Quatrième partie

Niveau logique

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Chapitre 9

Modélisation logique des traitements

La modélisation logique des traitements permet de spécifier la manière dont les différents traitements décrits au niveau organisationnel vont être effectivement mis en œuvre sur le matériel. Le MOT se préoccupait d'une vision externe des traitements, alors que le MLT décrit par l'intérieur les moyens utilisés par l'informaticien pour réaliser les activités prévues. On devra tenir compte, à ce niveau, des contraintes logicielles et matérielles et des principes d'ergonomie (tant d'un point de vue général que du point de vue du respect des habitudes générales et propres à l'entreprise).

Malheureusement, il n'existe pas de normalisation unanimement reconnue pour la modélisation de ce niveau[1].

9.1 Formalisme MLT

La méthode Merise propose l'utilisation de plusieurs concepts, afin de faciliter la description du MLT :

- Machine logique
- Événement/résultat-message
- État
- Unité Logique de Traitement
- Procédure logique

9.1.1 Machine logique

Le concept de machine logique s'oppose bien entendu à celui de machine physique. Cette dernière ne nécessite pas d'explications longues. Tout au plus peut-on éprouver parfois une difficulté à en situer exactement les limites (une imprimante est-elle une machine physique ou fait-elle partie intégrante d'une machine comprenant aussi d'autres ressources ?). Une machine logique peut dans les cas simples s'identifier à une machine physique, en sachant qu'elle peut figurer en plusieurs exemplaires, qui sont alors fonctionnellement équivalents. Dans le cas d'une terminal passif (ou d'un PC émulant un terminal), une machine logique sera composée de plusieurs machines physiques. Il y a aussi les machines virtuelles qui peuvent constituer une partie seulement d'une machine physique[2].

[1] On peut bien sûr citer le cas de UML (Unified Modeling Language), qui reste toutefois limité au contexte de la programmation objet.

[2] L'exemple le plus surprenant est celui des machines virtuelles VMWare qui permettent, sur un micro, de simuler sur une même machine physique deux machines logiques (voire plus) qui peuvent éventuellement collaborer.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

9.1.2 Événement/résultat-message

On retrouve en fait les événements déclencheurs et les résultats produits déjà évoqués au MCT et au MOT, mais cette fois, ils ne sont plus liés à des acteurs externes, mais provoqués par l'interaction des utilisateurs du logiciel ou par d'autres composants du système (minuteries ou démons, communications entre machines...). Il n'est donc plus nécessaire de faire figurer des acteurs à ce niveau.

9.1.3 État

Les états peuvent être considérés comme faisant partie des données et, à ce titre, ne pas figurer dans le MLT. Dans certains cas cependant, il paraît plus sérieux de représenter des diagrammes qui mettent en évidence les différents états par lequel un élément modélisé peut passer (un dossier est actif, suspendu, en attente...). Dans ce cas, le diagramme permettra de montrer les différentes opérations qui font changer l'élément d'état.

9.1.4 Unités Logiques de Traitement (ULT)

Une unité logique de traitement consiste en un ensemble de traitements informatiques perçus comme homogènes en terme de finalité. Cela suppose donc une certaine cohérence au niveau de l'environnement : on suppose qu'on part d'un état cohérent des données et l'ULT propose une série d'opérations au terme desquelles, l'environnement est de nouveau dans un état stable. Toute interruption de l'ULT serait en principe problématique. Cela permet, par exemple, de modéliser une transaction, une boîte de dialogue, une édition, un module de traitement par lots. On utilise pour modéliser les ULT le formalisme vu pour les opérations et les tâches. Différents effets de zoom peuvent intervenir dans la représentation, selon le niveau d'analyse où on se situe. Une ULT peut se voir décrite à travers une série de spécifications qui épuisent ses caractéristiques :

- présentation
- logique de dialogue
- logique fonctionnelle
- règles de calcul
- sous-schéma logique des données
- enchaînements

présentation : elle se présente sous la forme d'un écran, d'un formulaire, d'un état ou d'une maquette qui permet de voir la manière dont les informations sont disposées à l'écran et mises à la disposition de l'utilisateur.

logique de dialogue : elle prévoit l'ensemble des règles qui vont gouverner l'interaction entre l'utilisateur et l'interface (actions sur le clavier et la souris). On spécifiera donc les réactions du programme, les modifications d'écran qui en résulteront (affichage de résultats de calculs élémentaires, aide, messages divers...). On exclura ici tout algorithme et tout accès aux données.

logique fonctionnelle : c'est la colonne vertébrale de l'ULT. Elle décrit le fonctionnement et l'enchaînement des traitements internes et règle les échanges entre les différents composants de l'ULT que nous sommes en train d'examiner.

règles de calcul : on ne tiendra pas compte ici de petits calculs déjà prévu au niveau de la logique de dialogue. On entend donc ici des calculs faisant l'objet d'un traitement plus

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

complexe ou sans rapport direct avec la présentation. On trouvera donc, dans un approche classique, des variables, des constantes liées au contexte, des algorithmes reprenant des alternatives, des boucles et des blocs.

sous-schéma logique des données : on retrouve ici les schémas partiels du MOT, mais réduits à ce qui est strictement nécessaire pour le fonctionnement de l'ULT. Ce sera un schéma extrait du MLD (table, ou requête sur des tables). On précisera également les conditions d'utilisation de ces données (création, consultation, modification et destruction).

enchaînements : il faut préciser la manière dont l'ULT sera appelée par d'autres (conditions préalables, paramétrage éventuel) et la façon dont l'ULT peut faire appel à d'autres, soit pour en utiliser les résultats, soit pour leur passer la main.

9.1.5 procédure logique

Une procédure regroupe un ensemble d'ULT pour réaliser une tâche complexe ou une phase du modèle organisationnel. Elle décrit et précise la logique de tout ce qui est compris entre le lancement d'une fonction de l'application (par exemple, choix dans un menu) et le retour au même contexte, avec possibilité de choisir une autre fonction. Cet aspect ne doit être explicitement décrit que si un grand nombre d'ULT interviennent dans la procédure.

9.2 Conception des MLT

9.2.1 trois approches de conception

On peut distinguer trois approches pour la définition des ULT, qui ont chacune des avantages et des inconvénients. A telle enseigne qu'il semble prudent de ne pas se limiter à une approche trop figée, faute de risquer de s'enfermer dans une démarche aboutissant à une décomposition trop peu conviviale.

Décomposition des tâches organisationnelles

La première démarche consiste à partir des différentes tâches décrites par le MOT et à définir des ULT dont la succession produit le résultat escompté à ce niveau.

[+] les ULT sont proches des conditions d'utilisation exprimées dans le MOT

[-] il faut que les procédures organisationnelles soient bien établies

[-] toute modification organisationnelle risque de rendre le logiciel inutilisable (manque de souplesse)

[-] on risque de ne pas voir que certains ULT similaires et identiques peuvent s'envisager à différents niveaux[3]

Réutilisation des ULT

Dérivant directement du génie logiciel, cette démarche préconise la définition et la réutilisation d'ULT standardisées, réalisant des tâches pouvant s'intégrer à différentes procédures organisationnelles, voire même différentes applications.

[+] On espère voir diminuer le nombre d'ULT et s'épargner du travail à l'avenir.

---

[3]Voir les dépendances stéréotypées de type `include` dans UML.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

[-] Cette réutilisation entraîne la volonté de gommer des différences plus ou moins ténues.

[-] Le fait que l'ULT soit similaire ou identique ne préjuge en rien des conditions contextuelles d'appels et d'enchaînements. On risque de payer cher la facilité de conception lors de l'intégration des ULT dans l'application.

Conception des ULT autour des données

On considère que certaines entités du monde réel joue un rôle central dans l'application et on les dote d'une série d'ULT permettant de gérer les données logiques qui leur sont associées[4] (actions de base : création, modification, suppression et consultation).

[+] On peut prévoir une assez bonne réutilisation des composants.

[+] Facilité de développer de cette manière dans certains environnements graphiques[5].

[-] Certains travaux d'étudiants se limitent parfois à une gestion des données placées dans les tables, en faisant en fait l'économie d'une analyse un peu sérieuse[6].

[-] En définitive, c'est l'utilisateur qui doit organiser les procédures logiques et maîtriser les enchaînements.

9.2.2 Modularité du MLT

On peut opérer une série de décompositions du MLT

Architecture logique d'application.

On distingue habituellement trois niveaux

l'interface homme/machine : la prise en compte d'un niveau séparé pour l'interface permet de faire tourner une même application sur des matériels hétérogènes. Seule la couche supérieure est spécifique à chaque type de machine. Notons que même dans un projet peu ambitieux, il est avantageux de ne pas mélanger interface et application, puisque différentes révisions du programme peuvent nécessiter de ne changer que l'interface : passage du mode texte au mode graphique, portage sur un autre environnement... Encore une fois, les environnements de type Visual nous tendent un piège en mettant trop à l'avant plan l'interface.

le noyau applicatif : il sera commun à toutes les versions développées pour différents matériels. Notons qu'à l'inverse, on peut garder l'interface et modifier le noyau applicatif lors d'une révision (utilisation d'un autre SGBD, passage d'un mode mono-utilisateur à multi-utilisateur...).

le guidage fonctionnel : il concerne la logique de déroulement de l'application et le passage d'une ULT à l'autre. Le fait qu'il soit isolé dans un module spécial permettra au développeur de réécrire son application en lui donnant une nouvelle logique tout en conservant la plupart des ULT. Dans les travaux d'étudiants, c'est généralement la partie la moins bien conçue. Il arrive même qu'elle ne fasse l'objet d'aucun réflexion. C'est l'utilisateur

[4] On notera que c'est la démarche qui est à l'origine de la programmation objet.

[5] Par exemple, en Visual BASIC, on peut générer une application de gestion de données atours d'un contrôle lié à une requête.

[6] A cet égard, des environnements comme Visual BASIC ou C++Builder, qui, par l'interface qu'ils génèrent, donnent parfois l'illusion d'une programmation soignée, peuvent constituer des pièges et inciter à la création de programmes très éloignés de ce qu'il aurait fallu développer.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

qui devra y suppléer tant bien que mal en activant les fonctions selon une logique qu'il doit créer ou reconstituer lui-même.

9.3 Modèles logique de traitements répartis

Il est de plus en plus fréquent de réaliser des tâches informatiques en utilisant plusieurs machines logiques, ayant chacune des aptitudes particulières. Il convient alors d'envisager la manière dont les tâches vont se répartir entre ces machines et d'assurer une cohérence entre elles. Notons qu'il ne s'agit plus ici de déterminer des postes de travail (qui ont été vus au niveau organisationnel), mais de définir un découpage du travail propre au système informatique (qui n'a la plupart du temps aucun équivalent dans le niveau organisationnel).

9.3.1 Démarche de répartition

On conseille généralement de bien définir les traitements à effectuer, indépendamment de l'architecture matérielle. On peut ensuite préciser l'architecture matérielle dont on dispose ou dont on souhaite disposer. Ce n'est que dans une troisième phase qu'on peut répartir les traitements.

9.3.2 Modalités de répartition

traitements coopératifs

On parlera de traitement coopératif quand une même ULT est exécutée sur plusieurs machines logiques. Cela suppose des mécanismes d'interaction assez complexes (ce sera toujours le cas dans un modèle client/serveur).

données synchronisées

Les données sont dites synchronisées quand une copie des données existe sur chaque machine logique. Pour réaliser la synchronisation en temps réel, de puissants mécanismes sont nécessaires. La plupart du temps, on travaille avec des copies de type cliché (*snapshot*) qui sont mis à jour au bout d'un temps (par exemple, tous les jours).

client/serveur

Les cas du client serveur reprend les caractéristiques du traitement coopératif, mais en spécialisant les machines logiques. L'une d'elle se chargeant seule de mémoriser les données et d'en assurer la cohérence (c'est le serveur de données). On distingue d'ordinaire trois composantes :

- présentation (présentation proprement dite et logique de dialogue)
- traitements (logique fonctionnelle et règles de calcul)
- données (accès aux données)

Ces trois composants sont diversement assumés par le client ou le serveur.

modèles à trois couches

Variante récente du modèle client/serveur, le modèle à trois couches (*three tiers*) tend à s'imposer, notamment dans le contexte d'Internet/Intranet. Les trois composantes précisées dans le cadre client/serveur, sont cette fois réparties sur trois machines logiques :

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

– une machine client faisant tourner une application légère (de type navigateur) ;
– une machine assurant à l'autre bout de la chaîne le rôle de serveur de données ;
– une machine intermédiaire, qu'on peut appeler « serveur d'applications ».

Le modèle à trois couches présente l'avantage de distinguer la couche de présentation (assurée par un programme élémentaire transmis au client sous forme de fichiers HTML et d'images) et la couche de traitements, centralisée sur le serveur de pages. Si cette démarche est la seule possible dans le cas d'Internet, elle présente l'avantage, dans le cas d'une entreprise, de ne plus devoir gérer les applications sur chaque machine/client. La machine logique intermédiaire et le serveur de données peuvent continuer à se répartir les tâches de gestion des données, de logique fonctionnelles et de règles de calculs. La logique de dialogue est assurée par une interface tournant sur le client lointain (avec éventuellement des scripts exécutés en local). Mais le modèle de cette interface est situé sur la machine intermédiaire, et est susceptible de mises à jour aisées et de paramétrage avant envoi. Les langages tournant sur le serveur (Java, PHP, ASP ou Perl) permettent la génération de pages HTML personnalisées, qui s'adaptent parfaitement aux besoins réels du client et aux caractéristiques actuelles des données qu'il veut manipuler[7].

9.3.3 Incertitudes du futur : les modèles à client riche

Le modèle à trois couches popularisé par Internet a gagné du terrain sur les autres modèles répartis. Sa généralisation entraîne deux effets pervers : le trafic sur le réseau devient de plus en plus important, ce qu'on peut compenser partiellement par la vitesse, et le serveur d'application est sollicité pour la moindre réaction du client, ce qui entraîne son étouffement. Sans remettre en cause le principe général, on enrichit le client. Cela lui permet de réaliser des tâches complexes de traitement local des informations avant de renvoyer les données modifiées ou d'autres requêtes aux serveurs. Deux techniques sont possibles.

Exécution d'un programme dans un environnement riche

Le client ne se contente plus de recevoir du code d'affichage entrelardé de quelques petites procédures en Javascript. Il utilise un interpréteur à qui l'on fournit un véritable programme. Celui s'exécute alors comme une tâche locale. C'est la solution utilisée par Java dans les applets, par les animations Flash, et par les applications XUL ou XAML.

Java, à l'origine, propose un concept intéressant : une machine virtuelle, installée sur le client, permet de faire tourner une application envoyée par le serveur. On a misé sur la sécurité : le code est rendu inviolable et les fonctionnalités autorisées sont définies par le propriétaire de la machine. Flash possède des caractéristiques semblables, mais se veut plus interactif et résolument orienté vers le multimédia, il a quasiment évincé Java de sa niche initiale[8]. Flash présente trois désavantages : il est basé sur un format propriétaire[9], il n'autorise aucune analyse de contenu (les robots des moteurs de recherche sont incapables de procéder à l'indexation des pages contenant des programmes Flash) et son utilisation exige un *plugin* sur la machine, dont la mise à jour n'est pas toujours autorisée dans un contexte d'entreprise (ce dernier point est opposable à Java également).

XUL est un environnement élaboré dans le contexte du projet Mozilla. Il permet le fonctionnement des navigateurs Mozilla et Firefox et de leurs nombreuses extensions. Il devient

[7]Ce modèle sera illustré par un travail pratique dans le cadre du cours de SGBDR.

[8]Java est aujourd'hui essentiellement réservé à des applicatins locales, des processus embarqués (téléphones, appareils électroniques) et connaît un nouvel essor sur les serveurs d'entreprise.

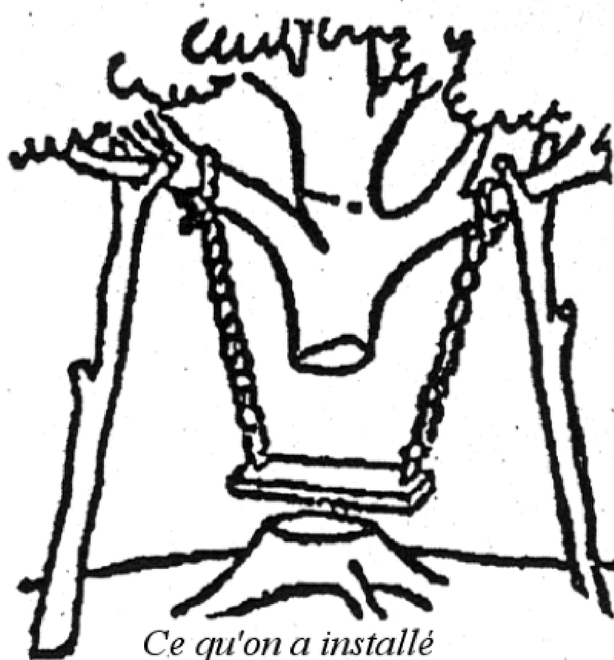
[9]Il existe mainenant des alternatives libres pour générer du code Flash, mais les outils disponibles ne sont pas aussi élaborés que ceux qu propose Adobe.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

possible de produire des modules XUL qui n'ont plus qu'un lointain rapport avec la navigation Internet. XAML est lié à Windows Vista. L'optique de Microsoft serait de réaliser des applications qui circuleraient sur Internet et rendraient obsolètes les pages statiques[10]. L'un comme l'autre sont basés sur du code rédigés en XML. XUL possède deux avantages : il est disponible depuis plus longtemps ans et peut tourner sur toutes les plates-formes. La diffusion de XAML est liée à la sortie de Windows Vista, qui a eu du mal à s'imposer dans le monde du travail. Les incessantes mises à jour des modules rendent parfois la maintenance pénible.

Ajax

Encore mal standardisé, Ajax (*Asynchronous Javascript And XML*) décrit plus une manière de procéder qu'un environnement précis. Il peut faire appel à de nombreuses technologies, tant sur le serveur que sur le client. Le principe consiste à scinder la page en plusieurs portions qui ne sont pas rechargées à chaque intervention de l'utilisateur. Des traitements locaux sont fortement encouragés et le serveur n'est sollicité que pour envoyer les parties de page qui ont besoin d'être mises à jour. Au niveau du serveur, toutes les technologies actuelles restent applicables (notamment les trois adversaires PHP, ASP et JSP). Au niveau du client, on retrouve un mélange de (x)HTML, de XML et de Javascript, mais enrichi par de nouvelles bibliothèques. Riche fonctionnellement et souvent esthétique, l'environnement Ajax exige cependant des connaissances pointues dans plusieurs domaines : le langage qui motorise le serveur (PHP, ASP ou Java), la présentation de la page (XHTML et CSS) , Javascript et le framework choisi spécifiquement. Des programmeurs possédant tous ces talents sont rares et chers.



[10]La tentative de coloniser Internet avec des sites basés sur des modules ActiveX a plus ou moins échoué parce que ceux-ci tournaient nativement sur le processeur, avec les conséquences qu'on a vues pour la sécurité.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Chapitre 10

Modèle logique des données (MLD)

Ce chapitre ne doit pas répéter ce qui ferait l'objet d'un cours de système de gestion de bases de données. On se trouve donc à la limite entre les deux disciplines. On se limitera ici à une présentation des SGBD relationnels, puisqu'ils représentent près de 90% du marché. Notons qu'une approche d'un autre système entraînerait d'autres développements. Après un rappel terminologique, le chapitre comporte une présentation des règles de normalisation (qui ont plus leur place dans une approche relationnelle que dans la méthode de Merise), suivie d'une méthodologie de la « traduction » d'un MOD en MLD. Il est à noter que cette « traduction » obéit à des règles tellement précises qu'elle peut faire l'objet d'un traitement automatique.

10.1 SGBDR

Les notions de relation (ou table), d'attribut (ou colonne) et de domaine, de tuple (ou de ligne), de clé primaire, de contrainte référentielle et de clé étrangère ainsi que les différentes opérations de l'algèbre relationnelle (restriction, projection, jointures, union, intersection et différence) devraient être connues.

10.2 Normalisation

La création d'une base de données relationnelles passe par une série d'étapes complexes. Les méthodes d'analyse fournissent une panoplie de méthodes qui permettent d'aboutir à un modèle de représentation qui peut directement se traduire sous la forme de tables. Ces méthodes seront vues dans la section suivante. En outre, des logiciels permettent aujourd'hui de réaliser une partie de cette tâche de manière automatique. Nous allons aborder une autre vision de ce travail en introduisant la notion de normalisation. On ne normalise pas pour le plaisir de normaliser. La présentation des données en formes normales est un idéal à atteindre. Plusieurs techniques sont envisageables :

– on procède à une analyse sommaire du problème à traiter. On établit un dictionnaire des données. On réalise une table regroupant toutes ces données et on tente de normaliser cette table. C'est l'optique que nous allons adopter. Dans une application réelle, cette méthode risque d'entraîner un fiasco, parce que l'analyse sommaire va probablement passer à côté d'une foule de problèmes qui resurgiront plus tard. Pour échapper à cet écueil, nous allons volontairement étudier un cas très simple, en nous interdisant d'envisager une extension de notre programme à d'autres tâches, qui nous obligeraient à mener

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

FIRME Jean Deuxfontaines
Avenue des Grands Pins 176
4000 Liège

Client n° 1045
Monsieur Jean PREVERT
rue de l'Eglise 67
4020 Liège

Facture N° *Date*
2001/148 *30/3/2001*

Code	Qté	Désignation	PrixU	PrixTotal
2001	1	Scie sauteuse KB 4566	1321,48	1321,48
2501	2	Jeu de 5 lames pour scie	123,96	247,92

Prix Total 1569,40
329,60
1899,00

FIG. 10.1 – **Exemple de facture**

une analyse plus fouillée. Dans une situation réelle, cette méthode est celle qu'il faudra employer quand on dispose déjà d'une application boiteuse dont on veut récupérer rapidement les données.
– on étudie sérieusement le problème à traiter au moyen d'une méthodologie éprouvée ou d'un logiciel spécialisé. A la fin de l'étude, on dispose d'une série de tables regroupant les données et on constate que ces tables obéissent effectivement aux règles de la normalisation. Cette méthode présente l'inconvénient de nous donner une solution immédiate qui occulte tout le travail de la normalisation et présente un résultat final tellement clair et évident qu'on se demande comment on aurait pu procéder autrement.

Nous allons étudier le problème d'une facturation. Pour éviter de nous perdre dans une étude théorique trop complexe, nous allons restreindre nos ambitions à l'impression d'une facture illustrée dans la figure 10.1.

Nous avons volontairement éliminé une série de problèmes potentiels :

– nous ne gérons que les renseignements indispensables ;
– nous n'envisageons pas de faire un suivi des factures ;
– le nombre d'éléments en stock ne sera pas pris en compte ;
– nous ne gérons pas les fournisseurs.

Le principal problème posé par cette facture consiste en la présence d'une donnée répétitive : les différents articles achetés par le client. Si on veut représenter les données de la facture sous la forme d'une relation, on va devoir d'abord identifier les différents attributs.

Nous aurons respectivement :

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

le numéro du client	**le code de l'article**
le prénom du client	l'intitulé de l'article
le nom du client	le prix unitaire
l'adresse du client	*le prix total*
le numéro de facture	*le montant de la TVA*
la date de facture	*le total général hors TVA*
	le total général TVA comprise

Remarquons que certaines informations (numéro du client, de la facture et le code de l'article) n'ont pas une existence préalable à l'informatisation, ce sont en fait déjà des clés primaires implicites. Il est certain que de telles informations ne figuraient pas dans les comptabilités pré-informatiques.

Une première approximation d'un système de facturation apparaît dans le tableau suivant. On remarque que le problème des champs répétitifs a été résolu en consacrant autant de lignes à chaque facture qu'il y a d'articles achetés.

ID_FACTURE	DATEFACTURE	MONTANT	CLIENT	ADRESSE	DESIGNATION	PRIXUNITAIRE	QUANTITE
1	02/06/08	14,1	Jacques Thoorens	rue de l'église 5 4020 Liège	Marteau	8.7	1
					Clé 10	2.7	2
2	02/06/08	14,3	Anne-Lise Comhaire	rue de la mosquée 7 4000 Liège	Marteau	8.7	1
					Clé 10	2.9	1
					Clé 12	2.9	1
3	03/06/08	11,6	Hubert Schyns	rue de la synagogue 8 4020 Liège	Marteau	8.7	1
					Clé 12	2.9	1
4	04/06/08	5,8	Jacques Thoorens	rue de l'église 5 4020 Liège	Clé 12	2.9	2

Si nous complétons les tupes incomplets à l'aide des données de la ligne précédente, nous obtenons une vraie relation :

ID_FACTURE	DATEFACTURE	MONTANT	CLIENT	ADRESSE	DESIGNATION	PRIXUNITAIRE	QUANTITE
1	02-JUN-08	14.1	Jacques Thoorens	rue de l'église 5 4020 Liège	Marteau	8.7	1
1	02-JUN-08	14.1	Jacques Thoorens	rue de l'église 5 4020 Liège	Clé 10	2.7	2
2	02-JUN-08	14.3	Anne-Lise Comhaire	rue de la mosquée 7 4000 Liège	Marteau	8.7	1
2	02-JUN-08	14.3	Anne-Lise Comhaire	rue de la mosquée 7 4000 Liège	Clé 10	2.7	1
2	02-JUN-08	14.3	Anne-Lise Comhaire	rue de la mosquée 7 4000 Liège	Clé 12	2.9	1
3	03-JUN-08	11.6	Hubert Schyns	rue de la synagogue 8 4020 Liège	Marteau	8.7	1
3	03-JUN-08	11.6	Hubert Schyns	rue de la synagogue 8 4020 Liège	Clé 12	2.9	1
4	04-JUN-08	5.8	Jacques Thoorens	rue de l'église 5 4020 Liège	Clé 12	2.9	2

10.2.1 La première forme normale

Une relation est normalisée si
- aucun attribut n'est représenté plusieurs fois et
- aucun attribut n'est décomposable en plusieurs (n'est lui même une relation).

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Pour parvenir à cette première forme normale, nous allons devoir nous occuper du second critère (le premier a été corrigé dans la sous-section précédente). Nous allons éclater les données du client en un champ nom et un champ prénom. Notons que cette séparation peut se faire facilement grâce à notre connaissance de l'onomastique belge. Ce serait plus compliqué si nous avions affaire à des noms étrangers. Ce qui est certain, c'est qu'aucun ordinateur ne sera capable de réaliser cette opération spontanément. Nous procédons de même pour l'adresse.

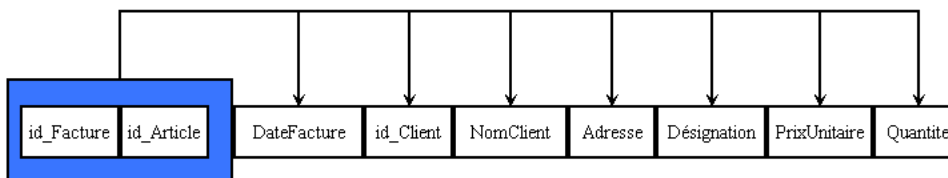
ID_FACTURE	DATEFACTURE	PRENOM	NOMCLIENT	ADRESSE	NUMERO	CODEPOSTAL	LOCALITE	DESIGNATION	PRIXUNITAIRE	QUANTITE
1	02-JUN-08	Jacques	Thoorens	rue de l'église	5	4020	Liège	Marteau	8.7	1
1	02-JUN-08	Jacques	Thoorens	rue de l'église	5	4020	Liège	Clé 10	2.7	2
2	02-JUN-08	Anne-Lise	Comhaire	rue de la mosquée	7	4000	Liège	Marteau	8.7	1
2	02-JUN-08	Anne-Lise	Comhaire	rue de la mosquée	7	4000	Liège	Clé 10	2.7	1
2	02-JUN-08	Anne-Lise	Comhaire	rue de la mosquée	7	4000	Liège	Clé 12	2.9	1
3	03-JUN-08	Hubert	Schyns	rue de la synagogue	8	4020	Liège	Marteau	8.7	1
3	03-JUN-08	Hubert	Schyns	rue de la synagogue	8	4020	Liège	Clé 12	2.9	1
4	04-JUN-08	Jacques	Thoorens	rue de l'église	5	4020	Liège	Clé 12	2.9	2

Il reste à trouver la clé primaire. Après un examen attentif, nous constatons que le couple formé par le numéro de la facture et la désignation de l'article est le seul à être unique (c'est vrai aussi actuellement du couple *Id_Facture - PrixUnitaire*, mais on conçoit bien que deux articles pourraient avoir le même prix).

Notre réalité est profondément influencée par l'informatique. Nous savons que les entreprises utilisent des codes pour désigner leurs articles et il n'est pas rare d'avoir des codes clients. Nous verrons que cela nous sera bien utile par la suite. Nous allons donc introduire ces deux notions (et au passage masquer les champs *Prenom*, *Numero*, *CodePostal* et *Localite* qui n'apportent rien à la démonstration).

ID_FACTURE	DATEFACTURE	ID_CL	NOMCLIENT	ADRESSE	ID_ARTICLE	DESIGNATION	PRIXUNITAIRE	QUANTITE
1	02-JUN-08	TJ	Thoorens	rue de l'église	1	Marteau	8.7	1
1	02-JUN-08	TJ	Thoorens	rue de l'église	2	Clé 10	2.7	2
2	02-JUN-08	CA	Comhaire	rue de la mosquée	1	Marteau	8.7	1
2	02-JUN-08	CA	Comhaire	rue de la mosquée	2	Clé 10	2.7	1
2	02-JUN-08	CA	Comhaire	rue de la mosquée	3	Clé 12	2.9	1
3	03-JUN-08	SH	Schyns	rue de la synagogue	1	Marteau	8.7	1
3	03-JUN-08	SH	Schyns	rue de la synagogue	3	Clé 12	2.9	1
4	04-JUN-08	TJ	Thoorens	rue de l'église	3	Clé 12	2.9	2

La clé primaire *Id_Facture - Id_Article* est encore plus sûre que celle que nous avions auparavant.



Le résultat obtenu est insatisfaisant à bien des égards :

– une même donnée est répétée plusieurs fois (par exemple, l'adresse de Schyns ou le prix du marteau) ;

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

– si on complète la table, il est parfaitement possible d'encoder une nouvelle facture pour Comhaire en lui attribuant une adresse différente ;
– si on supprime la facture d'un nouveau client, on risque de perdre les coordonnées de ce client.

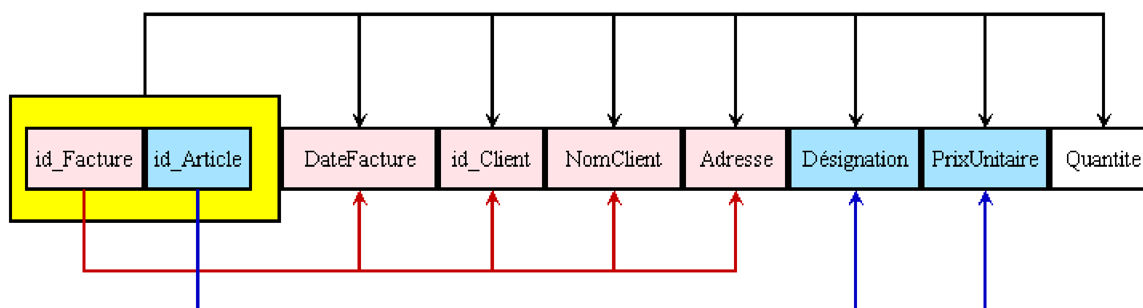
10.2.2 Deuxième forme normale

Une relation est en deuxième forme normale si
– elle est en première forme normale et
– toutes les dépendances issues de la clé sont élémentaires (c'est-à-dire qu'elles ont pour source la clé entière et non un de ses attributs).

On parle de **dépendance fonctionnelle** entre deux attributs *A* et *B* d'une relation lorsque pour tous les couples (A_i, B_j), obtenus par une projection de la relation sur les deux attributs *A* et *B*, on peut faire correspondre nécessairement à un même élément A_i un même élément B_j. L'attribut du premier élément s'appelle la source de la fonction, l'attribut du second le but.

Dans la relation en première forme normale, chaque fois que nous avons l'identifiant d'article 1, la valeur de *Désignation* est nécessairement 'Marteau'.

Examinons la relation. La clé complexe formée de *id_Facture* et *idArticle* spécifie une seule valeur de chacun des autres attributs. Mais on trouve d'autres dépendances fonctionnelles :
– entre le numéro de facture et soit la date, soit le numéro du client, soit son nom, soit son adresse
– entre le numéro de l'article et soit sa désignation, soit son prix.

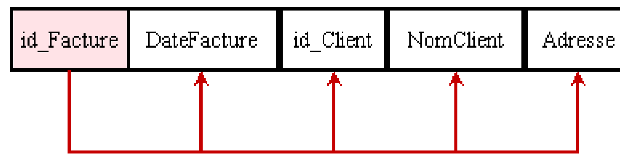


Nous allons devoir les éliminer.

Pour éliminer les dépendances fonctionnelles découvertes dans notre exemple, nous allons scinder notre relation initiale en deux autres relations. Nous allons éliminer les dépendances fonctionnelles entre l'attribut *id_Facture* et chacun des attributs de la relation en les regroupant dans une autre relation. La première relation obtenue possède une clé primaire simple. Cette particularité fait qu'elle est automatiquement en deuxième forme normale.

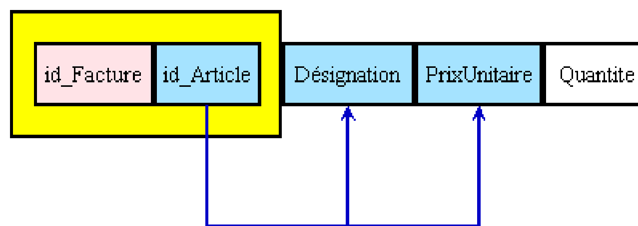
ID_FACTURE	ID_CL	DATEFACTURE	NOMCLIENT	PRENOM	ADRESSE
1	TJ	02-JUN-08	Thoorens	Jacques	rue de l'église
2	CA	02-JUN-08	Comhaire	Anne-Lise	rue de la mosquée
3	SH	03-JUN-08	Schyns	Hubert	rue de la synagogue
4	TJ	04-JUN-08	Thoorens	Jacques	rue de l'église

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET



La seconde relation ne conservera que les attributs qui ne dépendaient pas de *id_Facture*.

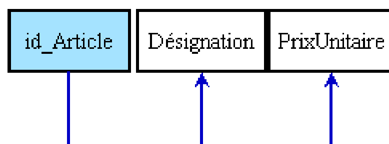
ID_FACTURE	ID_ARTICLE	DESIGNATION	PRIXUNITAIRE	QUANTITE
4	3	Clé 12	2.9	2
1	1	Marteau	8.7	1
1	2	Clé 10	2.7	2
2	1	Marteau	8.7	1
2	2	Clé 10	2.7	1
2	3	Clé 12	2.9	1
3	1	Marteau	8.7	1
3	3	Clé 12	2.9	1

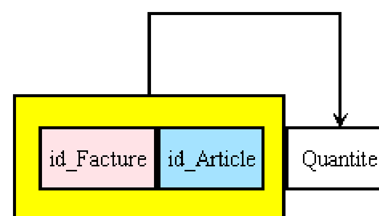


Ce qu'on pourrait appeler la table des commandes présente encore des dépendances fonctionnelles entre une partie de la clé (*id_Article*) et des attributs qui ne sont pas membres de la clé. Il devient nécessaire de la scinder à nouveau en deux nouvelles relations.

ID_ARTICLE	DESIGNATION	PRIXUNITAIRE
1	Marteau	8.7
2	Clé 10	2.7
3	Clé 12	2.9
4	Clé 14	3.1

ID_FACTURE	ID_ARTICLE	QUANTITE
4	3	2
1	1	1
1	2	2
2	1	1
2	2	1
2	3	1
3	1	1
3	3	1





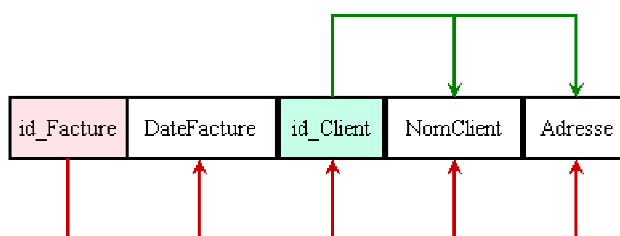
©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

10.2.3 Troisième forme normale

Une relation est en troisième forme normale si :
- elle est en deuxième forme normale
- toutes les dépendances fonctionnelles issues de la clé sont directes

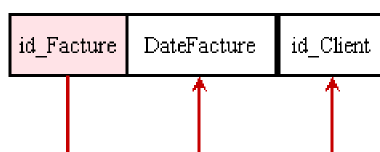
ID_FACTURE	ID_CL	DATEFACTURE	NOMCLIENT	PRENOM	ADRESSE
1	TJ	02-JUN-08	Thoorens	Jacques	rue de l'église
2	CA	02-JUN-08	Comhaire	Anne-Lise	rue de la mosquée
3	SH	03-JUN-08	Schyns	Hubert	rue de la synagogue
4	TJ	04-JUN-08	Thoorens	Jacques	rue de l'église

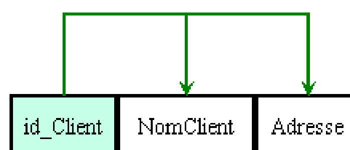
Toutes les dépendances n'ont pas encore été épuisées par notre normalisation. En effet, dans la relation des factures, on peut dire que le nom du client (ou son adresse) dépend de la clé primaire. Mais cette dépendance n'est pas directe. Le nom du client dépend du numéro qui est lui-même dépendant de la clé primaire. On parle alors de dépendance transitive. Cette dépendance entraîne à nouveau des risques d'incohérence. La solution consiste de nouveau à scinder la table.



ID_FACTURE	DATEFACTURE	ID_CL
1	02-JUN-08	TJ
2	02-JUN-08	CA
3	03-JUN-08	SH
4	04-JUN-08	TJ

ID_CL	NOMCLIENT	ADRESSE
TJ	Thoorens	rue de l'église
BA	Bawin	rue du temple
CA	Comhaire	rue de la mosquée
SH	Schyns	rue de la synagogue





©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

10.2.4 Reconstruction des données originales

ID_ARTICLE	DESIGNATION	PRIXUNITAIRE
1	Marteau	8.7
2	Clé 10	2.7
3	Clé 12	2.9
4	Clé 14	3.1

ID_CL	NOMCLIENT	ADRESSE
TJ	Thoorens	rue de l'église
BA	Bawin	rue du temple
CA	Comhaire	rue de la mosquée
SH	Schyns	rue de la synagogue

ID_FACTURE	ID_ARTICLE	QUANTITE
4	3	2
1	1	1
1	2	2
2	1	1
2	2	1
2	3	1
3	1	1
3	3	1

ID_FACTURE	DATEFACTURE	ID_CL
1	02-JUN-08	TJ
2	02-JUN-08	CA
3	03-JUN-08	SH
4	04-JUN-08	TJ

A quoi peut bien servir de disposer d'une base de données, si nous voyons nos données se disperser dans des tables séparées ? Il existe heureusement des opérations d'algèbre relationnelle qui permettent de recréer les relations originales, dont l'existence devient virtuelle, mais qu'on peut parfaitement exploiter pour afficher les données. Lorsque les applications doivent manipuler fréquemment certaines de ces relations, il est possible d'en enregistrer les commandes dans une *vue*, qu'on peut employer sans connaître les commandes complexes permettant les jointures de tables.

```
SELECT *
FROM Factures
INNER JOIN Clients USING (id_Client)
INNER JOIN Commandes USING (id_Facture)
INNER JOIN Articles USING (id_Article)
```

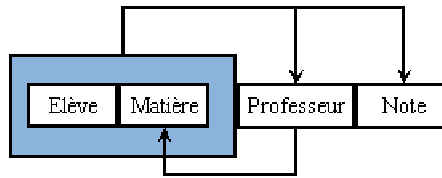
ID_FACTURE	DATEFACTURE	ID_CL	NOMCLIENT	ADRESSE	ID_ARTICLE	DESIGNATION	PRIXUNITAIRE	QUANTITE
1	02-JUN-08	TJ	Thoorens	rue de l'église	1	Marteau	8.7	1
1	02-JUN-08	TJ	Thoorens	rue de l'église	2	Clé 10	2.7	2
2	02-JUN-08	CA	Comhaire	rue de la mosquée	1	Marteau	8.7	1
2	02-JUN-08	CA	Comhaire	rue de la mosquée	2	Clé 10	2.7	1
2	02-JUN-08	CA	Comhaire	rue de la mosquée	3	Clé 12	2.9	1
3	03-JUN-08	SH	Schyns	rue de la synagogue	1	Marteau	8.7	1
3	03-JUN-08	SH	Schyns	rue de la synagogue	3	Clé 12	2.9	1
4	04-JUN-08	TJ	Thoorens	rue de l'église	3	Clé 12	2.9	2

10.2.5 Formes normales plus avancées

Troisième forme de Boyce et Codd

Il s'agit d'une forme plus restrictive de la troisième forme normale. La plupart des tables en troisième forme normale ont cette propriété, mais il existe quelques exceptions qui doivent être décelées. Ce sont les cas où une des colonnes de la clé est en dépendance fonctionnelle d'une autre colonne. L'exemple suivant illustre le problème.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET



Dans un Lycée français :
– à chaque couple Élève, Matière correspond un seul professeur
– à chaque couple Élève, Matière correspond une seule note finale
– chaque professeur enseigne une seule matière.

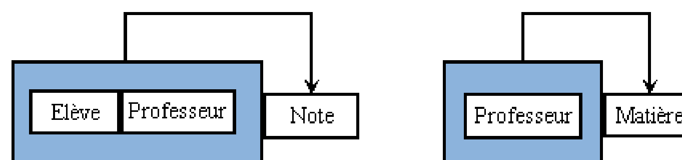
Dans une telle table, rien ne nous empêcherait d'avoir les tuples suivants.

ELEVE	MATIERE	PROFESSEUR	NOTE
Pierre	Anglais	Jospin	18
Paul	Math	Chirac	16
André	Allemand	Jospin	14
Robert	Math	Chirac	10

Selon ce qui a été dit, le professeur Jospin ne peut pas donner à la fois anglais et allemand[1]. Pour résoudre la difficulté, on scindera la table en deux autres tables.

ELEVE	PROFESSEUR	NOTE
Pierre	Jospin	18
Paul	Chirac	16
André	Jospin	14
Robert	Chirac	10

PROFESSEUR	MATIERE
Jospin	Anglais
Chirac	Math



Tables NF3BC - Les tables en forme normale de Boyce et Codd

> Une relation est en forme normale de Boyce et Codd si la source de chaque dépendance formelle existant entre les attributs de la relation fait partie de la clé de la relation.

Notons que si la clé est simple, l'exigence de Boyce et Codd est implicitement respectée.

[1]Remarquons, pour éviter tout malentendu, que l'exemple perd toute pertinence si on l'applique à nos écoles où les professeurs sont polyvalents. Je doute d'ailleurs que la dépendance relevée existe ailleurs que dans quelques lycées parisiens particulièrement favorisés.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

La quatrième forme normale

Une table est en quatrième forme normale si :
- elle est en forme normale de Boyce et Codd
- elle ne possède pas de dépendance multiévaluée.

Il existe un autre type de dépendance, qui fait intervenir au moins trois attributs . On parle alors de dépendances multiévaluées. Prenons l'exemple d'une université dans laquelle des professeurs donnent des cours et sont susceptibles d'utiliser des manuels. La table suivante exprime la situation :

COURS	PROFESSEUR	MANUEL
Mathématique	Pr Green	Analyse vectorielle
Mathématique	Pr Green	Trigonométrie
Mathématique	Pr Green	Principes d'Algèbre
Physique	Pr Brown	Mécanique de base
Physique	Pr Brown	Principes d'optique
Physique	Pr Green	Mécanique de base
Physique	Pr Green	Principes d'optique

Supposons maintenant qu'un nouveau professeur de physique soit engagé, il faudrait obligatoirement pouvoir imposer l'ajout de deux lignes, puisqu'il est susceptible d'utiliser les deux ouvrages de physique.

Dans cette table, en troisième forme normale de Boyce et Codd, nous n'avons aucun moyen de l'imposer. Par contre en faisant éclater cette table en deux autres, on aura un solution élégante à notre problème. Ces deux tables seront dites en 4me forme normale.

COURS	PROFESSEUR
Mathématique	Pr Green
Physique	Pr Brown
Physique	Pr Green

COURS	MANUEL
Mathématique	Analyse vectorielle
Mathématique	Principes d'Algèbre
Mathématique	Trigonométrie
Physique	Mécanique de base
Physique	Principes d'optique

Tables NF4ab - Deux tables en quatrième forme normale

La cinquième forme normale

Une table est en cinquième forme normale quand elle ne contient pas de dépendances de jointure.

La cinquième forme normale fait intervenir la notion de dépendance de jointure. On peut intuitivement comprendre cette notion en tentant d'établir une décomposition-jointure, qui décompose une relation par projection, la recompose par jointure, ce qui devrait avoir pour effet de reconstituer la relation initiale. Voici un exemple qui marche bien :

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

MALADIE	GERME	ANTIBIOTIQUE
M1	G1	A2
M2	G2	A2
M2	G2	A4
M3	G1	A2
M3	G1	A3
M3	G2	A2

Table candidate à la 5me forme normale

En décomposant par projection la table ci-dessus en trois relations, on obtient les trois tables suivantes :

MALADIE	GERME
M1	G1
M2	G2
M3	G2
M3	G1

GERME	ANTIBIOTIQUE
G1	A2
G2	A2
G2	A4
G1	A3

MALADIE	ANTIBIOTIQUE
M1	A2
M2	A2
M3	A2
M2	A4
M3	A3

En combinant les deux premières par jointure naturelle, on en obtient une autre. En effectuant une jointure du résultat avec la troisième table, on élimine les lignes qui ont été mises en évidence et on retrouve la table initiale.

La table initiale peut donc sans difficulté être réduite aux trois tables simples. On dit qu'elle comporte des dépendances de jointures. Elle n'est donc pas en forme normale. Par contre, si on supprime la dernière ligne de cette relation, la même décomposition est produite (à cause de l'absence de doublons). Cette fois, la recomposition va générer une table qui comprendra une ligne de plus, celle qui a été supprimée. Il ne faut donc pas procéder à la scission puisque celle-ci entraîne une perte d'information. La table ainsi modifiée ne comporte pas de dépendance de jointure et se trouve donc en cinquième forme normale.

MALADIE	GERME	ANTIBIOTIQUE
M1	G1	A2
M2	G2	A2
M2	G2	A4
M3	G1	A3
M3	G2	A2

Table en 5me forme normale

L'étude des quatrième et cinquième formes normales présente plus de difficultés que celle des trois premières. Heureusement, les cas de dépendances multiévaluées et surtout de dépendances de jointure sont relativement rares.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

10.3 Transformation du MOD en MLD

Une série de règles autorisent une véritable traduction entre les deux modèles (MOD et MLD). Notons que l'opération inverse n'est pas possible et qu'à un même modèle logique peuvent correspondre plusieurs modèles organisationnels possibles.

10.3.1 Entité

Entité → Table
Propriété → Attribut
Identifiant → Clé primaire

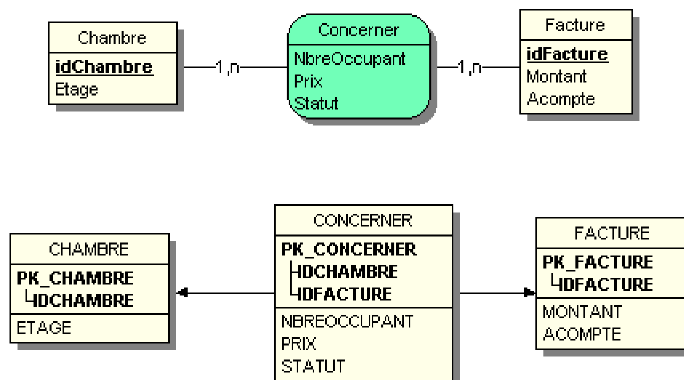
Si l'entité possède comme seule propriété son identifiant, la table ne comportera qu'un seul attribut. Il est cependant probable que cette table disparaîtra lors de la phase d'optimisation.

10.3.2 Relations binaires

a) Relations binaires (0,n)-(0,n) ou (1,n)-(1,n) ou (0,n)-(1,n)

Il s'agit du cas le plus complet. On remarquera que la cardinalité maximale est n sur les deux pattes. Chacune des occurrences d'entité peut intervenir dans plusieurs associations. Seule l'association de deux entités précises est unique. On va donc créer une table intermédiaire reprenant comme attributs les propriétés de la relation et comme clé primaire les deux identifiants des tables intervenant dans la relation. Les composants de cette clé sont évidemment des clés étrangères des tables d'entités.

Dans notre exemple, une chambre d'hôtel peut faire l'objet de plusieurs occupations et une facture peut concerner plusieurs chambres. Par contre, une même chambre ne figurera qu'une seule fois dans une même facture.



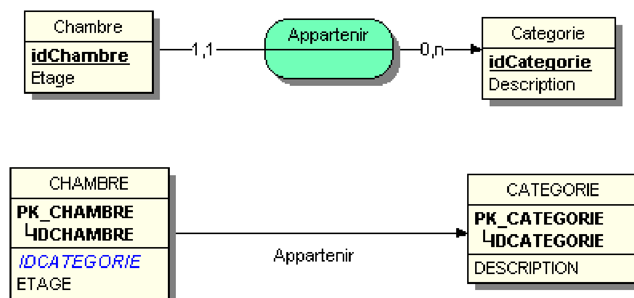
b) Relations binaires (0,n)-(1,1) ou (1,n)-(1,1)

Vu la cardinalité maximale 1 placée sur une des entités, il n'est pas nécessaire de créer une table intermédiaire. On place une copie de la clé primaire de la deuxième entité dans la table ayant une cardinalité (1, 1). Les éventuelles propriétés de la relation seraient également placées dans cette table[2]. Dans l'exemple suivant, la clé primaire de la Catégorie devient une

[2]WinDesign n'accepte pas de telles propriétés dans l'association. C'est, me semble-t-il, une restriction regrettable.

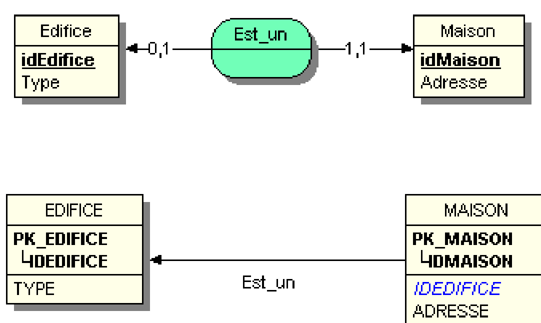
©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

propriété de la Chambre. En termes de SGBD, cette clé primaire constitue une clé étrangère (*foreign key*).



c) Relations binaires (0,1)-(1,1)

C'est un cas particulier de ce qui a été vu plus haut. Cela revient à exprimer un sous-type. Comme précédemment, l'identifiant de l'autre entité est placé comme clé étrangère dans l'entité à cardinalité (1,1).



d) Relations binaires (0,n)-(0,1) ou (1,n)-(0,1)

On pourrait traiter ce cas comme le précédent, mais se pose le cas de la cardinalité minimale 0, qui impose d'utiliser une clé étrangère de valeur nulle, ce que tous les SGBD ne tolèrent pas toujours facilement. On peut alors proposer une solution dans laquelle la relation devient une table ayant l'identifiant de l'entité à cardinalité (0,1) comme clé primaire et l'autre identifiant comme simple attribut (en fait clé étrangère) .

Dans l'exemple dans la figure 10.2, une voiture est possédée par au plus une personne. La première solution propose de placer une clé étrangère *Nom* renvoyant au propriétaire dans la table *Voiture*. S'il n'y en a pas, on placera la valeur Null dans le champ *Nom* de la table *Voiture*. La cardinalité maximale de 1 est assurée par l'existence d'un seul champ permettant de désigner le propriétaire. Dans la seconde solution, une table *Posséder* reprend les propriétés de la relation, plus les deux identifiants des entités. Lorsque la voiture n'a pas de propriétaire, aucune occurrence de *Posséder* n'a le numéro de la voiture comme clé primaire. La cardinalité maximale de 1 est assurée par le fait que le numéro de la voiture est une clé primaire de la relation *Posséder*.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

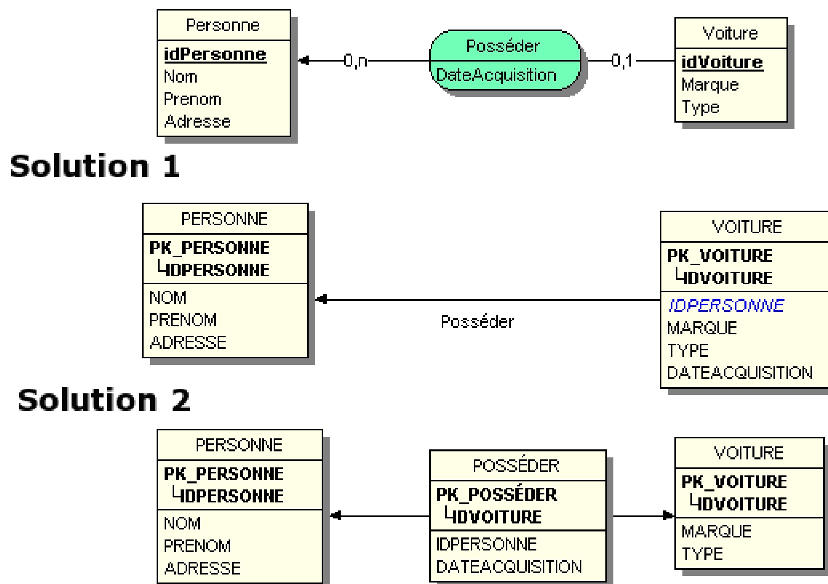


FIG. 10.2 – Relations binaires (0,n)-(0,1) ou (1,n)-(0,1)

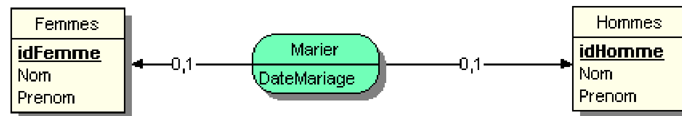
e) Relations binaires (0,1)-(0,1)

On retrouve en fait les deux possibilités du cas précédent, mais avec chaque fois deux variantes puisque la cardinalité maximale est 1 des deux côtés de la relation. Deux des solutions utilisent une clé externe qui peut prendre une valeur nulle, avec les inconvénients que cela peut amener (voir figure 10.3).

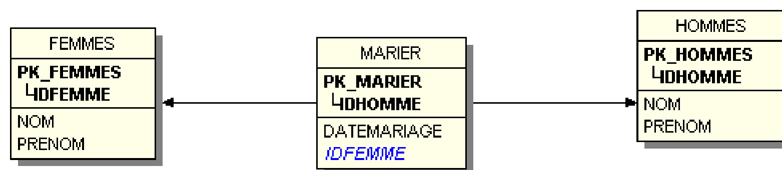
10.3.3 Relations ternaires ou supérieures

Le traitement des relations ternaires ou supérieures nécessite la création d'une table reprenant les propriétés de la relation et une clé primaire composée des identifiants de chacune des entités intervenant dans la relation.

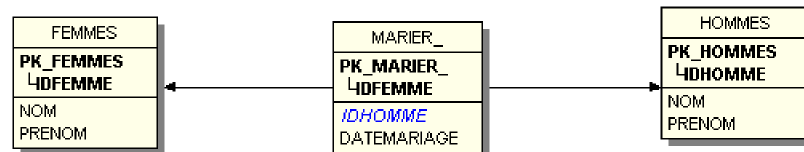
©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET



Solution 1



Solution 2



Solution 3

Solution 4

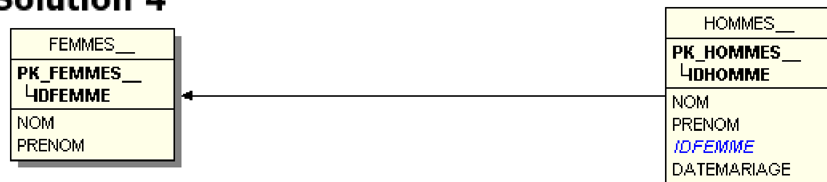
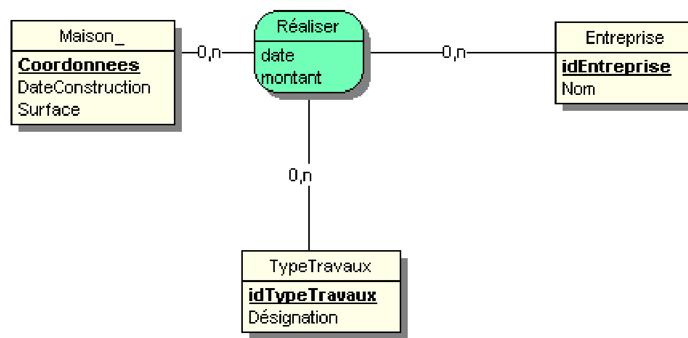
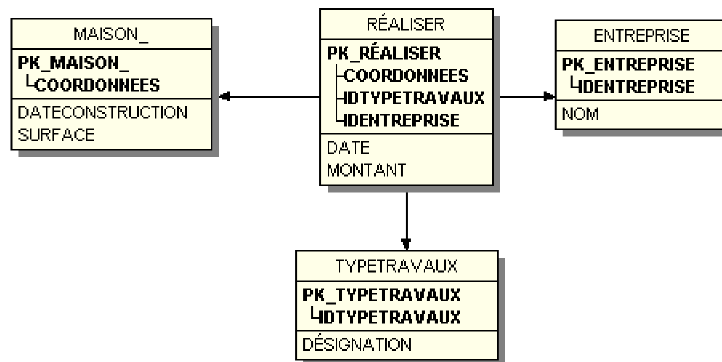


FIG. 10.3 – Relations binaires (0,1)-(0,1)

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET



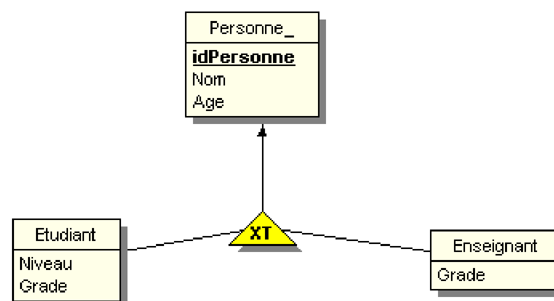


10.3.4 Relations réflexives

On entend par relation réflexives des relations qui impliquent plusieurs fois une même entité. Il n'y a pas de précautions particulières avec celles-ci (on distingue les mêmes cas que dans d'autres relations), avec cependant une difficulté particulière résultant de la présence de clés primaires forcément homonymes. On règle les conflits en renommant les attributs homonymes.

10.3.5 Sous-types

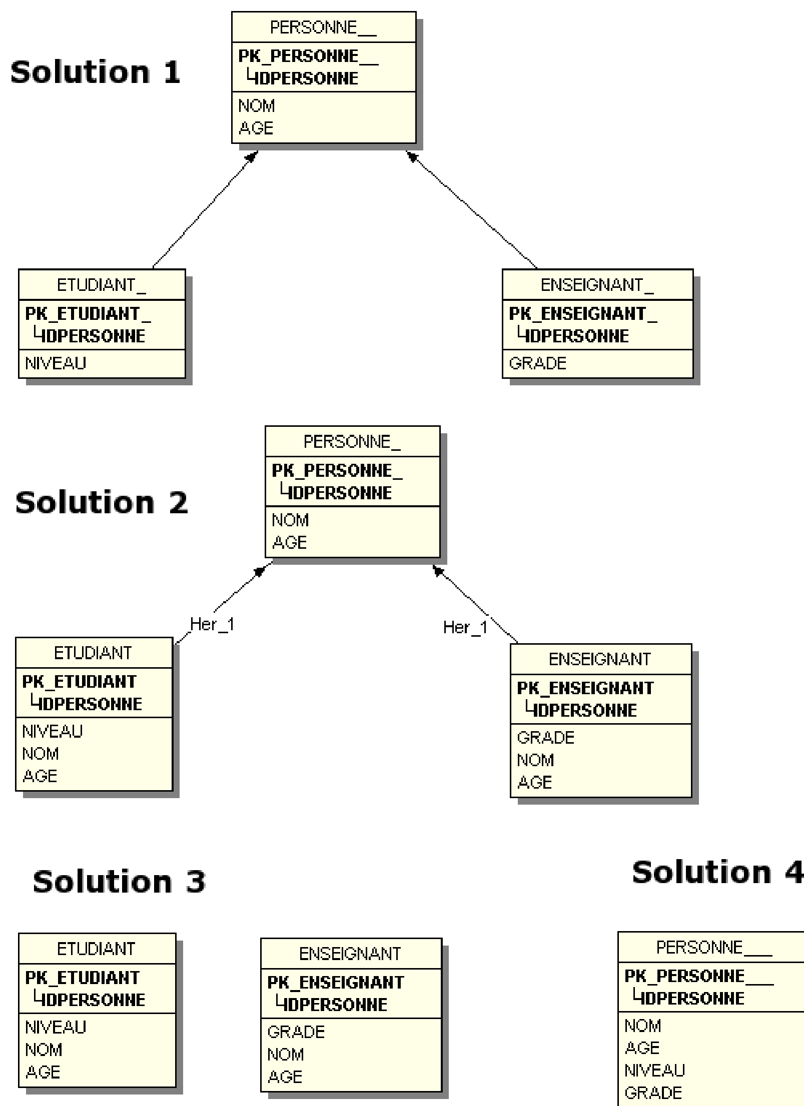
a) Spécialisation



La méthode la plus simple consiste à créer une table générale reprenant les propriétés communes et l'identifiant comme clé primaire et autant de tables qu'il y a des sous-types.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Chacune d'entre elles contient une clé primaire reprenant la valeur de la clé du sur-type et les seuls champs spécifiques au sous-type.



Trois autres méthodes existent également :

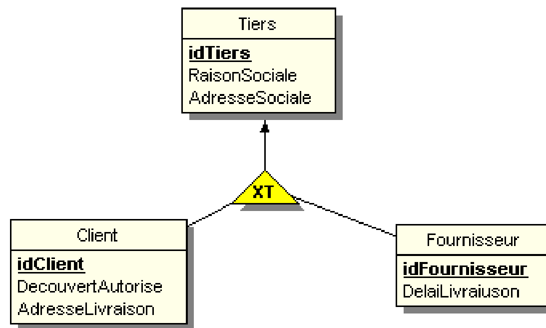
- une deuxième représentation qui reprend ce qui vient d'être dit, mais avec une recopie des propriétés communes dans chaque sous-type. Cela sous-entend une mise en œuvre automatique de cette recopie.
- une troisième représentation où chaque sous-type devient une table à part entière. Cette solution est déconseillée si les différents sous-types interviennent chacun dans des relations différentes.
- une dernière représentation reprend dans une seule table toutes les propriétés de tous les sous-types (avec forcément des valeurs nulles quand la propriété ne correspond à rien pour une occurrence particulière). Les sous-types sont créés à l'aide de vues qui ne montrent que les propriétés pertinentes.

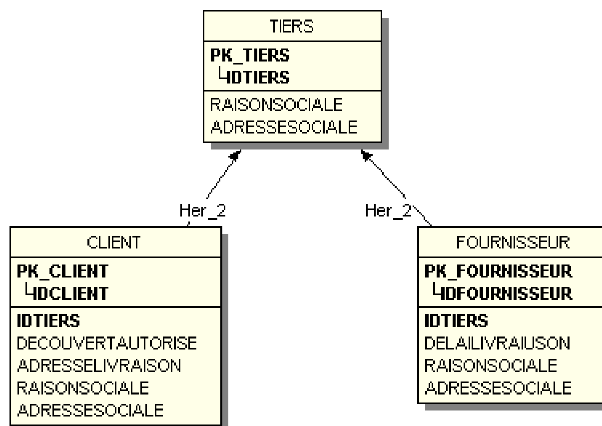
b) généralisation

On trouve ici des sous-types comme des tables indépendantes avec leurs clés primaires différentes. L'appartenance à une classe générique (implémentée dans une autre table) se fait

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

au moyen d'une clé externe que chaque sous-type contient. On trouvera ici encore la possibilité de recopier les champs propres au sur-type dans les sous-types (possibilité non illustrée dans le schéma).

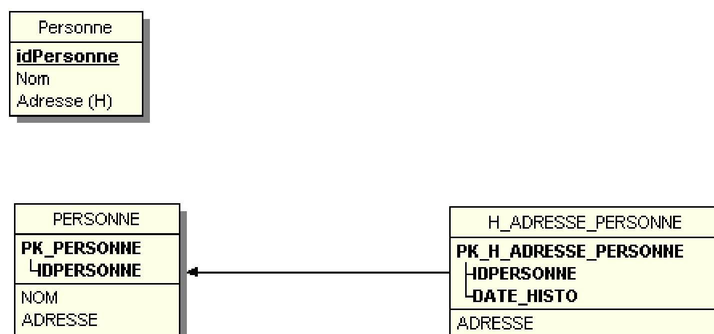




10.3.6 Historisation

a) historisation de propriétés

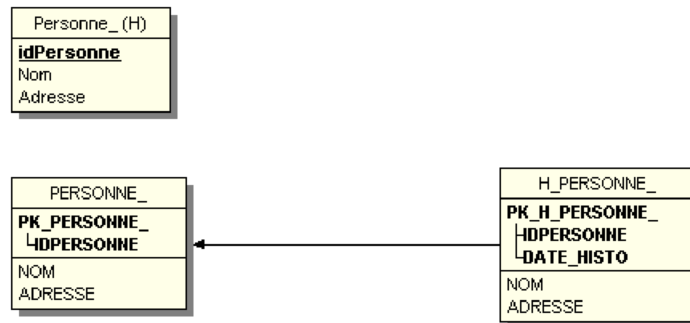
On crée une table contenant les anciennes valeurs de la propriétés comme unique attribut original. La clé primaire de cette table est double et comporte un attribut reprenant les valeurs des clés primaires de l'occurrence d'entité dont la propriété a changé et la date de l'historisation.



©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

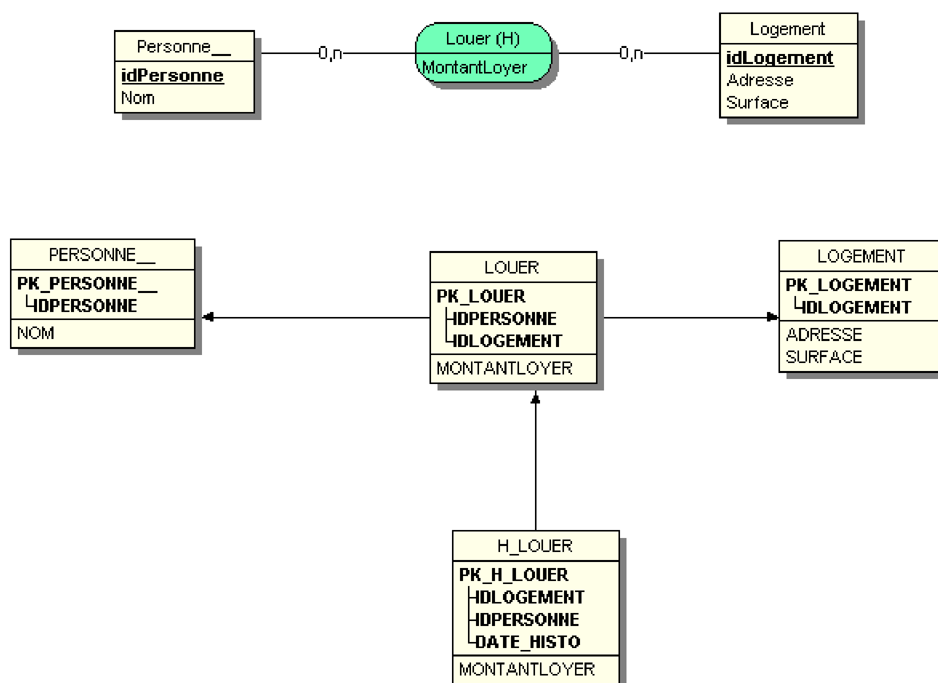
b) historisation d'entités

On crée une table qui est la copie parfaite de l'entité historisée mais dont la clé primaire est enrichie d'un attribut comprenant la date d'historisation.



c) historisation de relations

Ici encore la table exprimant la relation est dupliquée et sa clé primaire est enrichie de la date d'historisation.



10.3.7 Prise en compte des contraintes d'intégrité

La manière de rendre compte de multiples contraintes dépend fortement du SGBDR employé et de ses possibilités. Rappelons qu'on peut situer cette vérification à trois niveaux :

- au niveau d'une application (solution peu avantageuse car facile à contourner)
- au niveau d'un serveur d'applications (modèle à trois couches), ce qui permet de modifier certaines contraintes sans remettre en cause la structure globale des données (on parle de contraintes d'entreprise)

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

– au niveau de la base de données, la solution la plus élégante et surtout la plus fiable.

Certaines contraintes peuvent avoir un aspect *ad hoc* (elles ont été notées en marge des modèles), d'autres au contraire sont inscrites dans un formalisme particulier. Cela ne signifie pas qu'il soit toujours possible de les inclure dans le modèle logique. On aura recours à différents types de moyens mis à disposition par le SGBD[3] :

- choix d'un type standard ou création d'un type utilisateur (possédant éventuellement des contraintes spécifiques) ;
- utilisation de l'expression `CONSTRAINT`, avec toutes ses variantes ;
- écriture de `TRIGGER`s pour régler au coup par coup des contraintes plus complexes ;
- enfin, sur les systèmes qui en disposent, utilisation des `ASSERTIONS`, sortes de contraintes non liées à une table particulière[4].

<table>
<tr><th></th><th>1,n</th><th>0,n</th><th>1,1</th><th>0,1</th></tr>
<tr><td>1,n
0,n</td><td colspan="2">A - table relation avec clé double</td><td>B - lien avec clé étrangère</td><td>D - table relation avec clé simple ou lien avec clé étrangère</td></tr>
<tr><td>1,1</td><td colspan="2">B - lien avec clé étrangère</td><td></td><td>C - lien avec clé étrangère</td></tr>
<tr><td>0,1</td><td colspan="2">D - table relation avec clé simple ou lien avec clé étrangère</td><td>C - lien avec clé étrangère</td><td>E - table relation avec clé simple ou lien avec clé étrangère</td></tr>
</table>

a) intégrité référentielle

Le système des clés, primaires ou étrangères, garantit le fonctionnement des liens entre tables. Rappelons qu'aucune clé primaire ne peut avoir de valeur nulle. On trouve en fait quatre contextes remarquables où les clés interviennent.

CHAMBRE
PK_CHAMBRE
IDCHAMBRE
ETAGE

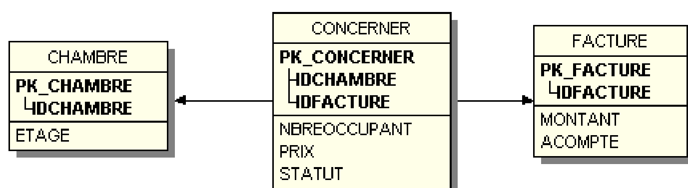
La clé primaire d'une table provenant d'une entité correspond à l'identifiant de cette dernière. Il s'agit le plus souvent d'une clé simple définie au niveau d'un attribut.

[3] Voir à ce sujet le chapitre sur la cohérence des données dans le cours consacré aux bases de données.

[4] Définies dans la norme étendue SQL2, les `ASSERTIONS` sont en fait de contraintes `CHECK` généralisées à l'ensemble de la base de données. Ni InterBase, ni SQL-Server ne les ont implémentées. Voici un exemple qui vérifie que le montant des commandes faites par un client ne dépasse sa limite de crédit :

```
CREATE ASSERTION Credit_Commandes
CHECK (Clients.LimiteCredit >=
   SELECT Sum(Commandes.Montant)
   FROM Commandes
   WHERE Commandes.Client = Clients.IdClient)
```

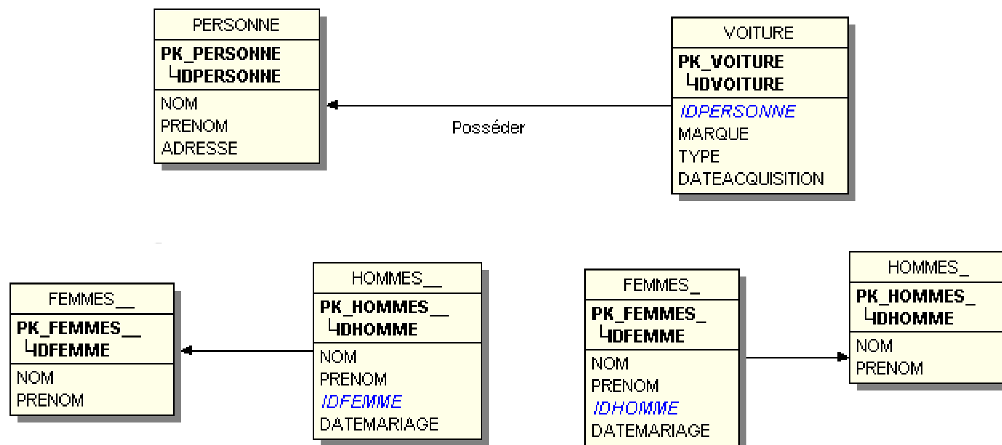
©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET



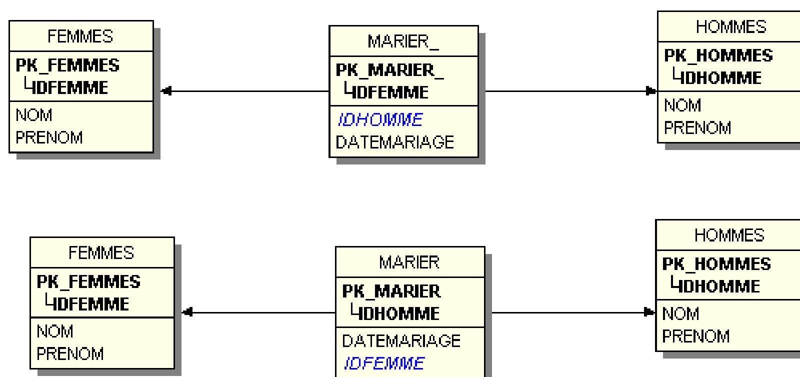
Dans le cas **A**[5] où une relation donne naissance à une table, nous avons une clé multiple qui consiste en une combinaison des identifiants de toutes les entités concernées. Aucun constituant de la clé ne peut être nul. Outre leur participation à cette clé primaire, chacun des composants sont des clés étrangères désignant une occurrence d'entité.

Lorsqu'une relation a été réduite à un simple lien dans une table d'entité (cas **B** et **C**), ce lien est matérialisé par un attribut qui sert de clé étrangère vers l'autre entité impliquée dans la relation.

Pour les cas **D** et **E**, on peut opter pour un lien (à condition d'accepter qu'il puisse prendre la valeur Null). Dans le cas E, ce lien pourra figurer indifféremment dans l'une ou l'autre table. Ce lien sera également une clé étrangère.



Si on opte pour une table-relation, elle aura une clé primaire simple. On retrouvera deux clés étrangères pointant vers les entités mises en relation. Dans ce cas, on devra en outre assurer l'unicité de la clé étrangère non clé primaire, à l'aide d'une contrainte `UNIQUE`. On emploie parfois le terme de « clé alternative ».



[5] Je reprends ici la numérotation déjà employée lors de l'examen des relations binaires.

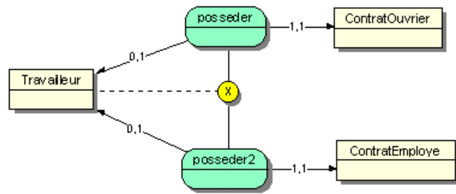
©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

b) valeurs par défaut

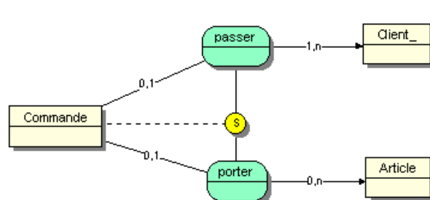
L'expression DEFAULT permet de donner une valeur initiale à une occurrence de propriété.

c) contraintes interrelation

Celles-ci concernent donc la participation d'une occurrence à plusieurs occurrences de relations potentielles. Il s'agit en fait d'une extension de la notion de cardinalité sur plusieurs pattes issues d'une même entité.

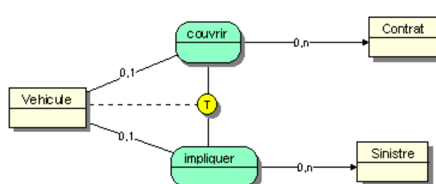


L'exclusion interdit à une occurrence d'entité de participer à plus d'une parmi plusieurs relations. Dans l'exemple[6], un travailleur peut posséder soit un contrat d'ouvrier, soit un contrat d'employé. La méthode la plus simple pour imposer cette contrainte est d'écrire deux triggers : le premier vérifiera lors de la création d'une occurrence de la relation *posséder1* que l'occurrence de travailleur n'est pas déjà impliquée dans une relation *posséder2*. Le second trigger fera la vérification symétrique.



La simultanéité suppose qu'une occurrence d'entité doit nécessairement participer à plusieurs relations. Il est en pratique impossible d'écrire des triggers pour rendre compte de cette situation. En effet, le trigger se déclenchera juste avant ou juste après la mise à jour de la table, et il faut forcément que la mise à jour commence par une des deux tables. On peut s'en tirer à l'aide d'une assertion (celle-ci ne sera vérifiée qu'à la fin de la transaction, donnant le loisir de modifier les deux tables avant vérification). On peut aussi, sur les nombreux systèmes qui ne connaissent pas les assertions, utiliser un trigger pour vérifier une des coexistences. Dans notre exemple, on vérifiera par exemple que si une commande porte sur un article, elle a été passée par un client. Cela implique qu'on devra toujours commencer par encoder la relation *passer*. Évidemment dans ce cas, rien n'empêchera qu'un client passe une commande sans article. On peut trouver des astuces pour régler le problème :

– par exemple, ajouter une propriété Valide à la table commande et mettre celle-ci à jour lors de la modification de la relation Passer ou de la relation Porter ;
– on peut écrire une requête qui recherche les commandes ne portant sur aucun article et la lancer en fin de journée ;
– il est encore possible de déclencher, par programme cette fois, une vérification de la présence d'un article au moins à la fin de la procédure d'encodage d'une commande[7].

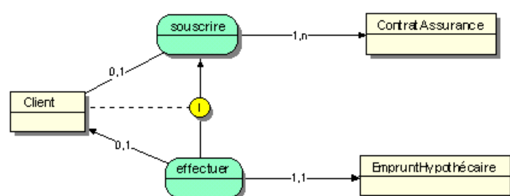


La totalité suppose qu'une occurrence d'entité participe à au moins une occurrence de relation. Ici encore, il est impossible d'écrire un trigger. Les règles d'intégrité interdisent la création des occurrences de *Couvrir* ou d'*Impliquer* avant la création de l'occurrence de *Véhicule* et un trigger lancé lors de l'insertion d'une occurrence de *Véhicule* ne pourra pas imposer l'existence d'une relation impliquant l'entité. On en est réduit à des vérifications *a posteriori*. Dans l'exemple, la présence d'un véhicule non couvert et non impliqué n'occasionnera pas beaucoup de désagrément , si ce n'est une perte de place. Les solutions proposées pour la simultanéité pourraient ici aussi s'appliquer.

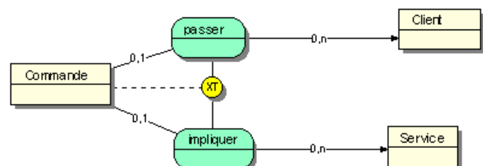
[6]Je reprends ici les exemples proposés lors de l'étude du modèle conceptuel des données.

[7]Il peut paraître surprenant qu'un utilisateur oublie d'encoder les articles. Mais il peut avoir été interrompu ou gêné par l'absence de l'article voulu. Une erreur est toujours possible.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET



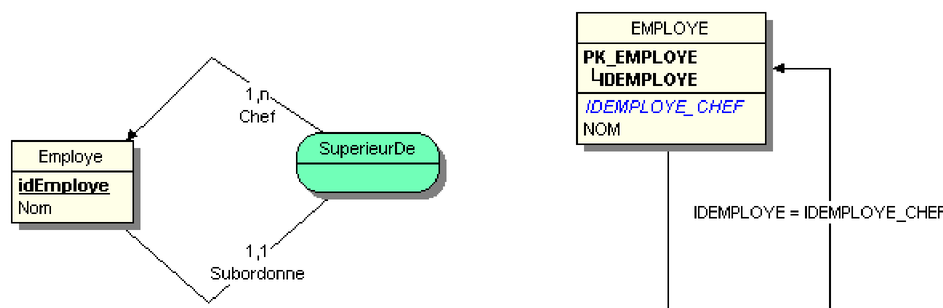
L'inclusion possède une orientation qui permet de résoudre les blocages des exemples précédents. Si quelqu'un effectue un emprunt hypothécaire, il doit forcément souscrire une assurance. On peut donc facilement placer un trigger qui vérifie l'existence d'une souscription avant d'effectuer l'emprunt. Il peut néanmoins sembler curieux de devoir introduire le contrat d'assurance dans la base avant l'emprunt qui le rend nécessaire. Il ne faut pas confondre l'intention d'emprunt (qui se passe effectivement avant tout contrat d'assurance, mais ne donne pas lieu à une trace dans la base, du moins dans les tables qui nous occupent ici) et la notification de l'emprunt qui ne peut s'effectuer que s'il existe un contrat préalable.



La partition combine une exclusion et une totalité. L'exclusion peut facilement être traitée par deux triggers, mais nous avons vu que la totalité ne pouvait pas facilement être rendue obligatoire par ce procédé. On en sera réduit à nouveau à des traitements applicatifs comme ceux que nous avons évoqués plus haut.

10.3.8 Contraintes complexes

La place et le temps manquent pour expliciter tous les traitements des contraintes qu'on peut avoir ajoutées au MCD. En général, on aura recours à des triggers, mais en prenant soin de ne pas provoquer des blocages. Certains cas simples débouchent parfois sur des modèles impossibles à gérer. Il suffira de prendre un cas classique. Soit une relation réflexive *SupérieurDe* définie sur une entité *Employés*.



Les règles vues plus haut nous suggèrent de ne garder qu'une table entité reprenant les employés et un lien réflexif (clé étrangère). On voit pourtant que la situation est inextricable. Si on se contente d'une contrainte de clé étrangère, nous ne pourrons pas avoir de clé nulle. Cela impliquerait que tout employé aura nécessairement un chef, ce qui nous ennuiera pour les cas de ceux qui se trouvent tout en haut de la hiérarchie[8]. Dans une telle manière de voir, on devra procéder à un encodage des employés en descendant l'arborescence de la hiérarchie, une méthode qui risque d'être extrêmement lourde dans une grosse entreprise. Une autre manière de voir serait de vérifier l'intégrité à l'aide d'un trigger et de ne l'activer que quand la base de données aura acquis son rythme de croisière. Une dernière solution consiste à garder une table *SupérieurDe* et de placer les vérifications à son niveau, de manière plus simple puisque les employés peuvent être créés indépendamment.

[8]La solution de les faire chefs d'eux-mêmes ne résout rien puisqu'ils n'existent pas encore dans la table au moment où la vérification se fait.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

On pourrait multiplier à l'envi l'examen des cas de cette espèce. Il suffit que l'informaticien soit conscient du problème et qu'il le résolve avec tout son bon sens. Cette dernière remarque pourra décevoir, mais il suffit d'examiner les solutions automatiques fournies par des logiciels comme dbMain ou WinDesign pour se rendre compte qu'il n'est pas pensable de traiter tout de manière algorithmique.

10.4 Quantification et répartition

L'informaticien sera à même de préciser la taille prise par les données et de tenir compte de l'espace nécessaire sur les mémoires de masse (on n'oubliera pas les index). Cette préoccupation avait une importance critique à l'époque où le Mo d'espace disque coûtait cher. Actuellement, une quantification précise sera nécessaire pour des applications de grand format manipulant des données de taille colossale. Nous savons que s'il ne s'agit que d'ajouter quelques Go, la dépense ne sera pas significative.

La répartition des données touche à la notion de machine logique (voir MLT). La règle la plus sage consiste à tenter de rapprocher physiquement les données de leur lieu d'utilisation, en prenant en compte les impératifs techniques et les types d'accès. Si certaines données doivent être dupliquées, on prendra soin de préciser quand et comment elles seront mises à jour. Le problème s'accentue quand les données doivent être disponibles sur des ordinateurs portables, qui ne sont pas connectés au reste du réseau en permanence. Il existe des solutions, notamment la *réplication* disponible dans certains SGBD. Celle-ci permet d'assurer une copie et une recollecte des données. Si le principe en est simple, une mise en œuvre efficace suppose une analyse sérieuse des risques et une anticipation des conflits, en cas de mises à jour concurrentes.

©2008-2009 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET