtenue par jointure. Cette jointure permet donc de gagner de la place mémoire et elle simplifie aussi l'interrogation de la base.

EXEMPLE 3.4 Dans l'exemple 3.2, on remarque que

- les tables Employe(**Enom**,salaire) et travaille(**Enom**,Rnom) ont l'attribut Enom comme clé principale commune, leur jointure donne une nouvelle table Employetravaille(**Enom**,salaire,Rnom).
- les tables Rayon(**Rnom**,*Rnumero*) et responsable(**Enom**,*Rnom*) ont l'attribut Rnom comme clé commune, leur jointure donne la nouvelle table Rayonresponsable(**Rnom**, *Rnumero*,chefRnom) en changeant le nom de l'attribut Enom en chefRnom.

- les tables Article(**Anom**,*Anumero*) et presente(**Anom**,Rnom) ont l'attribut Rnom comme clé principale commune, leur jointure donne la nouvelle table Articlepresente(**Anom**,*Anumero*,Rnom).
- les tables Commande(**Conumero**,date) et emane-de(**Conumero**, Cnom) ont l'attribut Conumero comme clé principale commune, leur jointure donne la nouvelle table Commandeemane-de(**Conumero**,date,Cnom).
- Il n'est pas souhaitable de joindre les tables Fournisseur(**Fnom**, Fadresse) et fournit(**Fnom**,**Anom**,prix) : la clé Fnom de la première est incluse dans la clé { Fnom , Anom } de la seconde mais ces deux clés sont différentes. Joindre ces deux tables revient à répéter l'adresse du fournisseur pour chaque article qu'il fournit, ce qui conduit à occuper inutilement de l'espace mémoire et à ralentir la consultation des données.

Ces simplifications donnent un nouvel ensemble de 9 tables (au lieu de 13)

1. Employetravaille(**Enom**,salaire,Rnom) (jointure ex 1 et 8)
2. ChefRayon(**Enom**)
3. Rayonresponsable(**Rnom**,*Rnumero*,chefRnom) (ex 3 et 9)
4. Fournisseur(**Fnom**, Fadresse)
5. Articlepresente(**Anom**,*Anumero*,Rnom) (ex 5 et 10)
6. Commandeemane-de(**Conumero**,date,Cnom) (ex 6 et 13)
7. Client(**Cnom**, Cadresse,Csolde)
8. fournit(**Fnom**,**Anom**,prix)
9. comprend(**Conumero**,**Anom**,Quantite)

On remarque finalement que la table ChefRayon(**Enom**) n'est autre que la projection de la table Rayonresponsable(**Rnom**,*Rnumero*,chefRnom) sur son troisième attribut ; on peut donc supprimer la table ChefRayon(**Enom**), et la base de données se réduit à 8 tables au lieu de 13.

Remarque 3.5 Problème des n-uplets incomplets, ou nuls (en anglais dangling tuples ou null values). Lorsque deux tables T et T' ont une clé commune F, il peut se faire que la restriction à F d'une ligne de T ne soit égale à aucune des restrictions à F des lignes de T'. Quand on joint ce deux tables, pour ne pas perdre d'information, il faut admettre dans la jointure la présence de lignes incomplètes correspondant à ces lignes de T (resp. T') dont la restriction à la clé ne se retrouve pas dans T' (resp. T) : ce sont des n-uplets incomplets, dont on marque les valeurs manquantes par le symbole $\perp$, appelé nul (en anglais "bottom" ou "null"). On peut aussi imposer à la base de données une contrainte interdisant la présence de tels n-uplets incomplets. Par exemple, supposons avoir les tables

Rayon

Rnom	*Rnumero*
textiles	t221
liquides	l332

responsable

Enom	*Rnom*
Jennifer	textiles

leur jointure sera la table (avec des nuls)

Rayonresponsable

Rnom	*Rnumero*	*ChefRnom*
textiles	t221	Jennifer
liquides	l332	$\perp$

3.3 Les opérations de l'algèbre relationnelle

On notera $R(A_1, \ldots, A_n)$ une relation R dont les attributs sont nommés $A_1, \ldots, A_n$.

3.3.1 Réunion, Intersection, Différence

Soient R et R' deux relations qui *ont le même schéma*, c'est-à-dire les mêmes attributs, alors :

- la réunion de R et R', notée $R \cup R'$, s'obtient en mettant ensemble tous les n-uplets de R et R' ; c'est donc une opération qui ajoute des lignes, à l'opposé des opérations d'intersection et de différence ci-dessous.
- l'intersection de R et R', notée $R \cap R'$, s'obtient en ne retenant que les n-uplets communs à R et R'.
- la différence de R et R' , notée $R \setminus R'$, s'obtient en ne retenant que les n-uplets qui sont dans R et pas dans R'.

3.3.2 Produit cartésien

Le produit cartésien R et R', notée $R \times R'$, s'obtient en prenant chaque n-uplet de R et en le couplant avec chaque n'-uplet de R'. C'est donc une opération qui rajoute des colonnes et multiplie les lignes : si R a n colonnes et l lignes et si R' a n' colonnes et l' lignes, alors $R \times R'$ aura $n + n'$ colonnes et $l \times l'$ lignes. Le schéma de la relation $R \times R'$ s'obtient en mettant les attributs de R, puis les attributs de R' ; s'il y a un attribut commun A , on le renomme en deux attributs $A.R$ et $A.R'$ dans le produit. Le produit est peu utilisé car il fait exploser la taille des données.

3.3.3 Projection

La projection utilise un paramètre qui est un sous-ensemble S de l'ensemble$A$ des attributs de R, il s'agit donc en fait d'une famille d'opérations π_S, pour $S \subset A$. $\pi_S(R)$ s'obtient en oubliant dans chaque n-uplet de R les attributs qui ne sont pas dans S ; c'est donc une opération qui efface des colonnes et diminue éventuellement le nombre de lignes.

En SQL la projection s'opère par la commande

SELECT (liste d'attributs) FROM table.

Remarque 3.6 Si R a n attributs, il y aura 2^n projections différentes. La projection correspondant à la partie pleine ($S = A$) est l'identité, $\pi_A(R) = R$. La projection correspondant à la partie vide sera par convention définie comme étant la relation vide (la relation sans attribut et qui ne comporte aucun 0-uplet).

Exercice 3.4 Étant données les relations R et S suivantes :

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

S	A	B	C
	b	g	a
	d	a	f

Calculer l'union $R \cup S$, la différence ensembliste $R \setminus S$, le produit cartésien $R \times S$, la projection $\pi_{A,C}(R)$ et la sélection $\sigma_{B=b}(R)$: ♢

3.3.4 Quotient

Soient $R(A_1,\ldots,A_n,B_1,\ldots,B_m)$ et $R'(B_1,\ldots,B_m)$ deux relations. Le quotient $R \div R'$ a pour attributs $A_1,\ldots,A_n$ et est formé des n-uplets $(a_1,\ldots,a_n)$ tels que pour tout m-uplet $(b_1,\ldots,b_m)$ de R', le $(n+m)$-uplet $(a_1,\ldots,a_n,b_1,\ldots,b_m)$ est dans R. On voit que

$$R \div R' = \pi_{A_1,\ldots,A_n}(R) \setminus \pi_{A_1,\ldots,A_n}\Big(\big(\pi_{A_1,\ldots,A_n}(R) \times R'\big) \setminus R\Big)$$

La terminologie est par analogie avec la division euclidienne : $R \div R'$ est le plus grand sous-ensemble T de $\pi_{A_1,\ldots,A_n}(R)$ tel que $T \times R'$ soit contenu dans R.

Exercice 3.5 Soient R et S les relations suivantes :

R A	B	C	D
a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
e	d	e	f
a	b	d	e

S C	D
c	d
e	f

Calculer le quotient $R \div S$. ♢

3.3.5 Sélection

La sélection est aussi une opération qui enlève des lignes, et comme la projection, il s'agit d'une famille d'opérations. Soit C une formule construite à l'aide de conjonctions. disjonctions et négations à partir des formules atomiques $t * u$, où t, u sont des attributs de R ou des constantes, et $*$ est un opérande pris dans l'ensemble $\{<,=,>,\leq,\geq\}$: $\sigma_C(R)$ est la relation formée de tous les n-uplets de R qui satisfont la formule C. $\sigma_C(R)$ est donc une relation qui a le même schéma que R, et qui a moins de lignes que R. On peut définir formellement $\sigma_C(R)$ par induction sur la formule C.

- Si C est de la forme $A * B$ avec $* \in \{=,<,>=,>,<=\}$ alors $\sigma_C(R)$ est la sous-table formée des lignes pour lesquelles les éléments xA et xB des colonnes A et B vérifient la condition $xA * xB$
- Si C est de la forme $A * \alpha$ avec $* \in \{=,<,>=,>,<=\}$ et α une constante, alors $\sigma_C(R)$ est la sous-table formée des lignes pour lesquelles l'éléments xA de la colonne A vérifie la condition $xA * \alpha$
- Si C est de la forme $C1\&C2$, $\sigma_C(R)$ est l'intersection des tables $\sigma_{C1}(R)$ et $\sigma_{C2}(R)$
- Si C est de la forme $C1ouC2$, $\sigma_C(R)$ est la réunion des tables $\sigma_{C1}(R)$ et $\sigma_{C2}(R)$
- Si C est de la forme $nonC1$, $\sigma_C(R)$ est la table $R \setminus \sigma_{C1}(R)$

En SQL la sélection s'opère par la commande

SELECT (liste d'attributs) FROM table WHERE condition.

Si on ne précise pas la liste d'attributs, tous sont pris par défaut. On peut aussi ne pas préciser la condition, on obtient alors une projection.

Remarque 3.7 Seules les sélections correspondant aux formules atomiques de la forme $C = (A = B)$, $C = (A < B)$, $C = (A = \mathit{constante})$, $C = (A < \mathit{constante})$, $C = (A > \mathit{constante})$, sont indispensables, les autres sélections peuvent facilement s'obtenir à partir de ces primitives à l'aide des opérations booléennes.

3.3.6 Renommage

$\rho_{S(B_1,\ldots,B_n)}(R)$ définit une relation identique à R mais qui se nomme S et dont les attributs se nomment $B_1,\ldots,B_n$.

3.3.7 Jointure naturelle

On suppose que R et R' ont des attributs communs qui ont le même nom et le même type. Alors la jointure de R et R', notée $R \bowtie R'$, est la relation dont les attributs sont ceux de R et R', et dont les lignes sont celles dont les restrictions aux attributs R (resp. R') sont dans R (resp. R'). En d'autres termes si R a $k+n$ attributs et R' a $k+n'$ attributs, avec k attributs communs, alors $R \bowtie R'$ est la relation ayant $k+n+n'$ attributs, et formée des $(k+n+n')$-uplets dont la projection $k+n$ est dans R et dont la projection $k+n'$ est dans R'.

On voit que la jointure s'obtient par restriction à partir du produit cartésien :

- le cas où $n = n' = 0$ redonne l'intersection (c'est-à-dire que dans ce cas les deux relations R et R' ont exactement les mêmes attributs),
- le cas où $k = 0$ redonne produit cartésien (c'est-à-dire que dans ce cas les deux relations R et R' n'ont aucun attribut commun).

Remarque 3.8 La jointure des relations est reliée à l'opération de composition des fonctions, qui est une opération fondamentale en théorie des ensembles. Soient $f\colon A \longrightarrow B$ et $g\colon B \longrightarrow C$ deux fonctions, alors

$$\begin{aligned}\text{graphe}(g\circ f) &= \{(a,c)/\exists b \text{ avec } (a,b)\in\text{graphe}(f) \text{ et } (b,c)\in\text{graphe}(g)\}\\ &= \pi_{(1,3)}\Big(\text{graphe}(f)\bowtie\text{graphe}(g)\Big)\end{aligned}$$

La jointure est une opération très couteuse en temps et espace. Il faut l'implanter judicieusement (avec des index ou tables de hash-coding).

La jointure peut perdre des informations (cas des “dangling tuples” ou n-uplets d'une des deux relations qu'on ne peut joindre avec aucun des n-uplets de l'autre relation : on introduit une opération de jointure externe, qui complète ce type de n-uplet avec des nuls ($\perp$) pour les faire apparaitre dans la jointure externe.

3.3.8 Theta-Jointure

Soient $R(A_1,\ldots,A_n)$ et $R'(B_1,\ldots,B_m)$ deux relations et soit θ le prédicat $=$ ou $<$. Alors $R \bowtie_{i\theta j} R'$ est la restriction du produit cartésien de R et R' par la formule $A_i\theta B_j$, c'est-à-dire $R \bowtie_{i\theta j} R' = \sigma_{A_i\theta B_j}(R\times R')$.

3.3.9 Semi-Jointure

$R \rhd\!< R'$ est la projection de $R \bowtie R'$ sur les attributs de R : ce sont donc les lignes de R qui peuvent se prolonger dans R'.

EXERCICE 3.6 Étant données les relations R et S suivantes :

R A	B	C
1	2	3
4	5	6
7	8	9

S D	E
3	1
6	2

Calculer les jointures $R \bowtie_{B<D} S$ et $R \bowtie_{B>D} S$. ♢

EXERCICE 3.7 Soient les relations R, S et T suivantes :

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

S B	C	D
b	c	d
b	c	e
a	d	b

T B	C
b	c
a	d

Calculer les jointures naturelles $R \bowtie S$, $S \bowtie T$, $R \bowtie T$, et $R \bowtie (S \bowtie T)$, ♢

EXERCICE 3.8 Soient les relations R, S et S' suivantes :

R A	B
a	b
c	b
d	c

S A	B
b	c
e	a
b	d

S' B	C
b	c
e	a
b	d

Calculer $R \cup S$, $R \backslash S$, $R \bowtie S'$, $\pi_A(R)$, $\sigma_{A=C}(R \times S')$, $R \bowtie_{B<C} S'$ (où $<$ est l'ordre alphabétique). ♢

3.4 Extension de l'algèbre relationnelle par des opérations d'agrégats

Pour pouvoir traduire SQL qui est plus puissant que l'algèbre relationnelle on introduit des extensions de l'algèbre relationnelle qui consistent à introduire des oprérateurs d'agrégat : somme, moyenne, min, max et count. Elles sont résumées sur l'exemple suivant emprunté à Ullmann.

EXEMPLE 3.9 On considère la table :

R A	B
1	2
3	4
2	2
2	2

$sum(B) = 2 + 4 + 2 + 2 = 10$, $avg(A) = (1 + 3 + 2 + 2)/2 = 2$, $max(B) = 4$, $min(A) = 1$, $count(A) = 4$.

On introduit aussi un opérateur de "group" qui partitionne les lignes selon la valeur de l'attribut qui est dans la colonne indiquée dans par les "group". Les opérateurs d'agrégats sont dénotés de manière synthétique sous la forme $\gamma_{opn_1(A_1),\ldots,opn_k(A_k)}$ où toutes les opérations $opn_1, \ldots, opn_k$ doivent être des opérations d'agrégat et si l'un (ou plusieurs) des opn_i est un "group" alors toutes les autres opérations indiquées dans le γ sont appliquées sur *chaque partie de la partition* définie par le "group" des opn_i.

Exemple 3.10 On considère la table :

fournit

Anom	*Fnom*	*Prix*
Lait	Carrefour	5,10
Lait	Carrefour	5,15
Lait	Leclerc	5,10
Lait	Auchan	5,00
Lait	Auchan	5,20
Pain	Carrefour	4,40
Pain	Leclerc	5,15
Chocolat	Carrefour	12,30
Chocolat	Intermarché	12,00
Chocolat	Intermarché	12,30

l'opération $\gamma_{group(Anom),group(Fnom),AVG(prix)\to prixmoyen}$ d'algèbre relationnelle étendue renvoie comme résultat la table qui donne le prix moyen de chaque article pour chaque fournisseur, c'est-à-dire la table :

Anom	*Fnom*	*Prixmoyen*
Lait	Carrefour	5,12
Lait	Leclerc	5,10
Lait	Auchan	5,10
Pain	Carrefour	4,40
Pain	Leclerc	5,15
Chocolat	Carrefour	12,30
Chocolat	Intermarché	12,15

Exercice 3.9 On considère la table fournit de l'exemple 3.10.

1. Donner une requête qui produise un agrégat en comptant le nombre de lignes par article et par magasin. On veut obtenir

Anom	*Fnom*	*Nombre*
Lait	Carrefour	2
Lait	Leclerc	1
Lait	Auchan	2
Pain	Carrefour	1
Pain	Leclerc	1
Chocolat	Carrefour	1
Chocolat	Intermarché	2

2. Donner une requête qui produise un agrégat donnant le prix moyen par article.

3. Donner une requête qui, pour chaque article et chaque magasin, donne la valeur des trois agégats suivant : le prix maximum, le prix minimum et le prix moyen. $\diamond$

3.5 Un petit Résumé des notions de base du modèle relationnel

- table = relation
- nom de colonne = attribut
- ligne = n-uplet
- schéma de relation = nom-de-relation(attributs) avec éventuellement d'autres informations : clés, autres contraintes (pas d'ordre sur les attributs)
- instance d'une relation = l'ensemble des lignes qui constituent la relation correspondant au schéma dans la base de donnée considérée
- schéma de base de données = un ensemble de schémas de relations
- requête = une expression de l'algèbre relationnelle

EXERCICE 3.10 On considère une base de données qui contient des informations concernant les épreuves de ski alpin de la saison en cours. La base contient trois relations de schémas $R_1(L, T, S, N, Pr, C)$, $R_2(N, Pr, A, S, Pa)$ et $R_3(L, Pa)$.

- R_1 permet d'obtenir pour une épreuve d'un type T donné (*descente, slalom, géant* ou *supergéant*), épreuve *homme* ou *femme* (selon la valeur de S), qui se déroule en un lieu L donné, le classement C d'un compétiteur dont on connaît le nom N et le prénom Pr.
- R_2 associe le nom N, le prénom Pr, l'âge A, le sexe S (*homme* ou *femme*) et le pays Pa d'un skieur.
- R_3 associe à un lieu L de compétition le pays Pa où se trouve ce lieu.

Exprimer les requêtes suivantes en algèbre relationnelle :

1. Qui (nom et prénom) est arrivé premier au slalom hommes de Chamonix ?
2. Quelles skieuses de moins de 20 ans ont obtenu une place en descente dans les 5 premières ?
3. Quels skieurs (homme ou femme) ont gagné un géant se déroulant dans leur pays ?
4. Qui (nom et prénom) est arrivé dernier au slalom hommes de Chamonix ?
5. Quels sont les pays ayant un skieur (de ce pays) qui a participé à au moins une épreuve. $\diamondsuit$

EXERCICE 3.11 Expliciter (en français) la signification des expressions suivantes :

(i) $\pi_{N,Pr}(\sigma_{T=slalom \wedge S=homme}(R_1)) - \pi_{N,Pr}(\sigma_{A>25}(R_2))$.

(ii) $\pi_{Pa,N,Pr}(R_1 \bowtie R_3) \div \pi_{N,Pr}(R_2)$. $\diamondsuit$

EXERCICE 3.12 Trouver une expression “plus simple” équivalente à (ii). $\diamondsuit$

CHAPITRE 4

SQL

SQL est un acronyme pour "Structured Query Language" qui a été conçu par IBM, et a succédé au langage SEQUEL. C'est maintenant le langage le plus utilisé dans les SGBD commerciaux.

SQL permet de

- interroger la base et formuler des requêtes (instruction SELECT)
- manipuler les données (ordres INSERT, UPDATE, DELETE)
- définir les données (ordres CREATE, ALTER, DROP)
- contrôler l'accès aux données (ordres GRANT, REVOKE)

SQL ne distingue pas majuscules ou minuscules et n'accepte pas d'accents.

4.1 Commandes de définition de données

Ce sont les commandes CREATE TABLE *Rname* qui crée une table *Rname*, et son opposée DROP TABLE *Rname* qui supprime la table *Rname*.

La syntaxe a la forme générale

CREATE TABLE *Rname* (liste d'éléments)

où chaque élément est de la forme : attribut_i format_i, où format_i définit le format (chaîne de caractères, nombre, etc.) des données de type attribut_i.

les formats sont de la forme

- INT ou INTEGER ou NUMERIC (en Oracle un seul type NUMBER)
- REAL ou FLOAT
- CHAR(n) chaîne de caractères de longueur n exactement, complétée par un caractère de remplissage si nécessaire
- VARCHAR(n) chaîne de caractères de longueur au plus n. Il peut y avoir des variations dans la syntaxe des types selon les versions de SQL.

Par exemple la commande SQL suivante permet de créer la table fournit de l'exemple 3.4

```
CREATE TABLE fournit(
   Fnom CHAR(20) NOT NULL,
   Anom CHAR(10) NOT NULL,
   prix NUMERIC(6,2)
);
```

Fnom et Anom sont des chaînes de caractères de longueur 20 et 10, et prix est un nombre ayant 6 chiffres dont 2 après la virgule. On interdit les nuls dans les composantes correspondant aux attributs Fnom et Anom : en effet ces deux attributs forment la clé de la relation, par contre on permet des nuls pour l'attribut prix qui peut être momentanement indéfini.

On peut aussi déclarer les clés et des index qui permettent d'accélérer les recherches dans la base. Par exemple une clé candidate est déclarée par UNIQUE(liste des attributs de la clé) et la clé principale est déclarée par PRIMARY KEY, qu'on peut mettre après le type de l'attribut qui constitue la clé si celle-ci est constituée d'un unique attribut (sinon il faut la déclarer à part). Une PRIMARY KEY ne peut avoir qu'un seul attribut, une clé formée de plusieurs attributs sera donc déclarée à l'aide de UNIQUE (qui peuvent se mettre soit aprés l'attribut, soit en fin de déclaration.

```
CREATE TABLE presente(
   Anom CHAR(10) NOT NULL,
   Rnom CHAR(10) NOT NULL,
   PRIMARY KEY  (Rnom )
      USING INDEX TABLESPACE indx
);
```

```
CREATE TABLE fournit(
   Anom CHAR(10) UNIQUE,
   Fnom CHAR(20) UNIQUE,
   prix NUMERIC6,2
);
```

Une clé étant utilisée comme un index force la création d'une table d'index, création qui peut se faire indépendamment. on aurait pu écrire (pour simplement créer un index) :

```
CREATE INDEX indx
ON  presente(Rnom);
```

ou (pour créer à la fois une clé et un index sur la clé) :

```
CREATE UNIQUE INDEX indx
ON  presente(Rnom);
```

On peut aussi déclarer que certains attributs seront "NOT NULL" (tout n-uplet doit avoir une valeur bien définie sur cet attribut) ou une valeur par défaut "DEFAULT" à utiliser quand la valeur de cet attribut est indéfinie.

Les clés des types d'entité faibles ou dépendants correspondent à la notion de *clé étrangère*. Par exemple, la table "equipe" de la figure 2.12 sera déclarée par :

```
CREATE TABLE studio(
nom CHAR(20) PRIMARY KEY ,
adresse CHAR(40)
);

CREATE TABLE equipe(
numero INTEGER PRIMARY KEY ,
nomstudio CHAR(20) REFERENCES
   studio(nom)
);
```

ou plus simplement

```
CREATE TABLE equipe(
numero INTEGER PRIMARY KEY ,
nomstudio CHAR(20)
FOREIGN KEY  nomstudio REFERENCES
   studio(nom)
);
```

Chaque fois qu'une valeur (non égale à nul) apparaît dans la colonne nomstudio de equipe, elle doit aussi apparaître dans la colonne nom de la table studio. Il faudra prévoir des mécanismes élaborés pour préserver l'intégrité de la base lors de mise à jour dans l'une ou l'autre des deux relations equipe et studio (triggers).

4.2 Commandes de manipulation de données

C'est principalement la commande SELECT, mais aussi INSERT, DELETE, UPDATE.

4.2.1 La commande SELECT

La commande SELECT qui est de loin la plus utilisée en SQL a la forme :

$$\texttt{SELECT}\ R_{i_1}.A_1, \ldots, R_{i_n}.A_n$$
$$\texttt{FROM}\ R_1, \ldots, R_m$$
$$\texttt{WHERE}\ C$$

où C est une condition : cette commande SELECT est équivalente à la requête d'algèbre relationnelle

$$\pi_{(R_{i_1}.A_1,\ldots,R_{i_n}.A_n)}\big(\sigma_C(R_1 \times \cdots \times R_m)\big)$$

Cette commande renvoie une table (sans nom) et de schéma $R_{i_1}.A_1, \ldots, R_{i_n}.A_n$.

Dans toute la suite, et sauf mention expresse du contraire, les exemples se rérèrent à la base à huit relations de l'exemple 3.4.

Dans l'exemple 3.4 la requête "donner les noms des employés du rayon liquides" sera traduite par la commande SQL

```
SELECT Enom
FROM  Employetravaille
WHERE  Rnom='liquides';
```

cette commande SQL correspond à la requête d'algèbre relationnelle

$$\pi_{\text{Enom}}(\sigma_{\text{Rnom=liquides}}\text{Employetravaille})$$

Si l'on voulait imprimer toute l'information contenue dans la table Employetravaille concernant les employés du rayon liquide, on aurait pu écrire

```
SELECT Enom, salaire ,Rnom
FROM  Employetravaille
WHERE  Rnom='liquides';
```

ou (dans certaines versions de SQL on peut même omettre l'étoile)

```
SELECT *
FROM  Employetravaille
WHERE  Rnom='liquides';
```

Donner la liste des noms de fournisseurs qui fournissent du beaujolais, se traduit en requête d'algèbre relationnelle par : $\pi_{\text{Fnom}}(\sigma_{\text{Anom='beaujolais'}}(\text{Fournisseur} \bowtie \text{fournit}))$ et en SQL :

```
SELECT Fnom
FROM  Fournisseur , fournit
WHERE  Anom='beaujolais'
AND  Fournisseur.Fnom = fournit.Fnom;
```

Exercice 4.1 La base de données est la base décrite par le schéma à huit relations de l'exemple 3.4. Donner la liste des fournisseurs qui fournissent au moins un des articles commandés par Jules. ♢

Exercice 4.2 La base de données est la base décrite par le schéma à huit relations de l'exemple 3.4. Donner la liste des clients débiteurs. ♢

Donner les noms des clients ayant commandé au moins cinq camemberts avec la date de la commande, en requête d'algèbre relationnelle :

$$\pi_{\text{Cnom,date}}(\sigma_{(\text{Anom=camembert})\&(\text{Quantite}>=5)}(\text{Commandeemane-de} \bowtie \text{comprend}))$$

et en SQL :

```
SELECT Cnom,date
FROM  Commandeemane-de, comprend
WHERE  Anom='camembert'
AND  Quantité>=5
AND  Commandeemane-de.Conumero = comprend.Conumero;
```

On peut aussi mettre la table renvoyée par la requête dans une table, il suffit pour cela de créer une **vue**, ce qui se fait par la commande

CREATE VIEW $vue(B_1, \ldots, B_n)$ AS
SELECT $R_{i_1}.A_1, \ldots, R_{i_n}.A_n$
FROM $R_1, \ldots, R_m$
WHERE C ;

On a choisi de renommer $R_{i_1}.A_1, \ldots, R_{i_n}.A_n$ en $B_1, \ldots, B_n$, mais ce n'est pas obligatoire.

Exercice 4.3 La base de données est décrite par le schéma à huit relations de l'exemple 3.4. Créer une vue qui donne la liste des articles vendus par le fournisseur McWay, avec leurs prix. ♢

Exercice 4.4 La base de données est la base décrite par le schéma à huit relations de l'exemple 3.4. Donner les noms des clients et les dates de commandes des clients ayant commandé du camenbert et du beaujolais le même jour. ♢

On peut aussi faire du pattern-matching en utilisant LIKE pour spécifier qu'une chaîne de caractères contient un certain motif : % représente "n'importe quelle chaîne de caractères" et _ représente “n’importe quel caractère”. Par exemple, pour trouver tous les clients dont l’adresse est rue Jussieu,

```
SELECT Cnom
FROM  Client
WHERE  Cadresse LIKE '% Jussieu';
```

Exercice 4.5 On considère le schéma de base de donnée suivant (extrait du polycopié de S. Grigorieff, et du livre de P. Matthieu *Des bases de données à l’internet*, Vuibert)

1. spectacle(**numero**,titre,salle,realisateur)
2. jouer(**Anom**,**numero**)
3. representation(**date**,**numero**, **heure**, tarif)

1. Donner la liste des titres de spectacles.

2. Donner la liste des titres de spectacles qui se jouent salle Pleyel.

3. Donner la liste des acteurs qui jouent dans Turandot.

4. Donner la liste des titres de spectacles qui ne se jouent qu’en soirée. ◇

On peut aussi mettre des sous-requêtes dans les WHERE ; la requête de l’exercice 4.1 peut ainsi s’exprimer comme suit :

```
SELECT Fnom
FROM  fournit
WHERE  Anom IN
   (SELECT Anom
   FROM  comprend
   WHERE  Conumero IN
      (SELECT Conumero
      FROM  Commandeemane-de
      WHERE  Cnom='Jules'));
```

4.2.2 Multi-ensembles

La sémantique de SQL est une sémantique de multi-ensemble (en anglais *bag*) et nom pas d’ensembles. Par exemple, dans l’exercice 4.1, si un même fournisseur fournit plusieurs des articles commandés par Jules, on retrouvera ce fournisseur plusieurs fois dans la réponse à la requête ; pour ne l’avoir qu’une seule fois, il faut modifier la commande SELECT en la remplaçant par SELECT DISTINCT Fnom.

Par exemple, si R est la relation

R A	B
1	2
1	4
3	2

La requête

```
SELECT A
FROM  R
WHERE  ( (B=2) OR (B=4) );
```

renvoie le résultat $\{1, 1, 3\}$. Pour avoir seulement $\{1, 3\}$ il faut écrire

```
SELECT DISTINCT A
FROM  R
WHERE  ( (B=2) OR (B=4) );
```

SQL a des commandes UNION, INTERSECT et EXCEPT, leur sémantique par défaut est la sémantique au sens des ensembles et pas au sens des multi-ensembles, alors que la sémantique de SELECT est prise au sens multi-ensemble.

REMARQUE 4.1 Sémantique multi-ensembliste :

- $\cup_M$ ajoute le nombre de fois où un élément apparaît dans chaque ensemble : $\{1,2,1\} \cup_M \{1,2,3\} = \{1,1,1,2,2,3\}$
- $\cap_M$ prend le minimum du nombre de fois où un élément apparaît dans chaque ensemble : $\{1,2,1\} \cap_M \{1,2,3\} = \{1,2\}$
- $\setminus_M$ soustrait le nombre de fois où un élément apparaît dans le second ensemble du nombre de fois où il apparaît dans le premier ensemble : $\{1,2,1\} \setminus_M \{1,2,3,3\} = \{1\}$

Toutefois, les commandes UNION, INTERSECT et EXCEPT de SQL ne prennent pas cette sémantique multi-ensembliste mais la sémantique usuelle de $\cup, \cap, \setminus$.

On a aussi les mots-clés ANY et ALL qui jouent le rôle de quantificateurs existentiel et universel (ne pas confondre any avec le "n'importe lequel" du langage courant qui signifie en général "pour tout", c'est-à-dire a la signification de ALL). Par exemple, pour trouver l'article le plus cher, on écrira la requête

```
SELECT Anom
FROM   fournit
WHERE  prix >= ALL(
   SELECT prix
   FROM   fournit
   ) ;
```

EXERCICE 4.6 Ecrire une requête qui donne l'article le moins cher vendu par Champion avec son prix. ◊

De même que l'on peut forcer une sémantique ensembliste pour un SELECT en ajoutant le mot-clé DISTINCT, ou peut forcer une sémantique multi-ensembliste en ajoutant le mot-clé ALL après UNION, INTERSECT etc.

4.2.3 Les mot-clés sont EXISTS, IN et NOT IN.

Ces mot-clés sont utilisés dans les conditions.

EXISTS R est vraie si et seulement si la relation R est non vide.

t IN R est vraie si et seulement si le n-uplet t est dans R.

t NOT IN R est vraie si et seulement si le n-uplet t n'est pas dans R.

Par exemple, la requête suivante donne les adresses de tous les supermarchés qui fournissent du beaujolais.

```
SELECT Fadresse
FROM   fournisseur
WHERE  Fnom IN (
   SELECT Fnom
   FROM   fournit
   WHERE  Anom ='beaujolais') ;
```

La requête suivante donne les articles qui ne sont vendus que par un seul supermarché.

```
SELECT Anom
FROM   fournit fournisseur1
WHERE  NOT EXISTS (
   SELECT *
   FROM   fournit
   WHERE ((Fnom<>fournisseur1.Fnom) AND (Anom=fournisseur1.Anom))
   );
```

On remarquera le renommage du n-uplet de fournit en fournisseur1 pour pouvoir être utilisé dans la sous-requête corrélée. On peut écrire aussi

```
SELECT Anom
FROM   fournit fournisseur1
WHERE  Anom NOT IN (
   SELECT Anom
   FROM   fournit
   WHERE ((Fnom<>fournisseur1.Fnom) AND (Anom=fournisseur1.Anom))
   );
```

Il faut parfois écrire `fournit AS fournisseur1` pour renommer la relation fournit.

4.2.4 Les commandes de jointure

SQL admet des opérations JOIN, par exemple : R NATURAL JOIN S, ou bien R JOIN S ON condition, etc. que l'on peut utiliser soit seules, soit dans un SELECT.

La commande CROSS JOIN réalise le produit cartésien. Par exemple, la commande

Rayon CROSS JOIN responsable

renvoie une relation de schéma (Rnom, Rnumero , Enom, Rnom).

La commande

Rayon NATURAL JOIN responsable

renvoie la relation Rayonresponsable(Rnom, Rnumero, ChefRnom).

La relation de schéma (Rnom, Rnumero, ChefRnom, Rnom) où la colonne Rnom est dupliquée sera obtenue par la commande

Rayon JOIN responsable ON Rayon.Rnom = responsable.Rnom.

Par exemple, si R et S sont les relations

R	A	B	C
	1	2	3
	3	4	5
	1	4	5

S	A	B	C
	1	2	5
	1	2	4
	3	4	3

R JOIN S ON (R.A = S.A AND R.B = S.B)

sera la relation

$R.A$	$R.B$	$R.C$	$S.A$	$S.B$	$S.C$
1	2	3	1	2	5
1	2	3	1	2	4
3	4	5	3	4	3

Exercice 4.7 Quel sera le résultat de la requête R JOIN S ON (R.A = S.A OR R.B = S.B) $\diamond$

4.2.5 Les commandes INSERT, DELETE, UPDATE.

Nous illustrons la syntaxe de ces commmandes en prenant l'exemples de la relation fournit(**Fnom**,**Anom**,prix) de l'exemple 3.4.

```
UPDATE fournit
SET  prix = prix *1.1
WHERE  Anom ='beaujolais';
```

```
UPDATE fournit
SET  prix = prix * 0.659;
```

```
INSERT INTO fournit
VALUES  ('Champion','pecharmant',60);
```

```
DELETE FROM fournit
VALUES  ('Champion','champagne',70);
```

On peut aussi insérer tout ou partie du contenu d'une autre table à l'aide des commandes INSERT et SELECT.

4.2.6 Les commandes d'agrégats : MIN, MAX, AVG, SUM, COUNT, GROUPBY

Il s'agit là de commandes qui donnent à SQL un pouvoir d'expression supérieur à celui de l'algèbre relationnelle pure (sans l'extension par les *group* et opérateurs d'agrégats). Nous donnons deux exemples de commandes utilisant des opérations d'agrégats.

Trouver le prix moyen du beaujolais dans la relation "fournit" est fait par la requête SQL

```
SELECT AVG(prix)
FROM fournit
WHERE  Anom='beaujolais';
```

Chaque n-uplet sera compté une fois. SELECT ayant une sémantique multi-ensembliste, si par exemple on veut compter le nombre de prix différents auxquels le beaujolais est vendu, il faudra introduire un DISTINCT et écrire

```
SELECT COUNT(DISTINCT prix)
FROM fournit
WHERE  Anom='beaujolais';
```

On peut utiliser le mot-clé GROUP BY qui permet de partitionner la table renvoyée par le SELECT et d'appliquer les opérations d'agrégat sur les éléments de chaque sous-ensemble de la partition et non pas sur l'entière relation renvoyée par le SELECT. Par exemple, pour trouver le prix moyen de chaque article dans la relation "fournit" on utilisera la requête

```
SELECT Anom ,  AVG(prix)
FROM fournit
GROUP BY  Anom;
```

qui renvoie une table contenant la liste des noms d'articles, avec le prix moyen correspondant.

REMARQUE 4.2 Si on utilise un agrégat alors **toutes** les colonnes du SELECT doivent être soit un agrégat, soit un GROUP BY, c'est-à-dire elles doivent toutes être au même niveau de regroupement.

La clause `HAVING` est similaire à `WHERE`, mais elle est utilisée pour faire des selections sur des groupes : la condition utilise les attributs figurant dans le `SELECT` ou les relations figurant dans le `FROM`, toutefois les attributs ne parcourent que les valeurs du groupe et doivent être un des attributs du regroupement.

Par exemple pour trouver le prix moyen des articles qui sont vendus par au moins 3 fournisseurs différents, on pourra écrire :

```
SELECT Anom ,  AVG(prix)
FROM fournit
GROUP BY  Anom
HAVING COUNT(DISTINCT  Fnom)>=3 ;
```

EXERCICE 4.8 La base de données est la base décrite par le schéma à huit relations de l'exemple 3.4.

Trouver le fournisseur le moins cher pour le beaujolais, avec le prix du beaujolais. ◊

EXERCICE 4.9 La figure 4.1 est un schéma entité-relation représentant des visites dans un centre médical.

1 Répondez aux questions suivantes *en fonction des caractéristiques du schéma*, indépendamment de leur vraissemblance.

   - Un patient peut-il effectuer plusieurs visites ?
   - Un médecin peut-il recevoir plusieurs patients dans la même consultation ?
   - Peut-on prescrire plusieurs médicaments dans une même consultation ?
   - Deux médecins différents peuvent-ils prescrire le même médicament ?

2. Construire le schéma relationnel correspondant, en indiquant précisément la clef primaire, les clefs étrangères et les contraintes éventuelles.

3. Exprimez les requêtes suivantes (en algèbre et en SQL) :

   - La liste des consultations du Dr Dupont.
   - Les patients qui ont consulté le 1er février 2001.
   - Les dates auxquelles on a prescrit de l'aspirine.

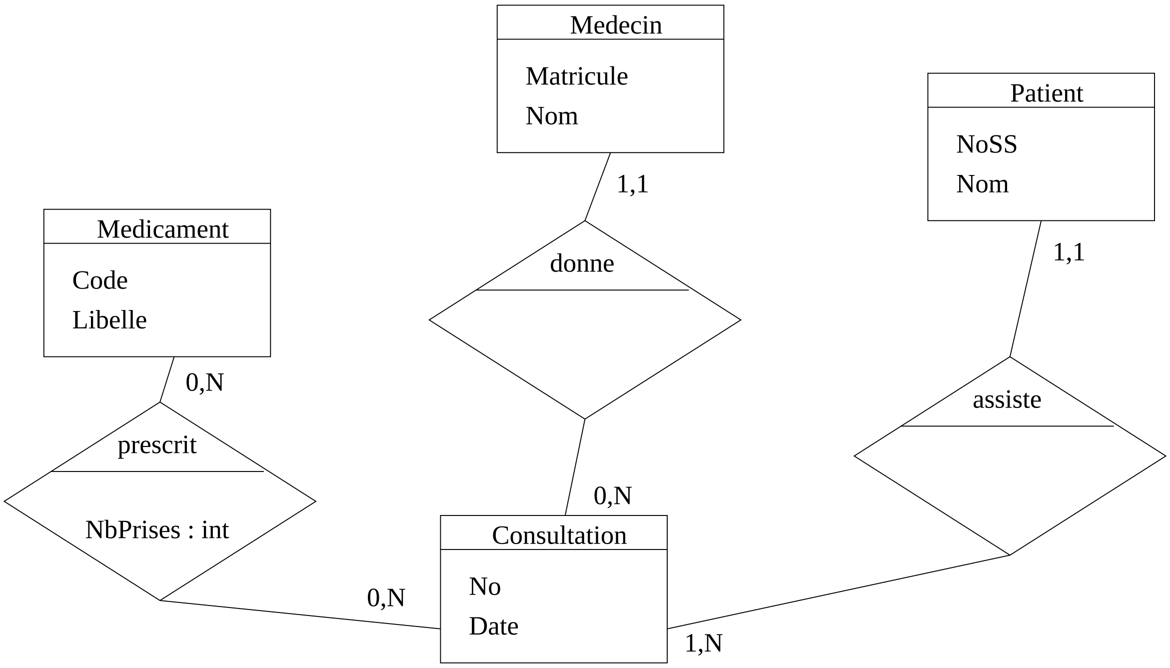

FIGURE 4.1 visites dans un centre médical

◊

EXERCICE 4.10 Voici une description du fonctionnement d'une bibliothèque.

- La bibliothèque peut posséder plusieurs exemplaires d'un même livre.
- Une même œuvre peut avoir été éditée par plusieurs éditeurs différents.
- On appellera "livre" une édition d'une œuvre.
- Un livre ne peut être édité que par un seul éditeur.
- Pour simplifier on supposera qu'un livre ne concerne qu'une seule œuvre et que tous les livres sont en français.
- Une œuvre peut avoir été écrite par plusieurs auteurs.
- On ne garde pas un historique des emprunts.
- Un lecteur ne peut emprunter plus de 3 livres la fois.
- Un lecteur ne peut emprunter de livre s'il est en retard pour en rendre un autre.
- Chaque jour une liste des retards dans la restitution des livres est imprimée.
- Si un lecteur rend un livre avec n jours de retard il n'a plus le droit d'emprunter un autre livre tant qu'il n'a pas rendu le livre et pendant les $2n$ jours qui suivent la date o il rend le livre.
- Un lecteur peut se mettre en liste d'attente pour l'emprunt d'un livre.

1. La figure 4.2 donne un diagramme de classes pour traduire le fonctionnement de cette bibliothèque. Ce diagramme comporte une erreur dans les multiplicités. Trouvez-la.
2. Traduisez ce diagramme de classes en un schéma relationnel.

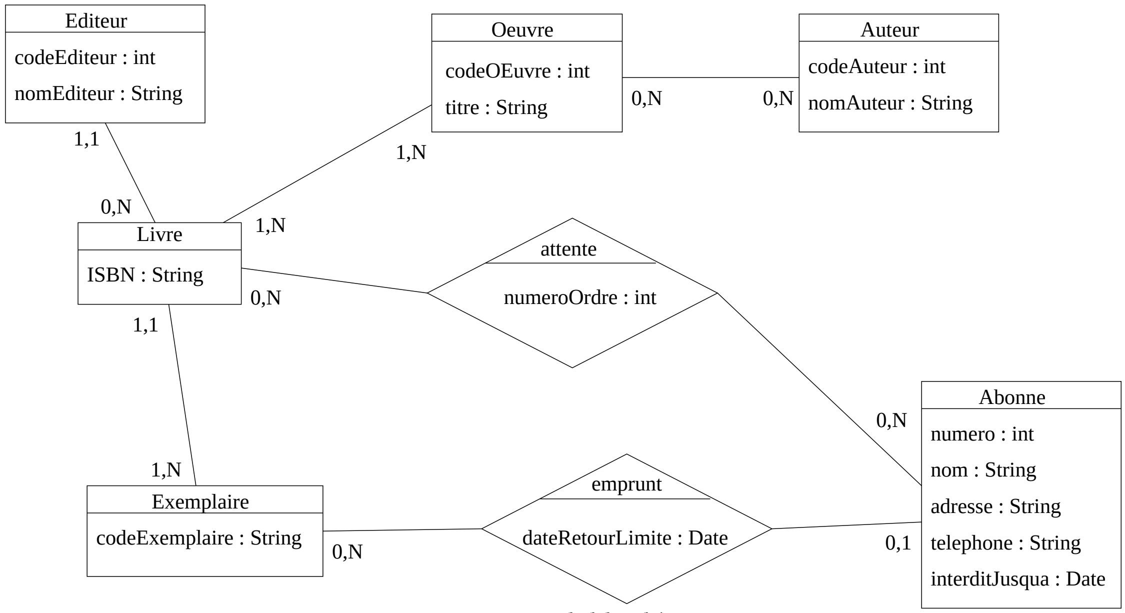

FIGURE 4.2 bibliothèque

◇

EXERCICE 4.11 Les tables equipes, coureurs, etapes, temps et pays sont un exemple simplifié du modèle de données relationnel utilisé pour la gestion du Tour de France 97.

Equipes

CodeEquipe	*nomEquipe*	*Directeur*
BAN	BANESTO	Eusebio UNZUE
COF	COFIDIS	Cyrille GUIMARD
CSO	CASINO	Vincent LAVENU
FDJ	LA FRANCAISE DES JEUX	Marc MADIOT
FES	FESTINA	Bruno ROUSSEL
GAN	GAN	Roger LEGEAY
ONC	O.N.C.E.	Manolo SAIZ
TEL	TELEKOM	Walter GODEFROOT
...	...	...

Coureur

NoCourr	*NomCoureur*	*CodeEquipe*	*CodePays*
8	ULLRICH Jan	TEL	ALL
31	JALABERT Laurent	ONC	FRA
61	ROMINGER Tony	COF	SUI
91	BOARDMAN Chris	GAN	G-B
114	CIPOLLINI Mario	SAE	ITA
151	OLANO Abraham	BAN	ESP
...	...	...	...

Etapes

NoEtpe	*Date*	*VilleDepart*	*VilleArrivee*	*NbeKM*
1	06-jul-97	ROUEN	FORGES-LES-EAUX	192
2	07-jul-97	ST-VALERY-EN-CAUX	VIRE	262
3	08-jul-97	VIRE	PLUMELEC	224
...	...	...	...	...

Temps

NoCourr	*NoEtpe*	*TempsRealise*
8	3	04 :54 :33
8	1	04 :48 :21
8	2	06 :27 :47
31	3	04 :54 :33
31	1	04 :48 :37
31	2	06 :27 :47
61	1	04 :48 :24
61	2	06 :27 :47
91	3	04 :54 :33
91	1	04 :48 :19
91	2	06 :27 :47
114	3	04 :54 :44
114	1	04 :48 :09
114	2	06 :27 :47
151	3	04 :54 :33
151	1	04 :48 :29
151	2	06 :27 :47
...	...	...

Pays

Codepays	*NomPays*
ALL	ALLEMAGNE
AUT	AUTRICHE
BEL	BELGIQUE
DAN	DANEMARK
ESP	ESPAGNE
FRA	FRANCE
G-B	GRANDE BRETAGNE
ITA	ITALIE
P-B	PAYS-BAS
RUS	RUSSIE
SUI	SUISSE

À l'aide des primitives SELECTION, JOINTURE, PROJECTION et de l'extension par agrégats de l'algèbre relationnelle, formulez les requêtes suivantes :

1. Quelle est la composition de l'équipe FESTINA (Numéro, nom et pays des coureurs) ?
2. Quel est le nombre de kilomètres total du Tour de France 97 ?
3. Quels sont les noms des coureurs qui ont participé à toutes les étapes ?
4. Quel est le classement par coureurs (nom, temps réalisé, uniquement pour les coureurs ayant participé à toutes les étapes) ?
5. Quelle est l'équipe la plus rapide (en moyenne) ?
6. Quelle est le temps total de la voiture balai ?
7. Quels sont les noms des coureurs qui sont arrivés dans les trois premiers à chacune des étapes auxquelles ils ont participé (question spécial dopage).

Exécutez vos stratégies de requête "à la main" sur les tables. Donnez la traduction de vos requêtes en SQL. ◇

CHAPITRE 5

EVALUATION DE REQUÊTES

Une requête SQL exprime le résultat que l'on souhaite obtenir, mais ne donne aucun moyen de le calculer. Une requête d'algèbre relationnelle donne en théorie un moyen de calculer ce résultat, mais ce moyen sera en général un calcul peu efficace qu'il faudra donc améliorer. Le SGDB devra donc

1. traduire et optimiser les requêtes SQL en opérations sur les tables de base
2. améliorer les requêtes d'algèbre relationnelle

En général cela se fera en trois étapes :

1. traduire la requête SQL en un arbre de requête
2. traduire l'arbre de requête en une expresion d'algèbre relationnelle
3. améliorer la requête d'algèbre relationnelle

Il y a de nombreuses manières d'exprimer une même relation, et quelques heuristiques pour améliorer l'efficacité. Toutefois, il n'y a pas en général de requête optimale équivalente à une requête donnée (sauf pour le cas de requêtes conjonctives). Cela découle du résultat théorique suivant : il n'est pas décidable si une requête est minimale parmi toutes les requêtes qui fournissent le même résultat. La seule exception est le cas des requêtes conjonctives, c'est-à-dire les requêtes SELECT FROM WHERE simples, où la condition du WHERE est une conjonction (i.e. un "and" sépare deux formules successives) de formules toutes sous la forme $A_i = B_j$, ou $A_i = constante$: pour ces requêtes, on peut trouver une requête minimale qui donne la même réponse, toutefois le temps pour calculer cette requête minimale est fort long (en termes techniques, le calcul de la requête minimale est NP-complet).

5.1 Traduction des requêtes SQL en arbres

Les requêtes simples

SELECT $R_{i_1}.A_1, \ldots, R_{i_n}.A_n$
FROM $R_1, \ldots, R_m$
WHERE C

se traduisent automatiquement en produit des relations dans le FROM, puis sélection correspondant au where et enfin projection sur les attributs du SELECT, soit

$\pi_{R_{i_1}.A_1,\ldots,R_{i_n}.A_n}\big(\sigma_C(R_1 \times \cdots \times R_m)\big)$

Considérons la requête

```
SELECT Fnom
FROM  Fournisseur , fournit
WHERE  Anom='beaujolais' ;
```

La traduction automatique en algèbre relationnelle donne

$$\pi_{\text{Fnom}}\big(\sigma_{\text{Anom='beaujolais'}}(\text{Fournisseur} \times \text{fournit})\big).$$

Il est clair que cette traduction automatique est simpliste et fort peu efficace : elle est largement améliorable, par exemple, si le FROM ne comporte que deux relations, on pourra remplacer le produit cartésien par une jointure, éventuellement une theta-jointure. Dans l'exemple précédent, on simplifie en

$$\pi_{\text{Fnom}}\big(\text{Fournisseur} \bowtie_{\text{Anom='beaujolais'}} \text{fournit}\big).$$

5.2 Heuristiques pour améliorer les requêtes

L'idée générale est de diminuer autant que faire se peut la taille des tables en jeu. D'où, si on représente la requête comme un arbre :

- faire descendre les sélections vers les feuilles, par exemple, on remplacera $\sigma_C(R \bowtie S)$ par $\sigma_C(R) \bowtie S$ si la condition C ne fait intervenir que les attributs de R, ou par $R \bowtie \sigma_C(S)$ si symétriquement la condition C ne fait intervenir que les attributs de S.
- diviser les sélections, par exemple, on remplacera $\sigma_{C\&D}(R)$ par $\sigma_C(\sigma_D(R))$

Par exemple, la requête suivante sur l'exemple 3.4 : donner les noms des clients qui ont commandé du beaujolais entre le 20 octobre et le 10 novembre, qui correspond à requête d'algèbre relationnelle

$$\pi_{\text{Cnom}}\Big(\sigma_{\text{Anom='beaujolais'}\&20-10\leq\text{date}\leq 10-11}(comprend \bowtie commandeemane - de)\Big)$$

sera grandement améliorée si on la remplace par

$$\pi_{\text{Cnom}}\Big(\sigma_{\text{Anom='beaujolais'}}(comprend) \bowtie \sigma_{20-10\leq\text{date}\leq 10-11}(commandeemane - de)\Big)$$

les arbres correspondants sont dessinés ci-dessous.

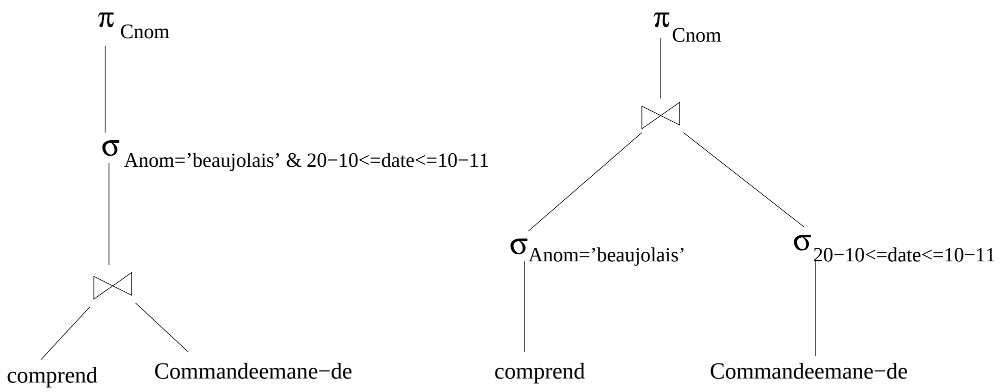

FIGURE 5.1

5.2.1 Lois pour simplifier les requêtes avec des sélections

Ces lois sont intéressantes car les sélections diminuent le nombre de $n$uplets.

- Remplacer $\sigma_C(R \cup S)$ par $\sigma_C(R) \cup \sigma_C(S)$ et remplacer $\sigma_C(R \cap S)$ par $\sigma_C(R) \cap \sigma_C(S)$
- Remplacer $\sigma_C(R \setminus S)$ par $\sigma_C(R) \setminus \sigma_C(S)$ ou bien $\sigma_C(R) \setminus S$
- pour les produits et jointures, les attributs de C peuvent ne figurer par exemple que dans R, alors on remplacera $\sigma_C(R \bowtie S)$ par $\sigma_C(R) \cup S$, et $\sigma_C(R \times S)$ par $\sigma_C(R) \times S$, et *mutatis mutandis* si C n'apparaît que dans S ; enfin, si C apparaît dans R et S on le fait passer à la fois dans les deux.

5.2.2 Lois pour simplifier les requêtes avec des projections

Ces lois sont moins intéressantes car les projections diminuent la taille des $n$uplets mais pas le nombre de $n$uplets. Faire descendre une projection dans l'arbre revient à ajouter une nouvelle projection. On peut introduire des nouvelles projections à conditions qu'elles éliminent des attributs qui ne sont pas utilisés par des opérations situées plus haut dans l'arbre ou dans le résultat.

- $\pi_L(R \bowtie S)$ est remplacé par $\pi_L(\pi_M(R) \bowtie \pi_N(S))$ où M (resp. N) sont les attributs de R qui sont communs avec S (resp. R) ou qui sont dans L.
- $\pi_L(R \times S)$ est remplacé par $\pi_L(\pi_M(R) \times \pi_N(S))$ où M (resp. N) sont les attributs de R (resp. S) qui sont dans L.
- $\pi_L(\sigma_C(R)$ est remplacé par $\pi_L(\sigma_C(\pi_M(R))$, où M contient les attributs de L et les attributs de C.

5.2.3 Lois pour simplifier les requêtes avec des agrégats

Il y a très peu de règles générales. Si δ est l'élimination de $n$uplets identiques, et γ un opérateur d'agrégat, alors :

- on peut remplacer $\delta(\gamma_L(R))$ par $\gamma_L(R)$
- on peut remplacer $\gamma_L(R)$ par $\gamma_L \pi_M(R))$, si M contient tous les attributs de R utilisés dans γ.

Lorsqu'on a appliqué toutes les simplifications algébriques proposées ci-dessus, on peut essayer de simplifier encore l'arbre correspondant à la requête en cherchant des sous-requêtes qui seraient communes à plusieurs branches. Enfin nous n'aborderons pas ici le problème de la traduction de la requête ainsi optimisée en un programme de langage machine qui optimise encore et exécute la requête au niveau interne.