Conception des bases de données relationnelles

Fabien De Marchi
Faculté des Sciences et Technologies - Laboratoire LIRIS
Université Lyon 1

24 novembre 2009

Pré-requis :
- Structure du modèle relationnel : attributs, relations, bases de données.
- Langages du relationnel : Algèbre ou calcul relationnel, SQL
- Connaissance élémentaire du modèle E/A.

Objectifs : Approfondir les connaissances du modèle relationnel, et les fondements de la conception des bases de données :
- Principales contraintes et dépendances - Notions de systèmes d'inférence et de bases de règles.
- Processus de normalisation d'une base de données à partir des contraintes
- Conception à l'aide d'un modèle conceptuel de données.

Remarque : ces éléments de cours sont ici pour faciliter le travail personnel des étudiants. Ils ne remplacent pas les cours et exercices effectués en classe, dont le contenu peut varier.

Références
- **Livres :**
  - M. Levene et G. Loizou *A guided tour of relational databases and beyond.* Springer, 1999. Les notations de ce cours sont principalement issues de cette référence.
  - S. Abiteboul, R. Hull et V. Vianu *Fondements des bases de données.* Addison-Wesley, 1995.
  - R. Ramakrishnan et J. Gehrke *Database Management Systems.* 2003 (troisième édition)
- **Sur le Web :**
  - Matériel allant avec le livre de Ramakrishnan et Gehrke

Table des matières

Chapitre 1

Introduction

Qu'est-ce qu'une base de données ? Une base de données est une *collection organisée de données électroniques, logiquement connectées entre elles*. Des données appartiennent à une même base si elles possèdent un lien sémantique, qui dépend d'une application. Par exemple, des restaurants et des cinémas peuvent apparaître dans une même base qui répertorie des sorties dans une ville. Un lien sémantique uni alors ces deux concepts : ils sont tous les deux des sorties particulières dans cette ville.

En général, les objectifs fondamentaux inhérents à la gestion d'une base de données sont :

- la pérénité des données : jamais de données ne doivent être perdues, ce qui nécessite de réfléchir à la sauvegarde, l'archivage, la reprise après panne.
- la cohérence des données par rapport à la réalité. C'est encore plus délicat lors d'accès concurrentiels, ou de réplications de la base.
- l'accès juste et efficace aux données.

Ces différentes capacités sont prises en charge par des logiciels appelés *système de gestion de base de données* (SGBD). Les principaux produits commerciaux sont Oracle, DB2 (IBM) et SQL Server (Microsoft). Plusieurs SGBD gratuits comme MySQL ou PostgreSQL sont de plus en plus utilisés pour la gestion de bases petites et moyennes, et particulièrement adaptés pour s'interfacer avec des pages Web. Ces outils sont basés sur le modèle relationnel, qui est un modèle de données parmi d'autres.

Qu'est-ce qu'un modèle de données ? Tout SGBD est conçu autour d'un *modèle de données* bien défini. Il s'agit de la combinaison de trois éléments :

- Une *structure*, qui correspond à l'organisation logique des données, la forme sous laquelle les utilisateurs vont les percevoir ou les représenter.
- Des *contraintes d'intégrité*, que l'on peut définir sur les données, afin d'en assurer l'intégrité et la cohérence avec le monde réel et les besoins des applications.
- Des langage de *manipulation des données*, pour les mises à jour et les interrogations.

Voici quelques exemples de modèles de données :

- *le modèle réseau*
  - Structure : graphe orienté, les noeuds sont des enregistrements
  - Contraintes d'intégrité : un identifiant pour chaque noeud, un mécanisme de référence entre des noeuds
  - Manipulation : parcours de réseaux.
- le modèle hiérarchique
  - Structure : arborescente (forêts)

  - Contraintes d'intégrité : un identifiant pour chaque noeud, un mécanisme de référence entre des noeuds
  - Manipulation : navigation hiérarchique
- le modèle relationnel
  - Structure : ensembles de relations, qui sont des ensembles de tuples. Graphiquement, une relation est un tableau.
  - Contraintes d'intégrité : principalement les clés primaires (identifiants de tuples) et les clés étrangères (liens entre les relations)
  - Manipulation : algèbre relationnel, calcul relationnel, SQL.
- le modèle déductif
  - Structure : quasiment la même que le modèle relationnel
  - Contraintes d'intégrité : les mêmes que le modèle relationnel
  - Manipulation : langages logiques, le principal est *Datalog*. Contrairement aux langages du modèle relationnel, il admet la récursivité.
- le modèle Objet
  - Structure : logique objet, soit des classes, des objets, des attributs et des méthodes. Peut être vu comme un graphe orienté.
  - Contraintes d'intégrité : identifiant pour les objets, référence entre objets.
  - Manipulation : extensions de SQL comme OSQL ou OQL.
- le modèle Entité/Association
  - Structure : Entités (avec des attributs) et associations entre des entités.
  - Contraintes d'intégrité : identifiants, cardinalités sur les associations
  - Manipulation : aucun.

Ajoutons les modèles semi-structurés, qui s'adaptent à la nature hétérogène des données, principalement trouvées sur le Web. Le principal exemple est XML. En pratique, le relationnel est, de loin, le plus utilisé de tous les modèles de données. Quelques SGBD objet existent, mais rencontrent des problèmes de performances pour les gros volumes de données, dus à l'exécution en cascade de méthodes lors des parcours de données. Des SGBD XML ont aussi plus récemment vu le jour, et sont utilisés, mais n'égalent pas les performances du modèle relationnel.

Les principaux SGBD relationnels intègrent de plus en plus des extentions permettant de "percevoir" les données sous la logique objet ou XML, mais utilisant derrière des bases de données relationnelles pour stocker les données.

Chapitre 2

Présentation du modèle relationnel

Sommaire

Ce chapitre rappel brièvement la *structure* et les *langages* du modèle relationnel, sans entrer dans les détails - l'objectif étant principalement de présenter le contexte et les notations. La partie sur les *contraintes* fera l'objet du chapitre suivant.

2.1 Introduction

Le modèle relationnel a été défini en 1970 par Codd. Cette proposition a succédé aux modèles hiérarchiques et réseaux, avec pour but de définir un modèle simple à comprendre, fondé sur des bases mathématiques solides (logique et théorie des ensembles), et pouvant conduire à une amélioration des performances des outils. L'idée de base est simple : mettre les données "à plat" dans des tableaux, de façon à ce que chaque valeur apparaissant dans les "cases" du tableau soient atomiques. Cette simplification est la principale explication de la performance du modèle relationnel, car elle évite les traitements récursifs (parcours de graphes et d'arbres complexes).

Après une dizaine d'année de développement de la théorie inhérente aux propositions de Codd, les premiers SGBD commerciaux font leur apparition dans les années 80, avec des outils comme ORACLE principalement. Les contributions techniques se combinent alors aux fondements théoriques et le modèle relationnel s'impose comme un standard. Dès le début des années 90, les progrès technologiques fulgurants sur le matériel (processeurs et mémoire disque) conduisent à des bases de plus en plus grosses, pour atteindre des proportions gigantesques de nos jours. La recherche dans le domaine des bases de données relationnelles est toujours active, placée sans cesse face à des nouveaux défis de taille, d'exigence de rapidité, d'hétérogénéité des données. Même si de nouveaux paradigmes pourraient être en passe de voir le jour dans le traitement des bases de données de production, les fondements théoriques présentés dans ce cours constituent toujours la base des techniques utilisées.

2.2 Structure

La structure d'une BD relationnelle est fondée sur la théorie des ensembles. Rappelons qu'un ensemble est une collection *non ordonnée* d'éléments *distincts*, à différencier de la *séquence* (lorsque les éléments sont ordonnés) et du *multi-ensemble* (lorsqu'on accepte les doublons).

Notations les notations suivantes sont communément admises lorsque le contexte le permet. Un ensemble $\{A_1; A_2; ...; A_n\}$ sera noté par la concaténation de ses éléments : $A_1A_2...A_n$. Si X et Y sont deux ensemble, leur union $X \cup Y$ sera notée XY. Une séquence d'éléments, est notée $< A_1, ..., A_n >$. Lorsque le contexte est clair, les séquences et les ensembles seront notés de la même façon.

Exemple 1 L'ensemble $\{A; B; C\}$ sera noté ABC dans ce cours. Les notations ABC et BCA sont équivalentes puisque l'ordre n'a pas d'importance : par convention, on notera les ensembles théoriques en respectant l'ordre alphabétique. L'ensemble $AABC$ n'existe pas, puisque l'élément A apparaît deux fois. Soit les ensemble $X = ABC$ et $Y = BD$, alors leur union $X \cup Y$ sera noté $XY = ABCD$.

2.2.1 Attributs, valeurs et domaines

Soit $\mathcal{U}$ un ensemble infini dénombrable de *noms d'attributs* ou simplement *attributs*, et $\mathcal{D}$ un ensemble infini dénombrable de *valeurs*. Soit $A \in \mathcal{U}$ un *attribut*, le *domaine* de A est un sous-ensemble de $\mathcal{D}$, noté $DOM(A)$.

Hypothèse d'unicité des noms : Soient deux valeurs v_1 et v_2 des éléments de $\mathcal{D}$. On dit que v_1 et v_2 sont égales si et seulement si elles sont syntaxiquement identiques, c'est à dire qu'elles ont le même nom. En d'autres termes, lorsque deux attributs ont la même valeur, ils désignent la même chose.

2.2.2 Schémas de relations, schémas de bases de données et première forme normale

Définition 1 (schémas de relations et de bases de données) Un *schéma de relation* R est un ensemble fini d'attributs (donc $R \subseteq \mathcal{U}$). Un *schéma de base de données* **R** est un ensemble fini de schémas de relation.

Exemple 2
L'exemple suivant sera utilisé tout au long de ce cours. On suppose une application gérant les cours dispensés au sein de l'UFR, regroupant des informations sur les étudiants, les formations, les enseignants, les modules, les UE, les salles de cours et les ressources matérielles (projecteurs, etc...).
La modélisation des étudiants pourrait se faire à l'aide du schéma de relation

$$ETUDIANTS = \{NUM; NOM; PRENOM; AGE; FORMATION\}$$

. Si on représente les autres informations dans d'autres schémas de relations, alors l'union de tous les schémas de relation obtenus constituera le schéma de notre base de données, qu'on pourra appeler $FACULTE$.

Hypothèse de schéma de relation universel : Un schéma de base de données **R** satisfait l'hypothèse de schéma de relation universelle si, lorsque deux attributs portent le même nom, ils désignent le même concept.

Définition 2 (Première forme normale) Un schéma de relation R est en première forme normale (notée $1FN$) si, pour tout attribut $A \in R$, $DOM(A)$ est composé de valeurs atomiques.

La première forme normale est une condition fondamentale du modèle relationnel, garante de sa simplicité et efficacité, et l'hypothèse de schéma de relation universelle est toujours vérifiés, permettant de désigner de façon unique un attribut, sans aucune ambiguïté.

Remarque *Ces restrictions sont nécessaires à assurer les fondements théoriques justes du modèle relationnel. En pratique, on trouve plus de souplesse dans les outils :*

- *Les types complexes sont autorisés dans plusieurs SGBD pour peupler les relations : il s'agit en fait d'extensions de la théorie de base, en utilisant une logique objet. Le modèle utilisé est alors appelé relationnel-objet (Exemple : Oracle). Il permet une modélisation plus intuitive pour les concepteurs, mais le modèle utilisé en fond par le SGBD pour stocker ses données est toujours le relationnel.*
- *Dans tous les outils existants, on peut nommer deux fois le même attribut pour représenter des concepts différents (au sein d'une relation, un attribut n'apparaît toutefois qu'une seule fois. Par exemple, dans notre base de données on pourrait imaginer un attribut NOM qui désigne le nom d'une salle de cours, et le même attribut qui désigne le nom d'une personne ; ce serait une faute de conception, mais impossible à vérifier par les outils !*

Un schéma de relation, ou de bases de données, est donc une boîte vide, avec une structure bien particulière, destinée à contenir des ensembles d'éléments possédant la même structure et donc sémantiquement équivalents.

2.2.3 Tuples, relations et bases de données

Définition 3 (tuple, Relation et Base de Données) Soit $R = A_1...A_n$ un schéma de relation. Un *tuple* sur R est un membre du produit cartésien $DOM(A_1) \times \ldots \times DOM(A_n)$. Une relation r sur R est un ensemble fini de tuples. Une base de données $\mathbf{d}$ sur un schéma de base de données $\mathbf{R} = \{R_1, ..., R_n\}$ est un ensemble de relations $\{r_1, ..., r_n\}$ définies sur les schéma de relation de $\mathbf{R}$.

De façon informelle, on peut donc voir un tuple comme une séquence de valeurs, prises par les attributs du schéma de relation. Si t est un tuple défini sur un schéma de relation R, et X un sous-ensemble de R, on peut restreindre t à X en utilisant l'opération de projection, notée $t[X]$.

Exemple 3 Le tableau suivante représente graphiquement une relation *etudiants* sur le schéma $ETUDIANT$. On suppose que le numéro de formation est une référence vers une autre table qui décrit les formations.

NUM	NOM	$PRENOM$	AGE	$FORMATION$
28	Codd	Edgar	20	3
32	Armstrong	William	20	4
53	Fagin	Ronald	19	3
107	Bunneman	Peter	18	3

Si on nomme t_1 le premier tuple, alors $t_1[NOM] = Codd$, $t_1[NUM, AGE] = (28, 20)$. L'ensemble de toutes les relations "peuplées" par des tuples constitue la base de données.

2.3 Langages de requêtes et de mises à jour

Plusieurs langages ont été définis pour l'interrogation et la manipulation des données dans le modèle relationnel. *Le résultat d'une requête est toujours une relation* : les langages diffèrent sur la façon de définir la relation en sortie. Ils sont de deux sortes :

- *les langages procéduraux* : cela signifie que, pour obtenir les réponses à sa requête, il faut déterminer où et comment aller rechercher l'information parmi les relations de la base. On définit la procédure devant être exécutée par le programme. Le langage procédural du modèle relationnel est l'algèbre relationnel. Il s'agit d'un ensemble d'opérateurs algébriques unaires et binaires applicables à des relations et retournant des relations : ils peuvent donc être composés pour exprimer le résultat souhaité.
- *déclaratif* : pour obtenir les réponses à sa requête, il faut caractériser précisément le résultat que l'on veut, sans se soucier de la méthode utilisée par le programme pour accéder aux données. C'est grâce à des expressions logiques qu'on détermine le résultat souhaité : c'est le cas du calcul relationnel de tuple (qui manipule des tuples dans des expressions logiques) ou de domaine (qui manipule des variables parcourant les domaines), ou encore du DATALOG (admettant la récursivité).

Le SQL correspond à l'implantation du calcul relationnel de tuple : c'est donc un langage déclaratif, possédant de nombreuses extensions pour faciliter l'usage et augmenter le pouvoir d'expression.

Chapitre 3

Contraintes d'intégrité dans le modèle relationnel

Sommaire

3.1 Généralités

La structure du modèle relationnel définie plus haut permet de modéliser la réalité à un certain niveau de granularité : les attributs vont décrire les objets, et on peut séparer les données dans différentes relations avec des noms explicites. Mais ceci est largement insuffisant pour représenter plus finement les différents aspects des données réelles.

Exemple 4 Supposons le schéma de relation $ETUDIANT$ décrit plus haut. Rien n'empêche l'utilisateur :
– d'insérer plusieurs étudiants avec le même numéro mais des données différentes ;
– d'insérer deux fois le même étudiant mais avec deux âges différents ;
– de mettre un numéro farfelu dans la colonne $FORMATION$, ne référant aucune formation existante.
– As-t-on le droit d'insérer un étudiant dans plusieurs formations ? Il faut un mécanisme pour pouvoir le préciser.
Rappel : en revanche, il est impossible d'avoir deux fois exactement le même tuple dans une relation, puisque une relation est un ensemble (donc sans doublons) de tuples.

Ce pouvoir d'expression est conféré par les *contraintes d'intégrité*, qui sont des expression logique restreignant l'ensemble des relations autorisées dans une base de données. Le but est de pouvoir interdire

tous les cas d'incohérence pouvant être générés lors des mises à jour, pour avoir des données collant toujours à la "logique" de la réalité. En effet, les applications sont souvent développées en utilisant certains postulats sur les données (exemple : le numéro d'étudiant est unique) - mais si ces postulats ne sont pas vérifiés, les applications vont générer des erreurs parfois imprévisibles.

On distingue plusieurs types de contraintes :

- Contraintes statiques ; sont vérifiées dans un état donné de la base
  - les dépendances de données
  - les dépendances de domaine
  - les contraintes de cardinalité
- Contraintes dynamiques ; vérifiées sur deux états différents de la base

Exemple 5

- Chaque étudiant a un numéro unique : cette contrainte est statique, puisse à tout moment, il suffit d'observer la base pour savoir si elle est bien respectée.
- Un étudiant ne peux pas avoir un âge qui décroît : contrainte dynamique, car ne peut être vérifiée que dans deux états successifs de la base.

Les contraintes les plus importantes en conception des bases de données sont les dépendances de données.

3.2 Les principales dépendances de données

Les plus connues et étudiées sont :

- Les dépendances fonctionnelles
- Les dépendances d'inclusion
- Les dépendances de jointure
- Les dépendances multi-valuées

Les deux premières permettent de représenter les situations les plus communes, et sont largement les plus connues, étudiées et utilisées en pratique. Nous allons nous restreindre à elles dans un premier temps - les dépendances de jointure et les dépendances multi-valuées seront présentées plus loin.

3.2.1 Dépendances fonctionnelles et clés

Dans toute présentation d'une classe de dépendance, on distingue :

- La syntaxe : c'est le langage logique autorisé pour définir une contrainte. C'est la "forme" de la contrainte.
- La sémantique : ensemble de conditions devant être remplies pour que la contrainte soit *satisfaite* (c'est à dire juste) dans les données. C'est le *sens* de la contrainte.

La syntaxe des dépendances fonctionnelles est définie ainsi :

Définition 4 (Dépendance fonctionnelle (DF)) Une *dépendance fonctionnelle (DF)* sur un schéma de relation R est une expression de la forme $R : X \rightarrow Y$ (ou simplement $X \rightarrow Y$ lorsque R est implicite), où $X, Y \subseteq R$. Une DF $X \rightarrow Y$ est dite *triviale* si $Y \subseteq X$ et *standard* si $X \neq \emptyset$.

Une DF est donc une contrainte définie sur un schéma. Une DF est dite *valide* sur un schéma R si elle est satisfaite dans toute relation r sur R, selon de la définition suivante pour la satisfaction :

Définition 5 (Satisfaction d'une DF dans une relation) Soit r une relation sur R. Une DF $R : X \rightarrow Y$ est *satisfaite* dans r si $\forall t_1, t_2 \in r$ on a $t_1[X] = t_2[X] \Longrightarrow t_1[Y] = t_2[Y]$.

Si $X \rightarrow Y$ est satisfaite dans une relation r, on dit aussi que X *détermine* Y dans r. De façon intuitive, on définit une DF $X \longrightarrow Y$ pour exprimer le fait que lorsqu'on connaît la valeur de X, alors on peut déterminer (en parcourant la relation) de façon certaine la valeur de Y.

La notion de *clé d'une relation* est très connue par les utilisateurs du modèle relationnel. Une clé peut-être définie de deux manières :

- une clé est un ensemble d'attributs qui ne prend jamais deux fois la même valeur (pas de doublons dans les relations) ;
- A l'aide des DF : une clé est un ensemble d'attributs qui détermine (au sens des DF) tous les autres attributs de la relation.

Ces deux définitions sont rigoureusement équivalentes.

Preuve
Soit X un ensemble d'attributs sur un schéma R sur lequel on interdit les doublons. Ainsi, la DF $X \longrightarrow R - X$ ne peut être ""violée"" dans aucune relation possible sur R, puisqu'il est impossible d'avoir deux tuples qui ont la même valeur sur X. Cela implique donc que la contraite $X \longrightarrow R - X$ est valide sur R. Inversement, supposons que la DF $X \longrightarrow R - X$ est valide sur R. Supposons une relation r sur R dans laquelle deux tuples t_1 et t_2 soient tels que $t_1[X] = t_2[X]$. Dans ce cas, on a également $t_1[R-X] = t_2[R-X]$ par application de la DF, et par conséquence $t_1 = t_2$ - ce qui est impossible puisque r est un ensemble de tuples (donc sans doublon). Donc quelque soit la relation r sur R, il n'y a pas de doublon sur X. □

Une clé primaire est simplement une clé parmi les autres, choisie par le concepteur pour sa simplicité ou son aspect naturel.

3.2.2 Dépendances d'inclusion

Les DI se différencient des DF sur plusieurs points. D'abord, elles peuvent être définies entre attributs de relations différentes, et possèdent ainsi un caractère plus "global" que les DF dont la définition est locale à une relation (bien que l'on parle parfois de *DF globale* [**?**], que nous n'abordons pas ici). Ensuite, les DI sont définies non pas entre deux ensembles quelconques d'attributs, mais *entre deux séquences d'attributs de même taille*. Cette fois, l'ordre des attributs est donc important.

La syntaxe (forme) des DI est la suivante.

Définition 6 (Dépendance d'inclusion) Soit $\mathbf{R}$ un schéma de base de données. Une dépendance d'inclusion sur $\mathbf{R}$ est une expression de la forme $R[X] \subseteq S[Y]$, où $R, S \in \mathbf{R}$, X et Y sont des séquences d'attributs distincts respectivement de R et de S, et $|X| = |Y|$.

Pour ce qui est de la sémantique des DI, l'intuition est la suivante : une DI est satisfaite dans une base de données si toutes les valeurs prises par la partie gauche apparaissent dans la partie droite.

Définition 7 (Satisfaction d'une DI) Soit $\mathbf{d} = \{r_1, ..., r_n\}$ une base de données sur un schéma $\mathbf{R} = \{R_1, ..., R_n\}$. Une dépendance d'inclusion $R_i[X] \subseteq R_j[Y]$ sur $\mathbf{R}$ est *satisfaite* dans $\mathbf{d}$, noté $\mathbf{d} \models R_i[X] \subseteq R_j[Y]$, si $\forall t_i \in r_i, \exists t_j \in r_j \mid t_i[X] = t_j[Y]$ (de manière équivalente, $\pi_X(r_i) \subseteq \pi_Y(r_j)$).

Exemple 6

Supposons des schémas de relation pour décrire les modules :

$$MODULE = \{\underline{NUMMODULE}; INTITULE; DESC\}$$

et un schéma de relation pour décrire les séances de cours :

$$SEANCE = \{\underline{DATE; NUMMODULE}; NUMSALLE\}$$

Pour forcer que les numéros de modules dans les séances soient bien des modules qui existent, on devra alors définir la contrainte :

$$SEANCE[NUMMODULE] \subseteq MODULE[NUMMODULE]$$

Supposons maintenant un schéma de relation EMPRUNTVIDEO modélisant les réservations de vidéos pour les séances, sous la forme :

$$EMPRUNTVIDEO = \{DATE, NUMMODULE; NUMPROF; NUMVIDEO\}$$

Pour assurer la cohérence de la base, on doit préciser que les vidéos doivent être réservés pour des séances existantes dans la base ; on définira alors la DI :

$$EMPRUNTVIDEO[DATE, NUMMODULE] \subseteq SEANCE[DATE, NUMMODULE]$$

On peut remarque l'importance de l'ordre dans les attributs (ce sont des séquences et non des ensembles) : si on inverse les attributs à gauche par exemple, la DI change complètement de signification !

Une *contrainte d'intégrité référentielle* est une DI dont la partie droite est une clé ; la partie gauche d'une contrainte d'intégrité référentielle est appelée *clé étrangère*.

Exemple 7 Les deux DI données dans l'exemple précédent sont des contraintes d'intégrité référentielle, puisqu'on peut supposer que les parties droites sont des clés de leur relation. Les parties gauches sont donc des clés étrangères des relations $SEANCE$ et $EMPRUNTVIDEO$.
En revanche, les DI ne définissent pas toujours des clés étrangères. Pour s'en convaincre, il suffit d'imaginer qu'on souhaite imposer que tous les cours possèdent au moins une séance dans l'année : on définira alors une DI

$$COURS[NUMCOURS] \subseteq SEANCE[NUMCOURS]$$

Ainsi, tous les cours apparaîtront au moins une fois dans la relation des séances ; toutefois, $NUMCOURS$ n'est pas une clé de $SEANCE$ (on imagine difficilement que tous les cours n'aient qu'une seule séance !) et donc ce n'est pas une contrainte d'intégrité référentielle.

3.3 Inférence, fermetures et couvertures de contraintes

Une *classe de contraintes d'intégrité* correspond à un type particulier de contraintes d'intégrité sur un schéma de relation ou de base de données. Par exemple, l'ensemble de toutes les dépendances fonctionnelles sur un schéma de relation R est une classe de contrainte d'intégrité.

L'*inférence de contraintes* est un concept fondamental pour la théorie des bases de données ; cela consiste à considérer les contraintes étant impliquées par d'autres contraintes. Les contraintes impliquées sont tout aussi valides que les contraintes de départ, mais souvent de façon implicite : elles échappent alors à la connaissance du concepteur. Le paragraphe suivant introduit ces notions importantes en prenant comme exemple les DF.

3.3.1 Inférence de dépendances fonctionnelles

Définition 8 (Implication de DF) Soit F un ensemble de DF sur un schéma de relation R, et f une DF sur R. On dit que F implique f, noté $F \models f$ si pour toute relation r sur R telle que $r \models F$, on a $r \models f$.

Exemple 8 Soit le schéma

$$ETUDIANT(NUMETUD, NOMETUD, NOMVILLE, REGION, CP)$$

et l'ensemble de DF sur ce schéma :

$$\{NUMETUD \longrightarrow NOMETUD, NOMVILLE, REGION, CP; NOMVILLE, REGION \longrightarrow CP\}$$

On comprend intuitivement que les deux DF suivantes seront également valides, par implication :

$$\{NUMETUD \longrightarrow NOMVILLE, REGION; NUMETUD \longrightarrow CP\}$$

Si ces implications sont intuitives, il est toutefois difficile et fastidieux de prouver que chaque intuition est bien juste ! Et surtout il est impossible d'être certain qu'on a bien pensé à toutes les DF impliquées par notre ensemble de DF de départ. Pour palier ces difficultés, et permettre un traitement automatique des contraintes, on définit la notion de *règles d'inférences.*

Définition 9 (Règle et système d'inférence) Une règle d'inférence est une expression de la forme $F \Rightarrow f$. Un système d'inférence est un ensemble de règles d'inférence. Si S est un système d'inférence, on note $F \vdash_S f$ le fait qu'on puisse dériver f à partir de F par des applications successives de règles de S.

Le système d'inférence des DF est le *système d'Armstrong.*

Définition 10 (Système d'inférence d'Armstrong) Soit F un ensemble de DF sur un schéma de relation R. Les règles d'inférence de DF suivantes sont appelées *système d'inférence d'Armstrong* :
- Réflexivité : si $Y \subseteq X \subseteq R$ alors $F \vdash X \rightarrow Y$.
- Augmentation : si $F \vdash X \rightarrow Y$ et $F \vdash W \subseteq R$ alors $F \vdash WX \rightarrow WY$.
- Transitivité : si $F \vdash X \rightarrow Y$ et $F \vdash Y \rightarrow Z$ alors $F \vdash X \rightarrow Z$

Si ce système d'inférence est important pour les DF, c'est qu'il possède les propriétés suivantes :
- Il est juste, soit $F \vdash f \Longleftrightarrow F \models f$;

– Il est complet, soit $F \models f \Longleftrightarrow F \vdash f$;
– Il est minimal en nombre de règles d'inférence.

Une conséquence de ce résultat est que $F \models \alpha \Leftrightarrow F \vdash \alpha$; nous utiliserons dans la suite la notation $F \models \alpha$.

Ainsi, le système d'Armstrong permet de systématiser la recherche de toutes les DF impliquées par d'autres DF. Rigoureusement, on appelle *problème d'inférence des DF* le problème suivant : Soit F un ensemble de DF, et f une DF, a-t-on $F \models f$? La résolution de ce problème est "facile" pour les DF, et plus précisément linéaire dans le nombre de DF en entrée.

Soit F un ensemble de DF, on note F^+ (fermeture de F) l'ensemble de toutes les contraintes f telles que $F \models f$.

Enfin, si X est un ensemble d'attribut et F un ensemble de DF, on appelle fermeture de X par rapport à F l'ensemble de tous les attributs qu'on peut "déduire" de X par des dépendances fonctionnelles. Naturellement, il faut aussi tenir compte des DF qui ne sont pas dans F, mais qui sont dérivables à partir de F. Donc :

$$X^+ = \{A \mid F \models X \longrightarrow A\}$$

ou encore

$$X^+ = \{A \mid X \longrightarrow A \in F^+\}$$

Remarque : On peut donner une définition alternative de la notion de clé d'une relation : X est clé d'un schéma R si et seulement si $X^+ = R$

3.3.2 Inférence de dépendances d'inclusion

Le système d'inférence pour les DI est le suivant :

Définition 11 (Système d'inférence de Casanova et al.) Soit I un ensemble de DI sur un schéma de base de données $\mathbf{R}$. Les règles d'inférence de DI suivantes sont appelées *système d'inférence de Casanova et al.* :
– Reflexivité : $I \vdash R[X] \subseteq R[X]$.
– Projection et permutation :
si $I \vdash R[A_1...A_n] \subseteq S[B_1...B_n]$ alors $I \vdash R[A_{\sigma 1}...A_{\sigma m}] \subseteq S[B_{\sigma 1}...B_{\sigma m}]$ pour chaque séquence $\sigma 1...\sigma m$ d'entier distincts dans $\{1...n\}$
– transitivité : si $I \vdash R[X] \subseteq S[Y]$ et $I \vdash S[Y] \subseteq T[Z]$ alors $I \vdash R[X] \subseteq T[Z]$

Ce système d'inférence est lui aussi juste, complet et minimal pour les dépendances d'inclusion. Contrairement aux DF, le problème d'inférence des DI est PSPACE-complet, donc infaisable dans le cas général.

La notion I^+ s'applique aussi pour noter la fermeture d'un ensemble de DI. En revanche, la notion de fermeture d'un ensemble d'attributs par rapport à un ensemble de DI ne s'applique pas, car les DI manipulent des séquences et non des ensembles.

3.3.3 Couvertures des DF et des DI

La notion de couverture est une relation d'équivalence entre des ensembles de contraintes.

Définition 12 (Couverture d'un ensemble de DF) Soit Σ et Γ deux ensembles de contraintes. Γ est une couverture de Σ si $\Gamma^+ = \Sigma^+$.

Une couverture d'un ensemble de DF est donc une représentation alternative, avec d'autres DF, mais véhiculant une sémantique rigoureusement équivalente : au final, c'est exactement le même ensemble de DF qui est implicite. C'est exactement la même chose pour les DI.

Trois propriétés sont importantes pour les couvertures des DF - peu d'études existent pour les DI.

- un ensemble F de DF est dit *non redondant* s'il n'existe pas de couverture G de F telle que $G \subset F$.
- un ensemble F de DF est dit *minimum* s'il n'existe pas de couverture G de F tel que $|G| \leq |F|$.
- F est dit *optimum* s'il n'existe pas de couverture G de F avec moins d'attributs que dans F.

Remarque : une couverture minimum est non redondante, une couverture optimum est minimum.

L'algorithme suivant calcule une couverture minimum pour un ensemble F de DF.

Require: F un ens de DF
Ensure: G une couverture minimum de F
  $G := \emptyset$;
  for all $X \longrightarrow Y \in F$ **do**
    $G := G \cup \{X \longrightarrow X^+\}$;
  end for
  for all $X \longrightarrow X^+ \in G$ **do**
    if $G - \{X \longrightarrow X^+\} \vdash X^+$ **then**
      $G := G - \{X \longrightarrow X^+\}$;
    end if
  end for
  Return G.

Cet algorithme est polynomial dans le nombre de DF dans F et le nombre d'attributs dans F. La couverture minimum calculée par l'algorithme n'est pas forcément unique : d'autres couvertures peuvent avoir le même nombre de DF, mais être différentes. Parmi celles-ci, certaines sont optimum ; malheureusement, leur calcul est un problème "NP-Complet" dans le cas général.

Chapitre 4

Conception de bases de données

Sommaire

Nous entendons par conception, dans ce cours, le fait de choisir une modélisation des données réelles à partir du cahier des charges des applications. La modélisation est en relationnel : il s'agit donc de déterminer l'ensemble des attributs, des relations et des contraintes qui constitueront le modèle.

Nous allons voir dans la suite quelles sont les propriété attendues d'une bonne modélisation en relationnel, et comment les obtenir.

4.1 Anomalies de mise à jour et redondance

Une anomalie de mise à jour à lieu lorsque, à la suite d'une modification de la base, des contraintes sémantiques valides se trouvent violées. Bien sûr, des mécanismes de contrôle sont intégrés aux SGBDR pour éviter ce genre de problèmes. Mais celà suppose :

- une perte de temps dans la gestion de la base, certains contrôles pouvant être assez lourds ;
- une implémentation rigoureuse de toutes les contraintes par le concepteur de la base. Sous Oracle, celà passe bien souvent par la mise en place de "Trigger" en PL/SQL.

Le compromis est alors atteint en faisant l'hypothèse suivante : le concepteur n'implémente que les clés et les clés étrangères. Le contrôle automatique de ces contraintes est peu coûteux par le SGBD, et leur implémentation est toujours intégrée dans les SGBDR. Ainsi, on considère que toute mise à jour respecte les clés.

R a une anomalie de mise à jour par rapport à F s'il est possible d'insérer ou de modifier un tuple t tel que :
- $r \cup t \models CLE(F)$, où $CLE(F)$ est l'ensemble des clés induites par F.
- $r \cup t \not\models F$.

Exemple 9 Supposons le schéma de relation

$$ETUDIANT(NUMETUD - A, NOM - B, VILLE - C, CP - D, DPT - E)$$

muni de l'ensemble de DF

$$A \longrightarrow BCD, CD \longrightarrow E$$

La seule clé minimale de la relation est donc A (Toutes les autres clés sont des sur-ensembles de A). Supposons qu'un tuple soit inséré pour un nouvel étudiant, avec une ville et un CP déjà présent mais un département différent. La clé ne sera pas violée (pas de doublon sur A) mais la DF $CD \longrightarrow E$ ne sera plus satisfaite. Puisqu'un tel cas de figure est possible, la relation $ETUDIANT$ possède une anomalie de mise à jour.

Une suppression ne peut entraîner aucune anomalie proprement dite ; toutefois, elle peut engendrer une perte involontaire d'information, lié à la redondance dans les données. Dans l'exemple précédent, si on supprime tous les étudiants de LYON, on aura également perdu le code postal et le département de Lyon, même si on souhaitais garder cette information.

La notion de *redondance* est une autre façon de considérer les problèmes de mises à jour. Elle se définit sur les relations, alors les problèmes de mise à jour portent sur des schémas.

Définition 13 (U) ne relation r sur R est redondante par rapport à un ensemble F de DF sur R si :
1. $r \models F$ et
2. il existe $X \longrightarrow A \in F$ et $t_1, t_2 \in r$ tels que $t_1[XA] = t_2[XA]$.

Exemple 10 Reprenons l'exemple précédent, et considérons cette fois la relation suivante sur le schéma $ETUDIANT$:

NUMETUD	NOM	VILLE	CP	DPT
1	Fagin	Lyon	69003	Rhône
2	Armstrong	Lyon	69001	Rhône
3	Bunneman	Clermont	63000	Puys-de-Dôme
4	Codd	Lyon	69001	Rhône

Cette relation est bien correcte car elle respecte les DF. Néanmoins, elle est redondante car il existe un doublon sur $(Ville, CP)$: l'information du département de LYON 1er apparaît deux fois.

On voit bien que les notions d'anomalie de mise à jour et de redondance sont très liées - elles sont en fait équivalentes, selon le résultat suivant.

Théorème 1 *Il y a équivalence entre :*
- *R a une anomalie de mise à jour par rapport à F,*

– *Il existe une relation r sur R qui est redondante par rapport à F.*

4.2 Les pertes d'information

Le principe de base est alors de décomposer les relations de telle sorte d'éviter d'avoir ces anomalies, c'est à dire de transformer une relation en plusieurs relations. Avec des schémas de relation à un seul attribut, il n'y a plus aucun problème de redondance ! Le risque en décomposant est de perdre de l'information.

4.2.1 Perte de données

Tout d'abord, il faut que toutes les informations de la base de donnée initiale puissent être retrouvées en effectuant des jointures sur les relations issues de la décomposition.

Soit R un schéma de relation (c'est à dire un ensemble d'attributs), que l'on décompose en un schéma de base de données (un ensemble de relations) $\mathbf{R} = \{R_1, \ldots, R_n\}$. On dira alors que $\mathbf{R}$ est *sans perte de jointures* par rapport à un ensemble F de dépendances fonctionnelles si, pour toute relation r sur R telle que $r \models F$ on a :

$$r = \pi_{R_1}(r) \ldots \pi_{R_n}(r)$$

Exemple 11 Reprenons la relation de l'exemple précédent. Supposons que pour régler le problème de redondance, on découpe le schéma en deux de façon à obtenir les relations suivantes :

NUMETUD	NOM
1	Fagin
2	Armstrong
3	Bunneman
4	Codd

VILLE	CP	DPT
Lyon	69003	Rhône
Lyon	69001	Rhône
Clermont	63000	Puys-de-Dôme

4.2.2 Perte de DF

Ensuite, il ne faut pas que la décomposition ait "coupé" des dépendances fonctionnelles, ce qui conduirai à une perte inévitable de sémantique. Pour cela, on définit la notion de projection d'un ensemble de dépendances fonctionnelles :

Définition 14 (projection d'un ensemble de DF) Soit F un ensemble de DF sur R, et S un schéma de relation tel que $S \subseteq R$.

$$F[S] = \{X \longrightarrow Y \mid X \longrightarrow Y \in F \text{ et } XY \subseteq S\}$$

La projection sur un schéma de bases de données est l'union des projection sur chaque relation du schéma.

Soit R un schéma de relation et F un ensemble de DF sur R. Un schéma de relation $\mathbf{R}$ est une *décomposition qui préserve les dépendances* de R par rapport à F si :

$$F[\mathbf{R}]^+ = F^+$$

4.3 Les principales formes normales

Les formes normales sont des propriétés que doivent vérifier les schémas pour éviter les anomalies de mises à jour. Plusieurs formes normales ont été définies. Une forme normale s'applique à un schéma de relation, en fonction d'un certain ensemble de contraintes d'une classe donnée. Concernant les DF, l'idée générale est de n'avoir que les clés à vérifier, et d'éliminer au maximum des DF qui ne définissent pas des clés. Dans la suite, soit R un schéma de relations et F un ensemble de DF définies sur R.

Rappel : en relationnel, on est toujours en première forme normale, soit toutes les valeurs des attributs sont atomiques.

Définition 15 (2FN) R est en deuxième forme normale par rapport à F si, pour chaque DF $X \longrightarrow A$ de F, l'une des deux conditions suivantes est remplies :
- A appartient à une clé de R,
- X n'est pas un sous-ensemble propre d'une clé de R

Donc, aucun attribut (en dehors de ceux qui forment les clés), ne sont déterminés par des sous-ensembles des clés.

Par contre, même en dehors des clés, certains attributs peuvent être déterminés par des ensembles qui ne sont pas des sous-ensembles des clés. Ce qui est exclu dans la troisième forme normale.

Définition 16 (3FN) R est en troisième forme normale par rapport à F si, pour chaque DF $X \longrightarrow A$ de F, l'une des deux conditions suivantes est remplies :
- A appartient à une clé de R,
- X est une clé (ou une superclé)

Ainsi, les seules DF "parasites" autorisées dans la troisième forme normale sont celles dont la partie droite est un attribut membre d'une clé.

Enfin, la forme normale de Boyce-Codd impose que toutes les parties gauches des DF sont des clés.

Définition 17 (FNBC) R est en forme normale de Boyce-Codd par rapport à F si, pour chaque DF $X \longrightarrow A$ de F, X est une superclé de R.

Remarques :
- Si toutes les clés d'une relation sont composées d'un seul attribut, alors la troisième forme normale est équivalente à la forme normale de Boyce-Codd.
- Si une relation n'est composée que d'un ou deux attributs, elle est en FNBC.

La FNBC est, en ce qui concerne les DF, la forme idéale d'un schéma de bases de données. En effet, les trois propriétés suivantes sont équivalentes :
- R est en FNBC par rapport à F ;
- R n'a pas de problème de redondances par rapport à F ;
- R n'a pas de problème de mise à jour par rapport à F ;

Ajoutons que la troisième forme normale est toujours possible, quelque soit les DF considérées - la deuxième forme normale n'a donc qu'un intérêt "historique". En revanche, il existe des situations où la FNBC n'est pas possible.

Exemple 12 Un exemple type : une rencontre sportive entre équipes, dans laquelle chaque équipe présente un seul athlète dans chaque épreuve ; le même athlète peut faire plusieurs épreuves. On a donc $(Equipe, Epreuve, Athlte)$ avec les DF :

$$(Equipe, Epreuve \longrightarrow Athlte; Athlte \longrightarrow Equipe)$$

Quelle que soit la décomposition sans perte réalisée, elle n'est pas en FNBC (essayer toutes les décompositions imaginables à partir de ces trois attributs)

Dans un tel cas, il faut soit :
- Accepter de ne pas être en BCNF, et donc accepter d'avoir une anomalie de mise à jour. En d'autre termes, on ne pourra pas se contenter de déclarer les clés, mais il faudra implémenter les DF (Trigger sous un SGBD, ou au niveau de l'application) qui ne sont pas des clés.
- Accepter de relâcher des contraintes, c'est à dire enlever quelques DF et rendre l'application "plus souple".
- On peut aussi rajouter des attributs et des DF. Dans l'exemple, on peut rajouter un attribut qui "regroupe" les couples Equipe/Epreuve. On obtiendrait :

  $$(Equipe, Epreuve, N^{\circ}Equipe/Epreuve, Athlte)$$

  ainsi que les DF

  $$(Equipe, Epreuve \longrightarrow N^{\circ}Equipe/Epreuve)$$

  $$(N^{\circ}Equipe/Epreuve \longrightarrow Equipe, Epreuve, Athlte)$$

  et

  $$Athlte \longrightarrow Equipe$$

  La décomposition en FNBC est alors possible, et aucune sémantique n'est perdue.

4.3.1 Au delà des dépendances fonctionnelles

Les dépendances fonctionnelles nous ont permis jusque-là de mettre en évidence une forme de redondance, que les formes normales cherchent à faire disparaître. Mais des cas de redondance ne sont pas capturés par les DF.

Exemple 13
Supposons l'énoncé suivant : "les étudiants suivent des parcours, et sont inscrits dans des transversales indépendantes du parcours. Chaque étudiant peut être inscrit à plusieurs parcours". Soit la modélisation suivante :

$$R(etudiants, parcours, transversale)$$

On ne peut dégager aucune DF dans ce schéma, donc la seule clé est la combinaison des trois attributs.

La relation de l'exemple précédent est donc en FNBC, puisqu'aucune DF n'est valide. On se rend bien compte pourtant de la redondance présente dans ce schéma, car il faudra, pour chaque étudiant, répéter tous les parcours à chaque transversale suivie. En effet, si on ne le fait, on risque d'induire une dépendance entre les parcours et les transversales qui n'est pas vraie.

La notion qui permet de modéliser cette redondance est celle de dépendance multivaluée. De façon intuitive, une dépendance multivaluée sert à modéliser que deux informations sont liées "indépendamment" des autres informations.

On notera :

$$etudiants->->parcours|transversale$$

pour signifier que "une étudiant est associé à plusieurs parcours et plusieurs transversales, indépendamment". Une telle contrainte est satisfaite dans la relation r si $xyz\ xy'z' \in r$ implique $xy'z$ et $xyz' \in r$.

Ainsi, on peut décomposer la relation est deux relations

$$R_1(etudiants, parcours); R_2(etudiantstransversale)$$

L'information pourra être retrouvée par jointure : en d'autres termes, on peut décomposer car la jointure reconstruira exactement la relation initiale (sans tuple supplémentaires).

On dira alors que R est en Quatrième forme normale, ce qui correspond à
- R est en troisième forme normale
- les seules dépendances multivaluées sont du type $X->->R-X$. Ou encore, R ne doit pas être décomposable en deux relations sans perte de jointure.

Remarque : 4FN $\Rightarrow$ 3FN $\Rightarrow$ 2FN.

4.4 Comment normaliser une relation

Un algorithme de normalisation prend en entrée une relation et des contraintes, et construit un schéma de BD normalisé, en 3FN ou en BCNF. Il existe deux grandes catégories d'algorithmes de normalisation : les algorithmes *de décomposition* et les algorithmes *de synthèse*. Nous ne verrons ici que les algorithmes de synthèse.

4.4.1 Algorithmes de synthèse

Entrée : une relation universelle (l'ensemble de tous les attributs du problème) et un ensemble de contraintes : DF, DI, DMV.

On commence par répertorier tous les attributs et construire ainsi la relation universelle de notre application. Puis on dresse l'inventaire des contraintes DF, DI, DMV.

Principe général
- Construire une couverture minimum de F, et réduire les parties gauches et droites au maximum
- Générer une relations XY pour chaque DF $X \longrightarrow Y$;
- Générer une relations XY' pour chaque DMV $X->->Y$ avec $F \models Y' \longrightarrow Y$;
- On supprime les schémas de relation qui ne sont pas maximaux par inclusion.
- S'il y a perte de jointure, alors on rajoute une relation composée d'une clé de F.

A partir de cette base, de nombreuses variantes incluent des heuristiques pour diminuer la redondance en sortie.

Propriétés : l'algorithme finit toujours, en donnant la meilleure forme normale (FNBC si elle existe, 4FN sinon)

L'algorithme suivant est utilisé pour réduire les parties gauches et droites des DF d'une couverture minimum. Attention, cet algorithme ne calcule pas une couverture *optimum*, qui contiendrait le nombre minimal d'attributs (qui est un problème NP-Complet). Ici, le nombre d'attribut est simplement minimal pour l'ensemble de DF pris en entrée.

Algorithm 1 Réduction du nombre d'attribut pour un ensemble de DF

Require: Un ensemble *minimum* de DF F sur R.
Ensure: F avec un nombre minimal d'attributs

1: $Min := F$
2: *//Réduction des parties gauches*
3: **for all** $X \longrightarrow Y \in Min$ **do**
4: $W := X$;
5: **for all** $A \in X$ **do**
6: **if** $Min \models (W - A) \longrightarrow X$ **then**
7: $W := W - \{A\}$;
8: **end if**
9: **end for**
10: $Min := (Min - \{X \longrightarrow Y\}) \cup \{W \longrightarrow Y\}$;
11: **end for**
12: *//Réduction des parties droites*
13: **for all** $X \longrightarrow Y \in Min$ **do**
14: $W := Y$;
15: **for all** $A \in Y$ **do**
16: $G := (Min - \{X \longrightarrow Y\}) \cup \{X \longrightarrow (W - A)\}$;
17: **if** $G \models X \longrightarrow Y$ **then**
18: $W := W - \{A\}$;
19: **end if**
20: **end for**
21: $Min := (Min - \{X \longrightarrow Y\}) \cup \{X \longrightarrow W\}$;
22: **end for**
23: **Retourner** Min ;

Exemple 14
On gère une liste de projets. Chaque projet a un responsable, un ensemble d'employés, et utilise certains produits en une quantité donnée. Pour un produit et un projet, plusieurs fournisseurs à des cout différents sont concernés. Un fournisseur a plusieurs adresses. Finalement, un employé a une date d'embauche et un salaire.
Soit $(Projets, produits, fournisseur, adresse, cout, responsable, employs, salaire, dateEmbauche)$.
Les DF sont :

$$Projet, produit \longrightarrow quantite; Projet, fournisseur, produit \longrightarrow cot$$

$$projet \longrightarrow responsable; employe \longrightarrow salaire, dateEmbauche$$

et on a deux MVD :

$$Fournisseur- > - > Location; projet- > - > employe, salaire, dateEmbauche$$

Après application de l'algorithme, on obtient les relations suivantes :
- $(Projet, Produit, Qute)$,
- $(Projet, Fournisseur, produit, cout)$,
- $(Projet, responsable)$,
- $(employe, salaire, dateEmbauche)$,
- $(fournisseur, location)$,
- $(projet, employe)$.

4.5 Convertir un schéma Entité/Association en schéma relationnel

4.5.1 Rappel : Le modèle Entité-Association

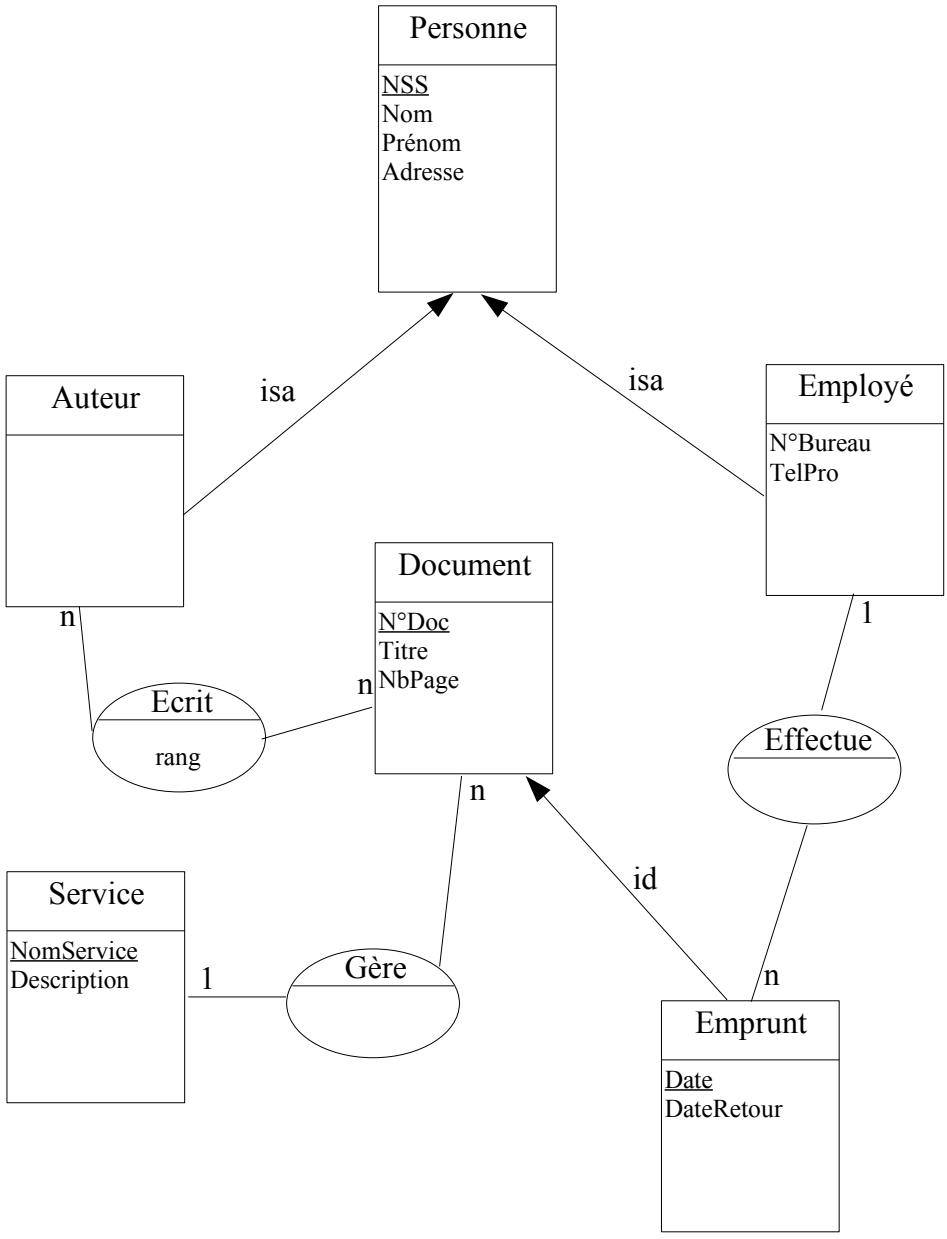
Le modèle Entité-Association est un modèle de données incomplet, dans le sens où il ne fournit aucun moyen de manipuler ou d'interroger les données. Sa popularité provient du fait qu'il permet de représenter *la sémantique* des données d'une application d'une manière relativement simple, intuitive pour un non spécialiste. Il facilite alors la tâche du concepteur d'une base de données, et possède l'avantage de parfaitement distinguer la spécification de la sémantique des données de l'implantation physique de la base, et plus généralement du modèle de données qui sera choisi par la suite pour la gestion effective de la base.

La part la plus importante et la plus populaire fournie par le modèle EA est le *diagramme entité-association*, qui permet une représentation visuelle (relativement...) intuitive des données. On distingue deux types d'*objets* différents : les entités et les associations entre entités. Plusieurs propositions ont été faites concernant les représentations graphiques des entités et associations ; celles utilisées ici sont une possibilité parmi d'autres, l'essentiel étant de comprendre les concepts abordés.

Les entités

Une **entité** est une "chose" qui peuple les données à représenter. Dans l'exemple, une entité peut être un employé donné, ou un service, ou encore un auteur. Une entité appartient à un ou plusieurs **types d'entité**. Par exemple, une personne de l'entreprise peut être à la fois un employé et un auteur.

Un type d'entité est donc une collection d'entités ; il est décrit par un nom unique, et par une série d'*attributs*. Parmi les attributs, on distingue :
- Les attributs mono-valués : chaque entité du type entité concerné prend au plus une valeur pour cet attribut. Par exemple, un document possède un seul titre, un seul numéro d'identification, une seule date de parution.
- Les attributs multi-valués : pour une entité donné, un tel attribut peut prendre plusieurs valeurs. Par exemple, un document peut avoir plusieurs mots-clés, ou un employé peut avoir un ou plusieurs prénoms.



Exemple de diagramme
Entité/Association

Certains attributs peuvent être des *clés* de leur entité, c'est à dire qu'ils identifient de façon unique chaque entité d'un type d'entité donné. Par exemple, le nom d'un service, ou le numéro NSS d'un employé sont des clés. Chaque type d'entité doit avoir au moins une *clé primaire*, qui une clé choisie parmi les autres. Graphiquement, la clé primaire sera soulignée pour la distinguer des autres attributs.

Les associations

Un *type d'association* est une association entre plusieurs types d'entités. nous nous limiterons ici aux types d'associations binaires, c'est à dire des types d'associations entre deux types d'entités. Par exemple, ECRIT est un type d'association entre les types d'entité DOCUMENT et AUTEUR. Un type d'association correspond donc à un ensemble d'*associations* qui peuplent les données. Une association de type ECRIT est donc une paire particulière composée d'une entité de type document et d'une entité de type auteur.

Un type d'association peut être décrite par des attributs. Sur le type d'association ECRIT, l'attribut *rang* dénote, pour une association composée d'un auteur et d'un document, quelle est la place de l'auteur parmi l'ensemble des auteurs de ce document (premier auteur, deuxième auteur, etc...). Une association est identifiée à partir des valeurs de clés primaires des entités participantes.

Les types d'associations peuvent être classés en trois catégories, suivant le nombre de fois qu'une entité donnée peut participer à une association.

- *Les types d'associations* $n-n$(ou plusieurs à plusieurs). Une entité peut alors participer à plusieurs associations de ce type. Exemple : l'association ECRIT entre AUTEUR et DOCUMENT est une association n-n ; un auteur particulier peut écrire plusieurs documents, et un document peut-être écrit par plusieurs auteurs.
- Les types d'association $1-n$ (ou un à plusieurs). Les deux types d'entités impliqués ne jouent pas alors des rôles symétriques. Une entité particulière du premier type pourra participer à plusieurs associations. Une entité particulière du deuxième type ne pourra participer qu'à au plus une seule association.

  Exemple 15 Sur l'exemple, le type d'association MEMBRE, qui associe des entités de type SERVICE avec des entités de type EMPLOYEE, est une association $1-n$. Un employé ne peut être associé qu'à un seul service, alors qu'un service peut être associé à plusieurs employés.

- *Les types d'association* $1-1$. Toute entité ne peut participer qu'une seule fois à une association de ce type.

Remarque - les cardinalités 1 et n se lisent "au plus 1" ou "au plus n". Il s'agit de cardinalités *maximales* ; une entité peut ne participer à aucune association. Par exemple, on peut avoir des documents qui n'ont pas d'auteurs, des employés qui n'ont pas de service. Notons qu'il est toutefois possible de contraindre des entités à participer à certaines associations, ce que nous ne détaillerons pas ici.

Les associations existentielles, ou entités faibles

Un type d'entité faible est un type d'entité particulier : l'existence d'une entité de type faible dépend de l'existence d'une autre entité. Le lien entre une entité faible et l'entité forte correspondante est appelé association existentielle. Nous en distinguerons deux sortes :

Associations ISA Elles permettent de coder la notion d'héritage entre types d'entités. Chaque entité du type faible est complètement identifié par une entité du type fort correspondant. Par exemple, un

employé est une personne particulière. Il en possède toutes les caractéristiques, ainsi que des propriétés supplémentaires comme le numéro de bureau ou la date d'embauche. Notons que l'entité faible et l'entité forte apportent des informations différentes sur un seul et même objet conceptuel.

Graphiquement, ce lien est représenté par une flèche étiquetée "ISA".

Associations ID Dans ce cas, l'entité faible et l'entité forte ne correspondent pas au même objet, mais il existe un lien de "subordination" entre les deux. L'entité faible a tout simplement besoin de l'existence de l'entité forte. Sur l'exemple, le type d'entité EMPRUNT est faible. L'existence d'un emprunt particulier dans la base de données est dépendante de l'existence du document correspondant. Ce lien de dépendance se traduit par le fait que la valeur de la clé de l'entité "dominante" se retrouve parmi la clé de l'entité faible. Dans l'exemple, la clé d'un emprunt est donnée par la clé du document correspondant, à laquelle on ajoute la date de l'emprunt. Muni de ces deux caractéristiques, un emprunt est parfaitement identifié.

Graphiquement, le lien entre un type d'entité faible et le type d'entité auquel il est rattaché est noté "ID".

4.5.2 Traduction E/A vers relationnel