

Bases de données relationnelles et normalisation : de la première à la sixième forme normale

*Relationland*Par François de Sainte Marie ([fsmrel](#))

Date de publication : 07/09/2008 ; Mise à jour : 14/07/2011

*Labor omnia vincit improbus*
Virgile (Géorgiques)

La normalisation des tables (plus formellement relations) composant une base de données relationnelle est incontournable si l'on veut garantir à celle-ci :

- une **structure** la plus fiable et robuste qui soit, d'évolution facilitée quand les règles du jeu de l'entreprise changent ;
- Un contenu valide, par l'élimination de la **redondance** de l'information au sein de chaque table, redondance pouvant être la cause d'incohérences suite aux opérations de **mise à jour** de la base de données, ce que l'on oublie trop souvent.

Certes, la normalisation n'est pas la panacée, on pourra toujours injecter des redondances échappant à son contrôle, mais elle joue en tout cas un rôle décisif ne serait-ce qu'en matière d'architecture ; nous avons pu en juger pendant de très nombreuses années, sur le terrain et pour de très gros projets, que les bases de données soient relationnelles ou non.

La normalisation est un sujet dont l'étude n'est pas toujours simple, car elle a fait l'objet d'une théorie élaborée par des mathématiciens, pourvoyeurs en l'occurrence de bien des théorèmes, dont certains incontournables pour s'assurer de la structure correcte des bases de données. L'objet de cet article est de chercher à rendre le sujet abordable, l'expliquer sans employer pour autant le langage parfois hermétique de nos chers théoriciens, aussi suivrons-nous les guides les plus sûrs, à savoir C. J. Date & H. Darwen. Les informaticiens emboîterent le pas des théoriciens et le sujet — fort en vogue dans les années soixante-quinze / quatre-vingts — fit alors l'objet d'une vulgarisation pour le moins sujette à caution, et cela continue aujourd'hui. Aussi, nous ne pouvons ni ne voulons nous contenter des définitions incomplètes, fantaisistes ou fausses — touchant notamment aux formes normales — qui abondent dans de nombreux ouvrages et sur la Toile : rigueur et pertinence sont de mise, il y va de la validité et de la crédibilité des bases de données.

Suite aux discussions que nous avons eues sur les forums Developpez.com, nous avons tiré nombre d'enseignements quant aux questions que se posent certains qui découvrent la normalisation, ou d'autres qui ne l'avaient pas abordée par le bon côté (et pensent parfois qu'elle relève de la norme ISO...) Nous tentons ici de répondre au mieux à leurs attentes tout en formalisant un peu plus et en les creusant, certains points en général juste effleurés (données temporelles par exemple, en relation avec la sixième forme normale).

Bonne et fructueuse lecture.

N.B. Dans cet article nous étudions la normalisation dite par **projection/jointure**, donnant lieu aux formes normales connues sous le nom de première, deuxième, ..., sixième forme normale (sans oublier l'incontournable et particulièrement importante forme normale de Boyce-Codd). En revanche nous ne traitons pas d'autres formes normales, relevant par exemple de la normalisation par restriction/union, ou de la normalisation par contraintes clés/domaines.

Et merci aux relecteurs chez DVP : TheLeadingEdge (qui m'a suggéré d'écrire cet article), mikedavem, qi130, CinePhil, SQLpro, Luc Orient, ypicot, et tout particulièrement Antoun qui n'a pas ménagé sa peine (et la mienne non plus d'ailleurs...), sans oublier le vaillant, vigilant et prometteur Oishiini.

Questions et commentaires à propos de cet article sont les bienvenus. Rendez-vous autour de la [discussion](#) ad-hoc.

La Normalisation

Bases de données relationnelles et normalisation : de la première à la sixième forme normale	1
Avertissement.....	5
1. De la normalisation	7
1.1. Contexte.....	7
1.2. Retour aux sources	7
1.3. Rappel de quelques définitions.....	9
1.4. Objet de la normalisation	10
1.5. Étapes de la normalisation	11
1.6. Normaliser, une obligation ?	12
1.7. Dénormalisation vs amélioration (optimisation).....	14
2. Première forme normale	21
2.1. La situation en 1969.....	21
2.2. 1970 : Acte de naissance de la première forme normale	22
2.3. Années 1970-1980. Les théoriciens du Modèle Relationnel sont-ils en phase avec Codd ?	23
2.4. L'esprit et la lettre	24
2.5. L'atomicité : un critère absolu ?	25
2.6. Début des années quatre-vingt-dix. Les RVA (attributs dont les valeurs sont des relations).....	25
2.7. Pour conclure avec la première forme normale.....	30
2.8. Le bêtisier. Les définitions non conformes de la 1NF	31
3. Forme normale de Boyce-Codd, deuxième et troisième formes normales.....	33
3.1. Les états d'âme provoqués par la première forme normale	33
3.1.1. Une tentative de normalisation en 1NF.....	33
3.1.2. Conséquences de la normalisation : des redondances à profusion	33
3.1.3. Les difficultés de mise à jour (Insert, Delete, Update)	35
3.1.4. Quelle alternative ?	35
3.1.5. Approche conceptuelle de la solution.....	35
3.2. L'approche analytique, ses outils	37
3.2.1. Remarque préliminaire.....	37
3.2.2. Dépendance fonctionnelle (DF)	37
3.2.3. Clé candidate.....	39
3.2.4. Surclé.....	39
3.2.5. Sous-clé.....	40
3.2.6. Clé primaire.....	40
3.3. Forme normale de Boyce-Codd (BCNF)	41
3.3.1. Énoncé de la BCNF	41
3.3.2. Théorème de Heath	42
3.3.3. Comment normaliser une relvar qui n'est pas en BCNF.....	42
3.3.4. Décomposition par projection/jointure préservant l'information	43
3.3.5. Normalisation et intégrité référentielle.....	43
3.3.6. Conséquences de la normalisation par projection - jointure.....	44
3.4. Deuxième et troisième formes normales	44
3.4.1. BCNF versus 2NF et 3NF	44
3.4.2. Deuxième forme normale (2NF)	44
3.5. Dépendance transitive, dépendance directe	45
3.6. Troisième forme normale (3NF)	46

3.7.	Un problème embarrassant de BCNF	47
3.8.	Retour sur la dénormalisation <i>a priori</i>	51
3.9.	Une relvar respectant la BCNF peut-elle violer la 3NF ?	53
4.	Quatrième forme normale	56
4.1.	Au-delà de la BCNF	56
4.2.	Observations à propos de la jointure naturelle.....	56
4.3.	Dépendance multivaluée (DMV)	59
4.4.	Décomposition sans perte de données. Premier théorème de Fagin	60
4.5.	La dépendance fonctionnelle, cas particulier de la dépendance multivaluée (règle de réplication)	60
4.6.	Dépendance multivaluée triviale.....	60
4.7.	Quatrième forme normale (4NF)	61
4.8.	Un (sympathique) théorème de Date et Fagin concernant la 4NF.....	62
4.9.	4NF et relvars « toutes clés », une légende à détruire	63
4.10.	La 4NF et Merise	64
4.11.	Merise et la chasse aux ternaires.....	68
4.12.	Observations concernant la décomposition des relvars	70
4.13.	Approche ascendante vs approche descendante	73
4.14.	Implication de la BCNF par la 4NF	75
5.	Cinquième forme normale.....	76
5.1.	Introduction	76
5.2.	Dépendance de jointure	77
5.3.	Relvars formant un cycle.....	77
5.4.	Dépendance de jointure triviale.....	81
5.5.	Définition de la 5NF (PJ/NF).....	81
5.6.	Exemples de relvars respectant ou non la 5NF	82
5.7.	Parallèle entre la BCNF, la 4NF et la 5NF	83
5.8.	Un autre théorème de Date et Fagin	83
5.9.	Pour conclure avec la normalisation en 5NF	83
5.10.	Un petit exercice	84
6.	Sixième forme normale.....	86
6.1.	Introduction	86
6.2.	Définition de la sixième forme normale (6NF)	87
6.3.	Relvars à caractère temporel, purement historiques.....	87
6.3.1.	Un exemple.....	87
6.3.2.	Typage des intervalles, opérateurs	89
6.3.3.	Opérateurs PACK et UNPACK	90
6.3.4.	Opérateurs U_PROJECT et U_JOIN.....	94
6.3.5.	Opérateurs de comparaison relationnelle généralisés	96
6.3.6.	Dépendance de jointure généralisée	97
6.3.7.	Retour sur la 6NF, place à la concision	98
	Remarque préalable	98
	2e impératif LDD	98
6.3.8.	Déclaration des relvars.....	99
6.4.	Données actives et données historisées.....	100
	Premières tentatives de modélisation	100
	5e impératif LDD	101
	6e impératif LDD	102
	1er impératif LDD	102
	3e impératif LDD	103

4e impératif LDD	103
Poursuite du processus d'historisation	103
6.5. Points particuliers.....	106
6.5.1. A propos de l'intégrité référentielle et des contraintes d'intégrité en général.....	106
6.5.2. Choix du mode d'historisation	107
6.5.3. Relvars « associatives »	107
6.5.4. Trois relvars ou une relvar unique ?	107
6.6. Pour conclure avec la normalisation en 6NF	108
Annexes	110
A. Tuples, relations, relvars (définitions)	110
A.1. Tuple (n-uplet), attribut	110
A.2. Relation, relvar	110
A.3. Note à propos de Tutorial D.....	111
B. Notation des opérateurs relationnels (Tutorial D).....	113
C. Notes concernant la première forme normale et la logique du deuxième ordre	117
C.1. Univers du discours	117
C.2. Termes généraux	118
C.3. Concept de classe	119
C.4. Vers la logique du deuxième ordre : termes généraux considérés comme des objets.....	120
C.5. L'exemple de la généalogie et ses applications	121
C.6. Au sujet des RVA de Date et Darwen	122
D. La normalisation et le bonhomme NULL	123
E. Fermeture des dépendances fonctionnelles, axiomes d'Armstrong, ensemble irréductible	125
E.1. Fermeture d'un ensemble de dépendances fonctionnelles	125
E.2. Axiomes d'Armstrong	125
E.3. Application des axiomes, calcul de la fermeture des DF	126
E.4. Fermeture d'un ensemble d'attributs, l'algorithme du seau à dépendants	127
E.5. Inventaire des clés et surclés. Quelques observations.	130
E.5.1. La technique du rouleau compresseur	130
E.5.2. Cas des attributs ne figurant pas dans les dépendants des DF	130
E.5.3. Surclés n'ayant pas d'attributs en commun et utilisation de l'algorithme du seau	131
E.5.4. Clés oubliées	131
E.6. Ensemble irréductible de dépendances fonctionnelles.....	132
E.6.1. Ensembles de DF et redondances	132
E.6.2. Propriétés d'un ensemble irréductible de DF.....	132
E.6.3. Pluralité des ensembles irréductibles pour une relvar	136
E.7. Décompositions sans perte	144
E.7.1. Préservation du contenu de la base de données	144
E.7.2. Préservation des dépendances fonctionnelles	146
E.8. Conclusion	148
F. Identification relative versus identification absolue	149
F.1. Consommation des ressources physiques	149
F.2. Performance des applications.....	150
F.3. Le clustering selon DB2 for z/OS	151
F.4. L'identification relative au service de l'intégrité des données	154
Bibliographie.....	156

Avertissement

A propos de Tutorial D

Le thème de la normalisation est traité ici dans le cadre du Modèle Relationnel de Données, inventé par [Ted Codd](#) et qui continue à s'enrichir et évoluer harmonieusement sous la houlette de ses continuateurs, citons Chris Date, compagnon de route de toujours de Ted, Hugh Darwen, et pour certains points sensibles, avec la participation de David McGoveran ou Nikos Lorentzos. Le Modèle Relationnel a de nombreux points en commun avec le Modèle SQL qui s'en est inspiré, mais il y a des différences très marquées tant sur le fond que sur la forme. Concernant la structure des données, ces différences seront signalées au fur et à mesure quand le besoin s'en fait sentir.

On trouvera en annexe (paragraphe A) la définition des concepts structurels fondamentaux propres au Modèle Relationnel et quelques notes concernant le langage **Tutorial D**, prototype relationnellement complet (cf. [Date 2006]). On y trouvera aussi (paragraphe B) quelques notes relatives à l'algèbre relationnelle, permettant de comprendre par exemple comment les opérations relationnelles sont exprimées en Tutorial D, avec au besoin, la traduction en SQL de ces opérations.

A qui s'adresse cet article ?

S'il n'est évidemment pas nécessaire que le lecteur sache tout du Modèle Relationnel, il est néanmoins préférable qu'il ait un minimum de connaissances, disons en SQL :

Savoir ce qu'est une table (aspect structurel), comment l'exploiter (utilisation des opérateurs relationnels figurant plus ou moins explicitement dans le bloc SELECT-FROM-WHERE), et comment garantir un strict minimum d'intégrité des données (clés primaires et étrangères).

Même s'il ne connaît que très peu SQL, celui qui construit des diagrammes au niveau conceptuel (MCD selon l'approche Merise ou Entité/Relation, IEF, voire diagramme de classes), est très concerné, puisque les tables qui constitueront la base de données seront le résultat de la dérivation de ce qu'il aura construit.

La normalisation permettra de s'assurer que les fondations de la base de données sont saines et robustes.

Tout en étant incontournable, la normalisation peut être considérée comme un sujet indépendant (orthogonal comme dirait Date), au point que Codd — qui connaît mieux que quiconque la chose — n'en parle dans son ouvrage de référence [Codd 1990] que pour rappeler l'intérêt qu'il y a à normaliser afin d'éviter les anomalies de mise à jour (deux pages et c'est tout). Quoi qu'il en soit, j'ai constaté que nombreux sont les « spécialistes » qui traitent du sujet de façon imparfaite, n'ayant vraisemblablement pas eu l'occasion — sinon la curiosité — de mesurer les conséquences funestes de leur légèreté, au préjudice de la Production informatique.

Cela dit, l'objet de l'article est quelque part de faire en sorte que le lecteur intéressé dispose de définitions **exactes** des formes normales, pour mieux bétonner sa base de données. J'essaie en l'occurrence de rester lisible car il est facile d'écrire de façon très hermétique sur le sujet, mais là n'est certainement pas le but de la manœuvre.

Le thème de l'« optimisation » des bases de données est aussi abordé, mais on sort alors carrément du cadre de la théorie relationnelle, ce qui fait que tous les administrateurs de bases de données (DBA) ne seront pas forcément convaincus par mon approche — basée quand même sur quarante années d'exercice du métier, dans les secteurs d'activité les plus variés, avec tous les types de SGBD, dans le contexte des très grandes bases de données et le plus souvent au fond de la soute — (cf. le paragraphe F en annexe, dont la rédaction fut provoquée par une réflexion sur ce thème de l'optimisation, en réaction à la confusion faite avec celui de la dénormalisation, confusion entretenue (sans doute bien involontairement) par certains auteurs et « experts » autoproclamés (se reporter au paragraphe 1.7, notamment à ce qui a trait à l'identification relative)).

Périodiquement, on relève chez Developpez.com (Mâtin, quel site !) des questions posées par des étudiants, questions ayant trait à la normalisation, mais sur des aspects plutôt théoriques et qui ne concernent que modérément les praticiens. A leur intention, quelques réponses sont fournies en annexe (cf. paragraphes E.6 et E.7).

Concernant le vocabulaire

Lorsqu'il a inventé le Modèle Relationnel de Données en 1969, l'objet de l'étude de Codd était celui des relations. En 1974-1976, en créant SEQUEL (ou SQL si vous préférez), Chamberlin a préféré remplacer « relation » par « table » (mot sans doute plus parlant). Mais il y a toujours intérêt à rechercher le mot le plus approprié pour un concept, par souci justement de précision. Ainsi, en 1994, Date et Darwen (D & D, que j'appelle encore affectueusement les Dupondt) en sont arrivés à définir le concept de **variable relationnelle** (*relation variable*, en abrégé **relvar**). Ils ont estimé — à l'instar des autres théoriciens du relationnel, du reste — qu'une relation étant une **valeur**, elle ne se situe ni dans le temps ni dans l'espace (Platon aurait approuvé), contrairement à la variable relationnelle, à laquelle on affecte, par le biais d'une requête (disons au sein d'un programme, d'une procédure ou de ce que vous voulez), des valeurs différentes (c'est-à-dire des relations) au fil du temps. Ainsi en va-t-il dans un programme, quand à la variable x on affecte les valeurs 1 ou 2, etc. qui, elles non plus, ne se situent ni dans le temps ni dans l'espace et dont nous ne représentons que des images.

Au contraire, en SQL on ne dispose que d'un concept unique, celui de table. Mais, quand une table est-elle une variable ? Une valeur ? L'ambiguïté peut devenir fort gênante.

J'aurais pu faire l'économie du terme *relvar* et me contenter de l'expression *schéma de relation* (comme l'on pratiquait du reste dans les années soixante-dix, quatre-vingts), mais un tel schéma (*en-tête* selon le Modèle Relationnel de Données) est en fait un des composants de la relvar, ça n'est qu'un ensemble d'attributs (au sens de la théorie des ensembles). Bref, quand on y a pris goût, on ne peut plus se passer de la relvar...

On pourrait encore poser la question : Mais pourquoi traiter de la normalisation en faisant appel au vocabulaire de Tutorial D (qui est celui du Modèle Relationnel) plutôt qu'à celui de SQL ? Je répondrai encore que c'est au nom de la précision dans la définition des concepts mis en jeu.

Par exemple, on pourrait énoncer ainsi la forme normale de Boyce-Codd (BCNF) :

Une relvar R est en forme normale de Boyce-Codd (BCNF) si et seulement si les seules dépendances fonctionnelles non triviales qu'elle doit vérifier sont de la forme $X \rightarrow Y$, où X représente une surclé et Y un sous-ensemble d'attributs de l'en-tête de R .

C'est du béton, même si à ce stade vous ne pouvez peut-être pas encore en juger. A la sauce SQL, on pourrait reformuler cela à peu près ainsi :

Un schéma de table T^* est en forme normale de Boyce-Codd (BCNF) si et seulement si les seules dépendances fonctionnelles non triviales qu'elle doit vérifier sont de la forme $X \rightarrow Y$, où X contient une clé candidate et Y représente un sous-ensemble de colonnes de T^* .

Je ne suis pas sûr qu'il n'y aurait pas de grincheux pour dire que ça n'est pas une très bonne formulation, que cela manque de rigueur (par exemple, quel sens exact prend le verbe *contenir* dans l'expression « contient une clé candidate » ?)

Quoi qu'il en soit, je trouve plus difficile de dire les choses en termes SQL, aussi Tutorial D mérite-t-il qu'on s'y intéresse, même s'il ne s'agit pas ici de l'étudier pour lui-même.



Veuillez prendre note que je ne fournis pas toujours le texte original de certaines citations mais seulement leur traduction. On n'est jamais à l'abri d'une bévue, mais je pense en avoir respecté l'esprit.

1. De la normalisation

1.1. Contexte

Quand on parle de bases de données relationnelles, on évoque inmanquablement les trois piliers qui constituent les fondements de la théorie relationnelle et ayant pour objet :

- 1° La **structure des données**, c'est-à-dire, si l'on se situe au niveau SQL, les règles de définition des tables en termes de lignes et de colonnes ;
- 2° La **manipulation des données** : comment exploiter ces tables, à l'aide par exemple — toujours dans le contexte SQL — de l'incontournable triplet SELECT, FROM, WHERE et des opérateurs INSERT, etc. ;
- 3° L'**intégrité des données**, c'est-à-dire les moyens mis à notre disposition par le SGBD, concourant à la validité de ces données, tels que les clés primaires, clés étrangères et contraintes diverses (assertions et triggers si SQL...)

Il existe par ailleurs un volet extrêmement important concernant les bases de données relationnelles, celui de la **normalisation**, dont l'objet est double :

- A l'intersection d'une ligne et d'une colonne, certes on trouve des données de type très simple, telles que les habituels nombres et chaînes de caractères, mais peut-on aussi légalement trouver des données de types plus complexes, telles que des listes, des tableaux, des tables, etc. ? La normalisation a pour objet de confirmer les règles du jeu à ce sujet, en relation avec les effets que cela peut avoir sur chacun des trois piliers précédents.

Cela concerne au premier chef ce qu'il est convenu d'appeler la Première Forme Normale (1NF) et sera développé dans le paragraphe 2 « Première forme normale ».

- La normalisation a aussi pour objet de fournir les outils et les techniques nous permettant de débusquer, d'éliminer les **redondances** qui non seulement rendent les tables obèses, mais par ailleurs nous compliquent la vie lors des opérations qui les mettent à jour (mises à jour nécessairement redondantes elles aussi) et finissent par rendre faux le contenu de la base de données, sans parler de l'effet néfaste sur les performances. Par voie de conséquence, en normalisant, tout en éliminant ce genre d'*impedimenta*, on améliore l'architecture de la base de données, ce qui n'est pas un mince avantage, ne serait-ce que si l'on a un jour à faire évoluer celle-ci.

Cela concerne les formes normales suivantes (2NF à 5NF) et sera développé dans les paragraphes 3 et suivants. Vu sa spécificité (garde-fou dans la manipulation des données intervallaires), la 6NF sera traitée dans le contexte de la modélisation des données temporelles.

L'objet de la normalisation est repris dans le paragraphe 1.4.

1.2. Retour aux sources

Tout d'abord, il sera régulièrement fait ici référence à Ted Codd (1923-2003), le génial inventeur du Modèle Relationnel de Données, qui a tout de suite traité de la normalisation, de manière très rigoureuse.

Ensuite, bien que Ted Codd ne traite que des **relations**, suivant le contexte, on utilisera aussi par la suite les termes « **table** » et « **relvar** » (voir [ci-dessous](#) : « Relvar, relation et table »). En attendant, commençons par rappeler ce qu'est une relation dans le cadre de la théorie relationnelle.

Alors que SQL n'était pas encore né, et pour cause — et *a fortiori* le concept SQL de table — voici la définition donnée par Ted Codd de la relation dans son article fondateur (cf. [Codd 1970], paragraphe 1.3) :

« Le terme *relation* est utilisé ici dans son acception mathématique. Étant donnés les ensembles S_1, S_2, \dots, S_n (non nécessairement distincts), R est une relation sur ces n ensembles si c'est un ensemble de n -uplets, le 1er élément de chacun d'eux tirant sa valeur de S_1 , le 2e de S_2 , et ainsi de suite (de manière plus concise, R est un sous-ensemble du produit cartésien $S_1 \times S_2 \times \dots \times S_n$). On fera référence à S_j comme étant le j ème domaine de R .

Suite à ce qui vient d'être énoncé, on dit que R est de degré n . Les relations de degré 1 sont souvent dites unaires, celles de degré 2 binaires, de degré 3 ternaires, et celles de degré n n -aires. »

Une représentation imagée d'un n -uplet d'une relation R de degré n , construit sur les domaines S_1, S_2, \dots, S_n :

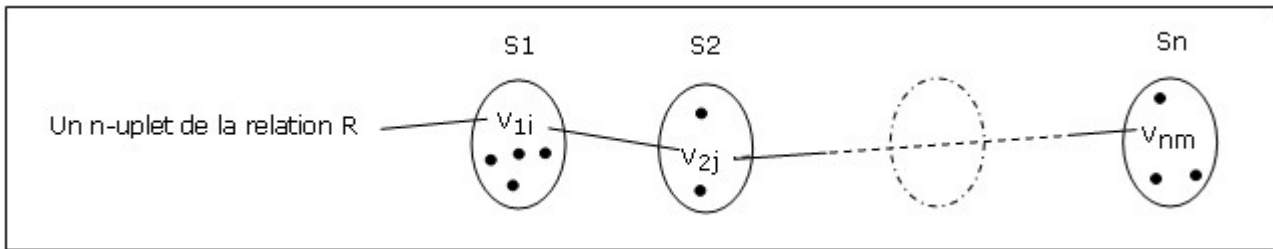


Figure 1.1 - Un n -uplet, comme des perles qu'on enfle

Codd poursuit :

« Pour simplifier l'exposé, on utilisera souvent une représentation des relations sous forme de tableaux ... Un tableau représentant une relation a les propriétés suivantes :

- (1) Chaque ligne représente un n -uplet de R ,
- (2) L'ordre des lignes n'a aucune importance,
- (3) Toutes les lignes sont distinctes,
- (4) L'ordre des colonnes est significatif — il correspond à l'ordre S_1, S_2, \dots, S_n des domaines sur lesquels sont définis les domaines de R .
- (5) La signification de chaque colonne est en partie rendue en l'affectant du nom du domaine correspondant. »

Représentation d'une relation R de degré n , sous forme imagée, rectangulaire, plate et traditionnelle :

R	S_1	S_2	\dots	S_n
	v_{11}	v_{21}		v_{n1}
	\dots	\dots		\dots
	v_{1i}	v_{2j}		v_{nm}
	\dots	\dots		\dots

Figure 1.2 - Une relation sous forme de tableau



Mais attention, l'image n'est pas la chose !

Dans ce qu'a écrit Codd, un point important peut paraître aujourd'hui choquant : il est en effet précisé que l'**ordre des colonnes** est significatif (point 4). Cela vient du fait que, dans ce tout premier jus du Modèle Relationnel, une colonne n'a pas de nom en propre, elle hérite implicitement de celui de son domaine de référence : contexte mathématique oblige. Ce n'est qu'en 1971 (cf. [Codd 1971], page 31) qu'apparaît le concept d'attribut, débarrassant le Modèle Relationnel de cette fâcheuse contrainte. Je cite (en rappelant que l'avatar SQL de la *relation* est la *table* et que le *degré* d'une relation correspond au nombre de colonnes d'une table) :

« Les n domaines ne sont pas nécessairement distincts. Plutôt qu'utiliser un ordre pour déterminer chaque domaine référencé (comme cela se fait en mathématiques), on utilisera un nom distinct pour chaque référence faite et nous l'appellerons nom de l'attribut... En conséquence, chaque référence faite à un domaine lors de la définition de R est appelée attribut de R . Par exemple, une relation de degré 3 pourrait avoir pour attributs (A_1, A_2, A_3) tandis que les domaines correspondants pourraient être les domaines (D_5, D_7, D_5). Les noms d'attributs sont un moyen d'éviter d'imposer aux utilisateurs la connaissance de la position des domaines. »

1.3. Rappel de quelques définitions

Les définitions qui suivent reprennent celles de Chris Date (cf. le paragraphe A en annexe).

- Un **domaine** tel que S_j est un ensemble de valeurs, par exemple celui des entiers, celui des chaînes de caractères, ceux des dates, des points, des lignes, des ellipses, polygones, des numéros de Siret, des codes postaux, des ISBN, des EAN13, etc. A noter qu'aujourd'hui on utilise le terme **type** plutôt que le terme domaine.
- Un **n-uplet** (ou **tuple**) est une valeur. C'est un ensemble de triplets de la forme $\langle A_i, D_i, v_i \rangle$ où A_i désigne un **nom d'attribut**, D_i désigne un nom de domaine et v_i une valeur appartenant au domaine D_i . Le couple $\langle A_i, D_i \rangle$ est un **attribut** du n-uplet ; v_i est la **valeur d'attribut** de l'attribut A_i ; le domaine D_i en est le **domaine d'attribut** correspondant (**type d'attribut**).

Exemple graphique, plat : un n-uplet composé des triplets $\langle \text{Attr1}, D1, a11 \rangle, \langle \text{Attr2}, D2, a21 \rangle, \dots, \langle \text{Attrn}, Dn, an1 \rangle$

Attr1 : D1	Attr2 : D2	...	Attrn : Dn
a11	a21	...	an1

Figure 1.3a - Un n-uplet au format tabulaire

Autre exemple plus parlant de n-uplet : un certain membre chez Developpez.com

Membre : Char	Statut : Char	...	Localisation : Char
Antoine	expert	...	Paris

Figure 1.3b - Autre exemple de n-uplet au format tabulaire

- Une **relation** est une valeur. Plus précisément, c'est une valeur constituée d'un **en-tête** (ou schéma ou intension, notez l'orthographe) et d'un **corps** (extension). L'en-tête est un ensemble de n **attributs**. Le corps est l'ensemble des **n-uplets** composant la relation.

Exemple de représentation sous forme tabulaire d'une relation n -dimensionnelle R (tout en rappelant que l'image d'une chose n'est pas la chose) :

R	Membre : Char	Statut : Char	...	Localisation : Char	← En-tête de la relation R Corps de la relation R
	Antoine	expert	...	Paris	
	Philou	confirmé	...	Atlanta	
	
	Frédéric	expert	...	Paca	

Figure 1.4 - Composants d'une relation

Dans un contexte informel, il est courant de ne pas faire figurer le nom des domaines dans [l'image de] l'en-tête :

R	Membre	Statut	...	Localisation
	Antoine	expert	...	Paris
	Philou	confirmé	...	Atlanta

	Frédéric	expert	...	Paca

Figure 1.5 - Représentation informelle d'une relation

Relvar, relation et table

Au vu de ces représentations, on pourrait penser qu'une **table** SQL est une relation (en remplaçant les termes « attribut » et « n-uplet » respectivement par « **colonne** » et « **ligne** »). Ça n'est pas le cas, car (outre bon nombre de propriétés non nécessairement partagées) une table peut changer de valeur, tandis qu'une relation **est** une valeur, donc par définition invariable, tout comme les entiers 1 ou 2. L'aspect variable des choses concerne la **variable relationnelle** (en abrégé **relvar**), type de variable affectée successivement de valeurs qui sont des relations : une relation y remplace une autre lors d'une opération de mise à jour. Notons que Codd n'utilisait pas le terme relvar, mais l'expression « time-varying relation », qui n'est plus jugée pertinente aujourd'hui, du fait du caractère justement invariable des relations.

Et n'oublions pas qu'une table SQL peut n'être qu'un sac (*bag*), dans la mesure où la présence d'une clé (disons primaire) n'est pas exigée, ce qui autorise l'existence de lignes en double (or un sac n'est pas un ensemble).

1.4. Objet de la normalisation

a) *A propos de la première forme normale (dont l'étude est développée dans le paragraphe 2).*

Les relations sont des êtres mathématiques. Elles sont soumises à certaines contraintes structurelles et leur finalité est d'être manipulées, combinées, à l'aide de l'**algèbre relationnelle** ou du **calcul relationnel** (qui est une application du calcul des prédicats). Ayant une préférence pour le calcul des prédicats, Ted Codd a raisonné en logicien. Nous verrons à l'occasion de l'étude de la première forme normale, qu'en 1969, il se plaça d'entrée dans le cadre de la logique du deuxième ordre, jugeant l'année suivante que la logique du premier ordre suffisait pour manipuler les relations¹. L'adéquation du calcul relationnel (et par contrecoup de l'algèbre relationnelle) à la logique du premier ordre eut pour conséquence une contrainte forte, conduisant à normaliser les relations en ce qu'il est convenu d'appeler la **première forme normale** (1NF), selon laquelle une relation ne peut pas être une valeur pour un attribut d'une autre relation : par exemple, les lignes de factures d'une facture ne peuvent pas être des valeurs d'un attribut *LigneDeFacture* d'une relation *Facture* (voir toutefois le cas des RVA, au paragraphe 2.6).

Certes, avec des systèmes comme IMS/DL1, par construction (modèle hiérarchique oblige), les lignes de facture sont nichées dans les factures, les engagements sur lignes de facture sont nichés dans les lignes de facture, etc. Mais IMS/DL1 ne permet pas de manipuler des **ensembles** à l'aide d'une algèbre ou d'un calcul, on est à un niveau inférieur où l'on ne traite qu'un **enregistrement** à la fois et, dans ces conditions, il n'y a pas de contrainte particulière quant à la façon de structurer les données.

b) *A propos des autres formes normales (dont l'étude est développée dans les paragraphes 3 et suivants).*

Ce que l'on appelle deuxième forme normale, troisième forme normale et forme normale de Boyce-Codd sont les éléments d'une théorie, d'abord développée par Codd dès 1970, puis complétée par Raymond Boyce (trop tôt disparu en 1974). Sept ou huit ans après que Codd l'eut entamée, des mathématiciens comme Jorma Rissanen et Ronald Fagin prirent le relais pour compléter la théorie de la normalisation, ce qui fut fait en 1979 avec la mise à notre disposition des quatrième et cinquième formes normales (et de la sixième, vingt ans plus tard).

Pour reprendre ce qui a été évoqué au paragraphe 1.1, respecter ces formes normales a pour effet (entre autres choses) de débarrasser les relations de **redondances** non seulement inutiles et causes d'obésité, mais surtout génératrices d'erreurs eu égard aux règles de gestion des données de l'entreprise, lors des opérations de **mise à jour** (disons INSERT, UPDATE, DELETE). Ces redondances sont le plus souvent la conséquence d'une modélisation conceptuelle en amont maladroite, voire inexistante, ou encore le mauvais fruit d'une « dénormalisation » inopportune (*horresco referens...*)

1. Les mérites respectifs des logiques du premier et deuxième ordre sortent du champ de cet article. Nous renvoyons le lecteur intéressé à l'ouvrage très didactique et non hermétique de Willard V.O. Quine [Quine 1972], chapitre 43 "Les classes" et suivants. Voir aussi dans l'article, le paragraphe C en annexe.

c) Observations concernant la modélisation conceptuelle.

Lorsqu'on représente les données sous forme graphique : modèles conceptuels de données (MCD) de la méthode Merise, et plus généralement diagrammes entités/reliations (voire diagrammes de classes), il y a tout un travail de vérification concernant chaque type d'association (ce qu'on désigne encore par association-type ou relation-type) entre types d'entités, consistant à « s'assurer que chacune des propriétés ne peut être vérifiée sur un sous-ensemble de la collection de la relation-type » [TRC 1989]. Attention, dans cette citation, la relation-type en question n'a rien à voir avec la relation du Modèle Relationnel, il s'agit de l'association (*relationship*) existant entre entités-types. Ce travail de vérification — portant lui aussi le nom de normalisation — conduit à expulser au besoin une propriété d'une association-type vers une entité-type (ou inversement). Ceci a à voir avec ce que Codd appelle la normalisation en deuxième forme normale (2NF), laquelle a en vérité une portée bien plus étendue, car elle concerne l'ensemble des relvars composant une base de données relationnelle. La 2NF est aussi beaucoup plus formelle quant à son énoncé.

La normalisation joue un rôle crucial quant à la qualité de l'architecture de la base de données, laquelle doit être structurellement valide et apte à évoluer, premièrement par le recours à une démarche au niveau conceptuel **synthétique, descendante** (donc en amont), à l'aide par exemple de la méthode Merise (démarche valant également pour les diagrammes de classes), deuxièmement par une vérification rigoureuse, mettant en jeu une démarche **analytique, ascendante**, pour laquelle on s'appuie justement sur la théorie de la normalisation : l'architecture de la base de données relève ainsi d'une approche mixte où l'on pratique l'art du yoyo, en alternant intelligemment les deux démarches.

d) Prise en compte des données temporelles (voir au paragraphe 6).

Ensemble, Hugh Darwen, Nikos Lorentzos et Chris Date, compagnon de route, fils spirituel (et parfois rebelle) de Ted Codd, ont enrichi la théorie relationnelle, en approfondissant avec une extrême rigueur le domaine des bases de données **temporelles**, et en nous fournissant les techniques pour nous y lancer à notre tour, autrement qu'à l'instinct, comme c'est hélas trop souvent le cas, ou sur la base de travaux théoriques jugés défectueux (cas de TSQL2 qui fut en son temps proposé pour être intégré à la norme SQL/2). C'est un sujet d'étude essentiel, que tous les concepteurs devraient approfondir, car la prise en compte du temps dans les bases de données a été, et reste quelque chose de compliqué et d'omniprésent, au moins dans le monde mouvant et agité de l'assurance, de la banque, de la grande distribution, de la retraite, et j'en passe.



Maintenant, comme dit Chris Date : « Normalization is **no panacea**, but it's a lot better than the alternative! »

1.5. Étapes de la normalisation

L'usage veut que l'on normalise en procédant par étapes : dans un premier temps, on s'assure que l'on respecte ce que l'on appelle la première forme normale (1FN ou 1NF) déjà évoquée (ceci concerne les tables SQL et apparentées, car pour leur part les relvars sont *de facto* en 1NF).

Ensuite, on s'assure que chaque **relvar** (ou table SQL), respecte ce que l'on appelle la 2e forme normale (2FN ou 2NF), puis la 3e forme normale (3FN ou 3NF), la forme normale de Boyce/Codd (FNBC ou BCNF), la 4e forme normale (4FN ou 4NF), la 5e forme normale (5FN ou 5NF) encore appelée PJ/NF (*Project/Join Normal Form*, forme normale par projection/jointure). La **projection** et la **jointure naturelle** (cf. le paragraphe B en annexe) sont les deux opérations utilisées tout au long du processus (d'où l'expression « normalisation par projection/jointure »). La projection est utilisée pour remplacer une relvar R qui ne respecte pas la xNF par deux relvars (ou plus dans le cas de la 5NF) R1 et R2 qui la respectent, et la jointure naturelle est utilisée pour recomposer très exactement R à partir de R1 et R2 au cas où le besoin s'en ferait sentir (par exemple au moyen d'une **vue**, ce qui permet de respecter le principe de l'**indépendance logique des données**, en garantissant la simplification de la manipulation des données par l'utilisateur, une stabilité de la représentation de celles-ci, tout à fait profitable pour les applications, etc.) Pour les données temporelles (plus généralement intervallaires), il est très vivement recommandé de pousser jusqu'à la 6e forme normale (6FN ou 6NF), qui marque la fin du processus de normalisation.

On notera qu'une relvar en 2NF est nécessairement en 1NF, qu'une relvar en 3NF est nécessairement en 2NF, etc., d'où la traditionnelle représentation graphique (enrichie de la 6NF) :

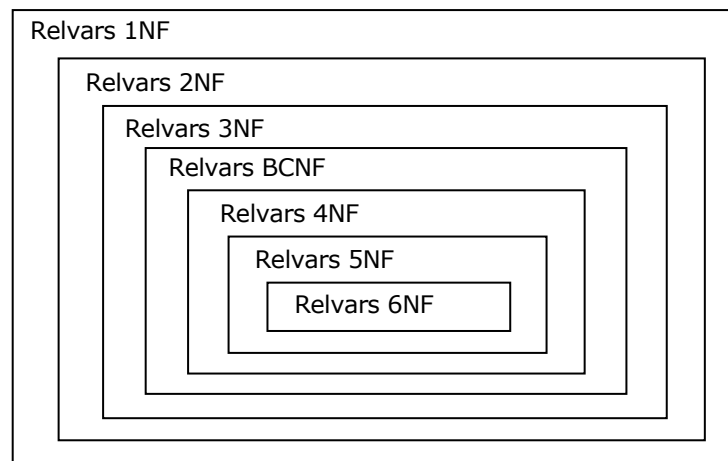


Figure 1.6 - Les relvars à la manière des matriochkas

Quand on a acquis un certain entraînement, on peut s'intéresser directement à la BCNF et se dispenser des étapes consistant à s'assurer que l'on est en 2NF et 3NF. Quant à la 4NF et à la 5NF, la partie est réputée assez difficile et l'on fait souvent l'impasse, en espérant que le MCD (modèle conceptuel de données) ou le diagramme de classes que l'on a réalisés soient normalisés, ce qui est en principe le cas avec des concepteurs expérimentés, mais gare quand même aux surprises (cf. paragraphe 4.11). Quant aux Sotomayor de la normalisation, ils placent la barre à la hauteur de la BCNF histoire de d'échauffer, puis attaquent directement la barre de la 5NF. Les barres intermédiaires ne les intéressent pas.

1.6. Normaliser, une obligation ?

Par construction chaque relvar d'une base de données respecte la 1NF, et chaque table SQL doit évidemment en faire autant. Normaliser en 2NF et au-delà, est très vivement **recommandé**, même si d'un point de vue théorique ça n'est pas une obligation stricte, dans la mesure où l'algèbre relationnelle n'est pas affectée.

Mais nombreux sont ceux qui, sous l'emprise de l'émotion ou par crédulité (lecture de la presse du coeur informatique, ragots de cafétéria, influence des légendes en tous genres colportées depuis l'arrivée des premiers SGBDR ...), préconisent une **non-normalisation a priori** et se limitent au respect de la 1NF. En effet, la normalisation conduit à casser une relvar en deux ou plusieurs relvars, en conséquence de quoi, « dénormaliser » (réassembler les morceaux) serait synonyme d'**optimiser**. Ceux-là devraient avoir à l'esprit cette réflexion de [Donald Knuth](#) (qui l'a peut-être empruntée à Tony Hoare) : « Premature optimization is the root of all evil. » Quelques variantes des arguments avancés par les ignorants :

- A cause de la **jointure**, les requêtes deviendraient compliquées. Developpez.com abonde en commentaires du genre : « D'accord, je vais faire comme vous dites, mais ça complique les requêtes ». Pour sa part, le DBA se fera un plaisir d'encapsuler ces requêtes dans des **vues**, lesquelles seront pour l'utilisateur des relvars comme les autres (respect du principe de l'**indépendance logique**).
- Recomposer une relvar à partir des « morceaux » fait une fois de plus intervenir l'opération de **jointure**, que l'on qualifie hâtivement de non performante, en toute méconnaissance de cause, c'est-à-dire en confondant allègrement le niveau logique et le niveau physique. Certes, selon un raisonnement simpliste et en se plaçant au niveau physique, si les enregistrements impliqués ne sont pas dans la même page (bloc physique) sur le disque, la jointure serait source d'accès au disque (ou au cache) supplémentaires, mais son statut d'opération relationnelle par excellence fait qu'elle est l'objet de tous les soins de l'optimiseur des SGBD relationnels dignes de ce nom et qu'il n'y a pas lieu de s'effoler. Se reporter à ce sujet au paragraphe 3.8. La jointure a bon dos, les problèmes de performance sont ailleurs.

Dénormaliser n'est pas interdit, mais on en connaît aussi les inconvénients, par exemple :

- Dégradation de la qualité de la modélisation que l'on finit par ne plus maîtriser et faire évoluer proprement.
- Nécessité de mettre en œuvre des contraintes (assertions ou triggers en SQL) garantissant sous le capot le respect de certaines dépendances fonctionnelles, multivaluées, etc., qui sont les conséquences du non respect de la normalisation.

- Anomalies potentielles de mise à jour, obésité des tables SQL due à l'inflation des redondances, à leur enneigement (données utiles clairsemées, noyées au milieu de tombereaux de *nulls* — lesquels ne facilitent pas la tâche des optimiseurs —, et de valeurs par défaut plus ou moins pertinentes).
- Parce qu'un SGBD comme DB2 ne respecte qu'en partie le principe de l'**indépendance physique** (à chacun de voir ce qu'il en est quant à son SGBD favori), un tuple d'une relation (ligne d'une table) est (physiquement) logé en totalité au sein d'un enregistrement sur le disque, c'est sommaire, mais c'est ainsi. En conséquence, l'accès à une valeur d'attribut d'un tuple provoque l'accès à l'ensemble des valeurs d'attributs de ce tuple. Suite à dénormalisation, la taille d'un enregistrement physique est supérieure à celle des enregistrements « normalisés », ce qui fait qu'il y aura moins d'enregistrements par page (bloc physique sur le disque). Ainsi, pour un traitement séquentiel par lots (batch) ou pour des transactions lourdes, le nombre de lectures/écritures sera accru et la durée du traitement en pâtira d'autant.
- Les opérations de mise à jour ne sont pas à l'abri : là encore, du fait du non respect de l'indépendance physique, la mémoire sera inutilement encombrée et les temps de traitement pénalisés, car le SGBD manipulera des enregistrements pondéralement surchargés.

Il faut descendre dans la soute et **prouver** le bien-fondé de cette dénormalisation, résultats de mesures sérieuses en main, suite à des séances de **prototypage de performance** poussées. Un exemple simple, celui qui est considéré au paragraphe 3.8 est caractéristique de situations où l'on se trompe de cible, et dans lesquelles dénormaliser revient à appliquer un cautère sur une jambe de bois sans améliorer la performance. Dénormaliser devient dangereux (anomalies potentielles de mise à jour, coût du stockage), comme par exemple dans le cas de l'hypothétique table des membres de DVP (cf. paragraphes 3.1.2 et 3.1.3). Une réflexion attentive portant sur cette table (cf. Figure 3.2) incite à penser que c'est bien plus la dénormalisation que la normalisation qui peut être source d'une inflation de lectures/écritures physiques sur disque et d'encombrement des caches.


En passant : existe-t-il des règles logiques nous permettant de dénormaliser de façon rationnelle ? En ce sens, je traduis Chris Date ([Date 2007b], chapitre 9, « Denormalization Considered Harmful » :

« Je voudrais mettre l'accent sur un point : une fois que l'on a décidé de dénormaliser, on s'est engagé sur une pente fort glissante. Question : Quand s'arrêter ? Avec la normalisation, la situation est différente car on a des raisons logiques et claires de poursuivre le processus jusqu'à ce qu'on ait atteint la forme normale la plus élevée possible. Doit-on en conclure qu'avec la dénormalisation on ait à procéder jusqu'à atteindre la forme normale la moins élevée qui soit ? Bien sûr que non, à ce jour nous ne disposons pas de critères logiques permettant de décider quand arrêter le processus. En d'autres termes, en choisissant de dénormaliser, on a décidé d'abandonner une position qui offre une base scientifique et une théorie logique solides, pour la remplacer par quelque chose de purement pragmatique et nécessairement subjectif. »

Au fond, la dénormalisation est un bel exemple d'*ignoratio elenchi*. Et comme le fait observer Frédéric Brouard (SQLpro) :

« On peut aussi pousser le bouchon à l'extrême : pourquoi pas une seule table contenant tout dans la base ? Vous commencez donc à douter de l'efficacité du tout au même endroit... Mais où il faut-il s'arrêter ? Ou placer le curseur ? C'est justement l'art du respect des formes normales qui nous en donne la clef ! »

Signalons quand même une situation dans laquelle on peut cette fois-ci se poser légitimement la question de la dénormalisation d'une relvar : il s'agit de la situation dans laquelle la BCNF ou la 4NF seraient violées, alors que la normalisation à tout prix n'arrangerait pas forcément les choses (cf. paragraphes 3.7 et 4.9). Mais il faut reconnaître que cette situation ne se présente fort heureusement pas souvent. Sinon, si la modélisation conceptuelle des données est réalisée selon les règles de l'art, on n'a guère de raison objective de ne pas normaliser :

 La dénormalisation est en général la conséquence d'une modélisation conceptuelle des données absente ou non maîtrisée.

Un dernier point. Si d'aucuns admettent que les bases de données utilisées dans un contexte transactionnel doivent être normalisées, ils n'ont aucun état d'âme à dénormaliser à tout va les tables de dimension dans le contexte des bases de données décisionnelles ([Dimensional Modeling](#)). Si ces tables sont uniquement reconstruites (par exemple chaque nuit) et ne font l'objet d'aucune mise à jour de type INSERT, UPDATE, DELETE entre deux chargements, pourquoi pas... En tout cas, il y aura du *null*, de la neige, de la redondance difficilement contrôlable malgré le soin extrême que l'on apportera aux opérations (*errare humanum est...*) Quoi qu'il en soit, les bases de données décisionnelles ne sont pas l'objet de cet article.

1.7. Dénormalisation vs amélioration (optimisation)

Le terme *dénormalisation* est bien souvent dévoyé. Chris Date met les points sur les i [Date 2007b] et mentionne quelques techniques d'amélioration (*optimisation* en français) présentées à tort comme relevant de la dénormalisation. (Dans ce qui suit, on parlera de tables plutôt que de relvars.)

1er exemple (comparable au 1er exemple du paragraphe 2.8).

Supposons que l'on ait à structurer une table des ventes journalières des magasins de l'entreprise Tartempion sur une période d'une semaine. Il est d'usage de structurer cette table, appelons-la Magasin_V, à l'aide des attributs suivants : MagId, Jour, ChiffreAffaires, la paire {MagId, Jour} étant clé. Pour améliorer (?) les performances (principe du « tout en une seule ligne »), certains préfèrent mettre en œuvre une table, appelons-la Magasin_H, de clé {MagId}, telle que le chiffre d'affaires fasse l'objet d'un attribut pour chaque jour de la semaine. En général, le terme employé pour ce changement de structure est celui de dénormalisation, et c'est à tort, car les deux tables respectent la BCNF (et même la 5NF), elles sont bien normalisées.



Figure 1.7 - Représentation verticale / horizontale

Maintenant, on peut faire observer que la structure de la table Magasin_H a le grave défaut de ne pas être évolutive : si les besoins de l'entreprise deviennent décennaires, quelles seront les conséquences ? Dans le cas la table Magasin_V, ceci sera transparent, par contre il faudra changer la structure de la table Magasin_H, opération lourde s'il en est (sans parler des requêtes existantes qui devront être modifiées pour prendre en compte les attributs supplémentaires).

Du point de vue de la manipulation des données, on observera que dans le cas de la table Magasin_H, l'utilisation des opérateurs classiques d'agrégation, SUM, AVG, etc. est pour le moins remise en question.

Du point de vue de la performance, dans les deux cas, une seule lecture d'un enregistrement physique suffira pour connaître le chiffre d'affaires hebdomadaire d'un magasin. Mais, dans le cas de la table Magasin_H, il serait déraisonnable d'indexer les sept colonnes si le besoin s'en faisait sentir, en effet la performance des mises à jour est (au moins) inversement proportionnelle au nombre d'index.

En fait, l'en-tête de la table Magasin_H ressemble plutôt à celui d'un résultat à présenter à l'utilisateur. On doit conserver la structure de la table Magasin_V et créer une **vue** VH afin de présenter les données selon la structure de la table Magasin_H, ça n'est quand même pas bien sorcier. Se reporter chez DVP à [l'exemple des clients et des options](#). Le mieux est encore d'en passer par un **instantané** (*snapshot*) rafraîchi à chaque mise à jour (la vue « matérialisée » (oxymore...) de SQL).

2e exemple.

Supposons que la table Magasin_V ci-dessus corresponde à un regroupement de tables par régions : Ces tables sont normalisées, elles respectent la 5NF. Pour connaître le chiffre d'affaires national, c'est l'opérateur **UNION** que l'on utilisera. On peut du reste créer une **vue** d'union ou un instantané pour définir la table virtuelle Magasin_V (puis la vue VH).

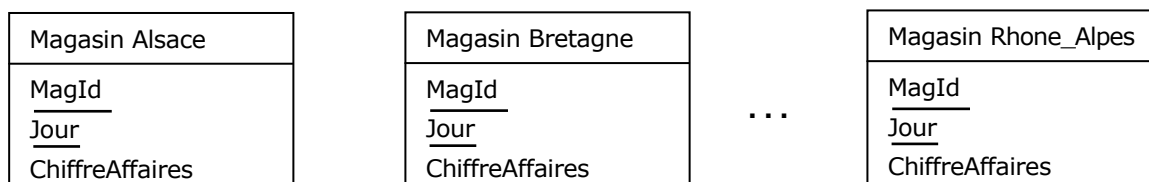


Figure 1.8 - Décomposition verticale

A noter que si le SGBD permet le partitionnement des tables, on pourra se contenter de n'avoir qu'une table des magasins, en affectant une partition à chacun d'eux (tout en définissant au besoin une vue par magasin).

3e exemple.

Si les magasins sont en relation avec les produits, leur chiffre d'affaires par produit peut être géré de façon redondante : le chiffre d'affaires total par produit, est égal à la somme des chiffres d'affaires par magasin et par produit, il y a donc **redondance** (avec tous les risques d'incohérence inhérents). Mais ces deux tables respectent encore la BCNF (et la 5NF).

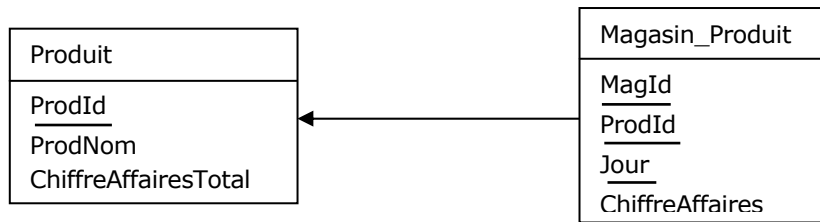


Figure 1.9 - Redondance inter-tables

N.B. Pour éviter de prendre des risques, on peut sous-traiter au SGBD le contrôle de la redondance, en attachant à la table Magasin_Produit un trigger chargé du calcul du chiffre d'affaires total des magasins concernés et qui mette ainsi à jour la table Produit en temps réel. Pour ne pas polluer cette table par les mises à jour, on peut préférer déplacer l'attribut ChiffreAffairesTotal dans un instantané, comme dans les exemples précédents.

4e exemple.

Un client passe commande. Une commande se décline en lignes de commande. Pour chaque ligne de commande, on s'engage en fonction des disponibilités des produits en stock, en cours de fabrication, des approvisionnements en cours, etc. ; un engagement est composé à son tour de parties livrables en fonction du nombre de camions nécessaires pour l'acheminement, etc.

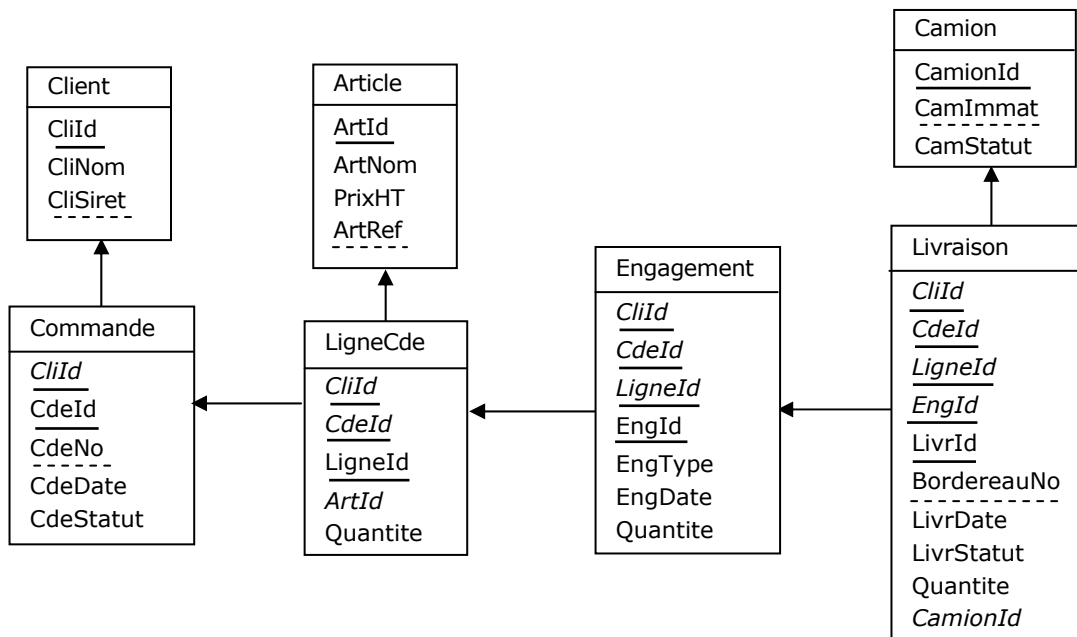


Figure 1.10 - Identification relative et redondance intra-clés

Par référence aux RVA (cf. paragraphe 2.6), la commande représente une propriété multivaluée du client, la ligne de commande représente à son tour une propriété multivaluée de la commande, l'engagement sur ligne de commande une propriété multivaluée de la ligne de commande, etc. Dans le diagramme ci-dessus, on a déplié les RVA et utilisé l'**identification relative** (cf. note qui suit), source de redondances au sein des clés. (N.B. Dans le diagramme, les clés primaires sont soulignées, les clés étrangères sont en italiques, sachant que les attributs appartenant à une clé primaire peuvent aussi appartenir à une clé étrangère ; les principales clés alternatives — celles qui sont connues de l'utilisateur — sont soulignées en traits discontinus).

Note concernant l'identification relative

Revenons sur l'exemple précédent. Utiliser l'identification relative revient à considérer (au niveau logique) que la clé d'une table — par exemple Commande — est composée des attributs composant la clé de la table dont elle est une propriété multivaluée (Commande est une propriété multivaluée de Client et conceptuellement parlant, il s'agit d'une entité-type *faible*), plus un attribut permettant de distinguer chaque commande d'un client donné. Selon l'usage, cet attribut supplémentaire est de type Entier et numérote chaque commande relativement à un client. Exemple (clés soulignées) :

Client { <u>CliId</u> , CliNom, CliSiret, ...}			
1	Dubicobit	12345678900001	
2	Frichmoutz	31415926500009	
...	

Commande { <u>CliId</u> , <u>CdeId</u> , CdeNo, CdeDate, ...}			
1	1	123456	15/02/2009
1	2	234567	01/04/2009
1	3	234575	02/04/2009
2	1	123023	20/01/2009
2	2	230239	17/03/2009
2	3	256789	06/01/2010
...

Du point de vue de l'utilisateur, le client Dubicobit a passé les commandes 123456, 234567 et 234575. Du point de vue du système, les commandes <1, 1>, <1, 2> et <1, 3> sont celles du client 1, tandis que les commandes <2, 1>, <2, 2> et <2, 3> sont celles du client 2.

Par contraste, utiliser l'**identification absolue** pour la table Commande (qui conceptuellement parlant n'est plus une entité-type faible) consiste à changer la composition de la clé, en remplaçant le couple {CliId, CdeId} par un singleton {CdeId} :

Commande { <u>CdeId</u> , CdeNo, CdeDate, ...}			
1	1	123456	15/02/2009
1	2	234567	01/04/2009
1	3	234575	02/04/2009
2	4	123023	20/01/2009
2	5	230239	17/03/2009
2	6	256789	06/01/2010
...

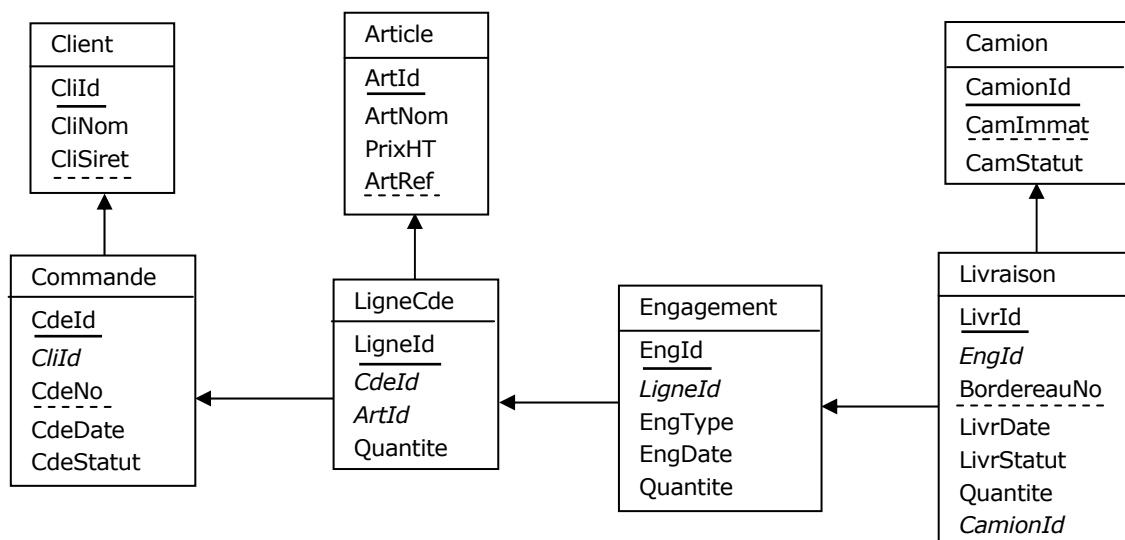


Figure 1.11 - Identification absolue systématique

L'intérêt de l'identification relative ne saute pas aux yeux, et l'on pourrait légitimement douter de sa pertinence, donc préférer ne pas la mettre en œuvre. Que l'identification soit absolue ou relative, les tables Commande, LigneCde, Engagement et Livraison respectent la cinquième forme normale, mais l'identification relative est cause de **redondance** au sein des clés, ce qui n'a pas lieu dans le cas de l'identification absolue. Toutefois, cette redondance est parfaitement contrôlée par le système grâce aux contraintes d'intégrité référentielle déclarées. Par ailleurs, d'aucuns objectent que cette redondance est nécessairement consommatrice de ressources (mémoire, disque...), tandis que l'utilisation de l'identification absolue entraînerait une consommation minimale. On peut montrer qu'il n'en est rien (cf. Annexe F.1) et que la durée de certains traitements — batchs lourds notamment — peut être réduite de façon très sensible, voire déterminante si l'on passe à l'identification relative (cf. Annexe F.2).

Conséquence de l'identification relative sur l'organisation des requêtes SQL

Supposons que l'on ait besoin de savoir quels camions sont concernés par les livraisons chez le client Gillou (Siret = 12345678900001). Si on utilise l'identification absolue, on devra coder une requête SQL faisant intervenir toutes les tables intermédiaires, à savoir Commande, LigneCde, Engagement et Livraison :

Requête 1 (identification absolue)

```
SELECT DISTINCT Camion.CamImmat
FROM   Client JOIN Commande
        ON Client.CliId = Commande.CliId
        JOIN LigneCde
        ON Commande.CdeId = LigneCde.CdeId
        JOIN Engagement
        ON LigneCde.LigneId = Engagement.LigneId
        JOIN Livraison
        ON Engagement.EngId = Livraison.EngId
        JOIN Camion
        ON Livraison.CamionId = Camion.CamionId
WHERE  CliSiret = '12345678900001' ;
```

(Incidentement, comme on s'intéresse en particulier au client Gillou, l'opération est performante, mais si l'on effectue des traitements de type batch, sans que les clés étrangères fassent l'objet d'index *clusters* (cf. Annexes F.2 et F.3), la dégradation des performances peut poser de très gros problèmes pour la Production informatique.)

Dans le cas de l'identification relative, on pourrait aussi écrire une requête analogue :

Requête 2 (identification relative)

```
SELECT DISTINCT Camion.CamImmat
FROM   Client JOIN Commande
        ON Client.CliId = Commande.CliId
        JOIN LigneCde
        ON Commande.CliId = LigneCde.CliId
        AND Commande.CdeId = LigneCde.CdeId
        JOIN Engagement
        ON LigneCde.CliId = Engagement.CliId
        AND LigneCde.CdeId = Engagement.CdeId
        AND LigneCde.LigneId = Engagement.LigneId
        JOIN Livraison
        ON Engagement.CliId = Livraison.CliId
        AND Engagement.CdeId = Livraison.CdeId
        AND Engagement.LigneId = Livraison.LigneId
        AND Engagement.EngId = Livraison.EngId
        JOIN Camion
        ON Livraison.CamionId = Camion.CamionId
WHERE  CliSiret = '12345678900001' ;
```

Mais on peut alléger la requête ainsi :

Requête 3 (identification relative, variante)


```
SELECT DISTINCT Camion.CamImmat
FROM   Client JOIN Commande
        ON   Client.CliId = Commande.CliId
        JOIN LigneCde
        ON   Commande.CliId = LigneCde.CliId
        JOIN Engagement
        ON   LigneCde.CliId = Engagement.CliId
        JOIN Livraison
        ON   Engagement.CliId = Livraison.CliId
        JOIN Camion
        ON   Livraison.CamionId = Camion.CamionId
WHERE  CliSiret = '12345678900001' ;
```

Et plus important, on améliorera plus que substantiellement la performance en prenant un raccourci façon couloir spatio-temporel, en codant encore plus simplement :

Requête 4 (identification relative, 2e variante)

```
SELECT DISTINCT Camion.CamImmat
FROM   Client JOIN Livraison
        ON   Client.CliId = Livraison.CliId
        JOIN Camion
        ON   Livraison.CamionId = Camion.CamionId
WHERE  CliSiret = '12345678900001' ;
```

Dans ce genre d'exercice et dans un contexte batch ou de requêtes lourdes, l'identification absolue ne pourra évidemment pas rivaliser en performances et fera que l'on aura l'impression de faire du surplace (cf. Annexe F.3)...

 Au passage, faisons observer que, grâce à la propagation de l'attribut CliId par identification relative, de lui-même l'optimiseur de DB2 for z/OS aménage les requêtes dans lesquelles les tables intermédiaires n'interviennent que comme courroies de transmission et les réécrit, les « optimise » pour produire la 4e requête.

5e exemple.

Certains auteurs merisiens sont certes des références reconnues dans leur partie, mais ils feraient mieux de garder le silence quand ils s'aventurent dans les terres relationnelles (Relationland), car ils sont manifestement mal équipés pour cela. Dans [RoMo 1989], au paragraphe 6.2.6 « Optimisation du MLD relationnel », page 200, est présenté un MCD partiel d'un système de facturation, comprenant les trois entités-types suivantes : CLIENT, FACTURE, REGLEMENT liées par des relations fonctionnelles, à savoir des CIF (contraintes d'intégrité fonctionnelle) :

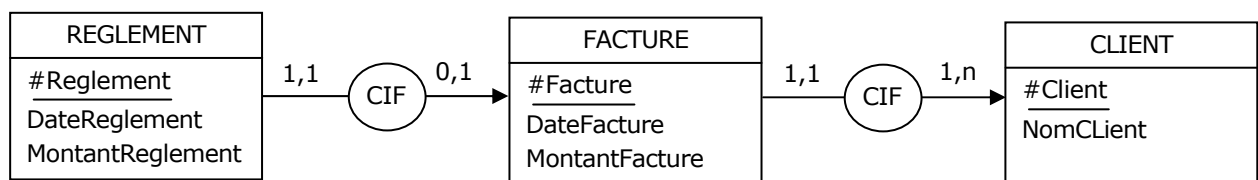


Figure 1.12 - CIF (REGLEMENT - FACTURE)

Lors du passage au MLD, est produit un ensemble de tables, dont l'équivalent reformulé en SQL peut être le suivant (on y remplace le symbole # par Id) :

```
CREATE TABLE CLIENT
(
    IdClient          Int          NOT NULL
    , NomClient       VarChar(32)  NOT NULL
    , CONSTRAINT CLIENT_PK PRIMARY KEY (IdClient)
) ;
CREATE TABLE FACTURE
(
    IdFacture         Int          NOT NULL
    , DateFacture     Date         NOT NULL
    , MontantFacture  Int          NOT NULL
    , IdClient        Int          NOT NULL
    , CONSTRAINT FACTURE_PK PRIMARY KEY (IdFacture)
    , CONSTRAINT FACTURE_FK FOREIGN KEY (IdClient) REFERENCES CLIENT (IdClient)
) ;
CREATE TABLE REGLEMENT
(
    IdReglement      Int          NOT NULL
    , DateReglement  Date         NOT NULL
    , MontantReglement Int        NOT NULL
    , IdFacture       Int          NOT NULL
    , CONSTRAINT REGLEMENT_PK PRIMARY KEY (IdReglement)
    , CONSTRAINT REGLEMENT_FK FOREIGN KEY (IdFacture) REFERENCES FACTURE (IdFacture)
) ;
```

Jusque là tout va bien. Maintenant je cite (en notant que l'auteur remplace « CIF » par « DF », mais peu importe) :

« [...] La DF de REGLEMENT vers FACTURE a pour cardinalités maximales 1 dans chaque sens. Il est indispensable de procéder à une optimisation [...]. L'optimisation présente se traduira par l'introduction dans FACTURE de la clé étrangère #Reglement, afin de faciliter la communication entre FACTURE et REGLEMENT. »

Selon l'auteur, la structure de FACTURE devrait donc **impérativement** être « optimisée » ainsi :

```
CREATE TABLE FACTURE
(
    IdFacture         Int          NOT NULL
    , DateFacture     Date         NOT NULL
    , MontantFacture  Int          NOT NULL
    , IdClient        Int          NOT NULL
    , IdReglement     Int          NOT NULL
    , CONSTRAINT FACTURE_PK PRIMARY KEY (IdFacture)
    , CONSTRAINT FACTURE_FK1 FOREIGN KEY (IdClient)
        REFERENCES CLIENT (IdClient)
    , CONSTRAINT FACTURE_FK2 FOREIGN KEY (IdReglement)
        REFERENCES REGLEMENT (IdReglement)
) ;
```

Quelle horreur ! Je ne sais pas trop ce que l'auteur entend par « faciliter la communication » (*sic* !), en tout cas on se retrouve maintenant avec un cycle entre FACTURE et REGLEMENT, outre que l'attribut IdReglement de FACTURE doit être marqué NULL lors de la création de chaque facture, puis mis à jour avec la valeur qui va bien à chaque règlement effectif. Cette proposition calamiteuse d'« optimisation » est évidemment bonne pour la poubelle. Ce qu'il faut faire :

1. Ne pas toucher à la structure de la table FACTURE, mais définir une clé alternative pour la table REGLEMENT :

```
CREATE TABLE REGLEMENT
(
    IdReglement      Int          NOT NULL
    , DateReglement  Date         NOT NULL
    , MontantReglement Int        NOT NULL
    , IdFacture       Int          NOT NULL
    , CONSTRAINT REGLEMENT_PK PRIMARY KEY (IdReglement)
    , CONSTRAINT REGLEMENT_AK UNIQUE (IdFacture)
    , CONSTRAINT REGLEMENT_FK FOREIGN KEY (IdFacture) REFERENCES FACTURE (IdFacture)
) ;
```

N.B. Si *nihil obstat*, au niveau conceptuel identifier REGLEMENT relativement à FACTURE, de telle sorte qu'au niveau logique l'attribut IdReglement disparaisse de REGLEMENT et que {IdFacture} y soit à la fois clé primaire et étrangère.

2. Définir au besoin une vue, appelons-la FACT_RGLT, pour effectivement « faciliter la communication », c'est-à-dire tout savoir sur les factures réglées. Dans le style SQL, cette vue (parmi d'autres, selon les besoin) pourrait être :

```
CREATE VIEW FACT_RGLT
    (IdFacture, DateFacture, MontantFacture, IdClient,
     IdReglement, DateReglement, MontantReglement)
AS
    SELECT      x.IdFacture, x.DateFacture, x.MontantFacture, x.IdClient,
                y.IdReglement, y.DateReglement, y.MontantReglement
    FROM        FACTURE AS x JOIN REGLEMENT AS y
                ON x.IdFacture = y.IdFacture ;
```

Pour des raisons de symétrie, définir aussi une vue pour tout savoir sur les factures non réglées :

```
CREATE VIEW FACT_NON_RGLT
    (IdFacture, DateFacture, MontantFacture, IdClient)
AS
    SELECT      x.IdFacture, x.DateFacture, x.MontantFacture, x.IdClient
    FROM        FACTURE AS x
    WHERE       NOT EXISTS
                (SELECT  ' '
                 FROM    REGLEMENT AS y
                 WHERE   x.IdFacture = y.IdFacture) ;
```

Et tant qu'à faire, pour disposer d'un jeu complet, définir une vue supplémentaire qui permette de présenter toutes les factures, qu'elles soient notées réglées ou non réglées :

```
CREATE VIEW FACT_STATUT
    (IdFacture, DateFacture, MontantFacture, IdClient,
     IdReglement, DateReglement, MontantReglement, Statut)
AS
    SELECT      IdFacture, DateFacture, MontantFacture, IdClient,
                IdReglement, DateReglement, MontantReglement, 'Régulé'
    FROM        FACT_RGLT
    UNION
    SELECT      IdFacture, DateFacture, MontantFacture, IdClient,
                ' ', ' ', ' ', 'Non Régulé'
    FROM        FACT_NON_RGLT ;
```

Les [tenants](#) de la jointure externe de SQL (opérateur exclu de la théorie relationnelle) préféreront sans doute directement coder ainsi la vue FACT_STATUT :

```
CREATE VIEW FACT_STATUT
    (IdFacture, DateFacture, MontantFacture, IdClient,
     IdReglement, DateReglement, MontantReglement, Statut)
AS
    SELECT      x.IdFacture, x.DateFacture, x.MontantFacture, x.IdClient
                , COALESCE (CAST (y.IdReglement AS Varchar(10)), 'Néant') AS IdReglement
                , COALESCE (CAST (y.DateReglement AS Varchar(10)), 'Néant') AS DateReglement
                , COALESCE (CAST (y.MontantReglement AS Varchar(10)), 'Néant') AS MontantReglement
                , CASE WHEN y.IdReglement IS NULL THEN 'Non réglé' ELSE 'Régulé' END AS Statut
    FROM        FACTURE AS x LEFT JOIN REGLEMENT AS y
                ON x.IdFacture = y.IdFacture ;
```

Mais revenons à l'étude de la normalisation, qui est quand même le sujet de cet article.

2. Première forme normale

Beaucoup de choses vraies ou fausses ont été dites au sujet de la première forme normale (1NF), aussi un retour aux sources ne sera-t-il pas de trop. Rappelons une fois de plus que c'est Ted Codd qui a inventé et théorisé le concept de normalisation pour les bases de données, et il serait malvenu de chahuter sans raison profonde les définitions qu'il en a données. Les théoriciens du relationnel sont restés en phase avec ce qu'a écrit Codd, tout en fournissant à l'occasion quelques précisions. Mais le Modèle Relationnel de Données n'est pas figé pour l'éternité, il a connu et connaîtra encore bien des évolutions. Ainsi, sans se départir de l'esprit insufflé par son inventeur, Date et Darwen ont-ils été amenés une vingtaine d'années plus tard à approfondir la 1NF, et ce avec une extrême rigueur.

2.1. La situation en 1969

Dans son tout premier article concernant le Modèle Relationnel de Données (cf. [Codd 1969] page 2), Codd fournissait un exemple (ici francisé) de représentation plate d'une relation de degré 4, décrivant les livraisons de pièces par des fournisseurs, à l'usage de projets, selon certaines quantités :

Livraison (Fournisseur	Pièce	Projet	Quantité)
1	2	5	17
1	3	5	23
2	3	7	9
2	7	5	4
4	1	1	12

Figure 2.1 - Les livraisons, représentation traditionnelle

Cette représentation tabulaire, tout à fait classique et « sage » n'appelle pas de commentaires particuliers.

Dans ce même article, Ted Codd avait écrit (pages 3-4) :

« Nous avons traité de relations définies sur des domaines **simples** — domaines dont les éléments sont des valeurs **atomiques** (non décomposables). Les valeurs non atomiques peuvent être prises en considération dans le cadre relationnel. Ainsi, certains domaines peuvent avoir des relations comme éléments. Ces relations peuvent à leur tour être définies sur des domaines non simples, et ainsi de suite.

[...] L'adoption d'une perception relationnelle des données, autorise le développement d'un sous-langage universel de recherche, basé sur le **calcul des prédicats du deuxième ordre**.

[...] Le calcul des prédicats du deuxième ordre est nécessaire (plutôt que celui du premier ordre) parce que les domaines sur lesquels les relations sont définies peuvent à leur tour contenir des éléments qui sont des relations. »

Et reprenant l'exemple précédent, Codd propose une représentation sous forme de relations emboîtées :

P (Fournisseur, Q (Pièce, R (Projet, Quantité)))
--

Figure 2.2 - Les livraisons, avec emboîtement de relations

Cette fois-ci, la relation P est binaire (degré 2), ses domaines sont Fournisseur (simple) et Q (non simple). La relation emboîtée Q est à son tour binaire et ses domaines sont Pièce (simple) et R (non simple). Les domaines de la relation emboîtée R sont tous simples.

2.2. 1970 : Acte de naissance de la première forme normale

En 1970, dans son article de loin le plus cité et considéré comme celui dans lequel sont posées les fondations du Modèle Relationnel de Données [Codd 1970], Codd encourage à l'utilisation de domaines simples (page 381) ce qui permet d'éliminer le problème (s'il existe) du calcul des prédicats du deuxième ordre adopté l'année précédente :

« L'utilisation d'un modèle relationnel de données [...] permet de développer un sous-langage universel basé sur le calcul des prédicats. Si la collection des relations est en **forme normale**, alors un **calcul des prédicats du premier ordre** est suffisant. »

« Forme normale » étant à interpréter aujourd'hui comme « Première forme normale » (1NF). Le concepteur de bases de données relationnelles — qu'il en soit conscient ou non — applique la 1NF à la lettre parce que les domaines des attributs de ses tables sont simples. Même chose pour celui qui conçoit des MCD au sens Merise et qui applique la règle dite de vérification, selon laquelle : « Dans chaque occurrence d'individu-type ou de relation-type on ne trouve qu'une seule valeur de chaque propriété » [TRC 1989].

A noter que divers chercheurs et auteurs (par exemple [Jaeschke 1982], [Abiteboul 1984], [Korth 1988]) ont fait des propositions pour « étendre » le Modèle Relationnel et réintroduire la possibilité d'emboîter les relations les unes dans les autres. Les systèmes dans lesquels on s'affranchit de la première forme normale sont dits « Non première forme normale » (*Non First Normal Form*, NF² en abrégé). Il s'agit d'un sujet qui sort de notre périmètre et que nous ne traiterons pas, d'autant plus que Date et Darwen ont montré que l'emboîtement des relations (en tant que valeurs) était possible sans avoir à étendre le Modèle Relationnel (cf. paragraphe 2.6).

1971 : Définition de la première forme normale

Dans son article de 1970, Codd décrit la 1re forme normale et c'est en 1971 qu'il en donne la définition (cf. [Codd 1971], page 31) :

« Une relation est en première forme normale si aucun de ses domaines ne peut contenir des éléments qui soient eux-mêmes des ensembles. Une relation qui n'est pas en première forme normale est dite non normalisée. »

Ainsi, une relation (ceci vaut pour une table SQL) est en première forme normale (1NF) si aucun de ses attributs ne peut prendre de valeur qui soit elle-même un ensemble, en l'occurrence une relation.

Selon les critères de Codd, si un SGBD acceptait l'instruction suivante, alors il autoriserait pertinemment le viol de la 1NF. En effet, l'attribut Messages est propre à contenir des relations (disons des tables dans un contexte SQL) :

```
CREATE TABLE Membre
(
    MbrId          INTEGER          NOT NULL
    , Pseudonyme   VARCHAR(16)     NOT NULL
    , DateInscription DATE          NOT NULL
    , Localisation VARCHAR(48)      NOT NULL
    , AdrCourriel  VARCHAR(48)      NOT NULL
    , Messages (
        MessageId  INTEGER          NOT NULL
        , MessageDate  TIMESTAMP      NOT NULL
        , MessageTexte VARCHAR(MAX)   NOT NULL
        , Forum      CHAR(8)          NOT NULL
    )
    , CONSTRAINT MbrPk PRIMARY KEY (MbrId)
    , CONSTRAINT CK2 UNIQUE (Pseudonyme)
    , CONSTRAINT CK3 UNIQUE (AdrCourriel)) ;
```

Figure 2.3 - Viol de la 1NF

Pour rester en accord avec Codd, la structure précédente peut être transformée exactement comme il le montre dans son article de 1970, en procédant par scissiparité, ce qui donne lieu à deux structures, Membre et Message :

```
CREATE TABLE Membre
(
    MbrId          INTEGER          NOT NULL
    , Pseudonyme   VARCHAR(16)     NOT NULL
    , DateInscription DATE          NOT NULL
    , Localisation  VARCHAR(48)     NOT NULL
    , AdrCourriel   VARCHAR(48)     NOT NULL
    , CONSTRAINT MbrPk PRIMARY KEY (MbrId)
    , CONSTRAINT CK2 UNIQUE (Pseudonyme)
    , CONSTRAINT CK3 UNIQUE (AdrCourriel)) ;

CREATE TABLE Message
(
    MbrId          INTEGER          NOT NULL
    , MessageId     INTEGER          NOT NULL
    , MessageDate   TIMESTAMP        NOT NULL
    , MessageTexte  VARCHAR(MAX)    NOT NULL
    , Forum         CHAR(8)          NOT NULL
    , CONSTRAINT MessPk PRIMARY KEY (MbrId, MessageId)
    , CONSTRAINT MessFk FOREIGN KEY (MbrId)
        References Membre (MbrId) On Delete Cascade) ;
```

Figure 2.4 - Respect de la 1NF

(La contrainte MessFk n'est présente que pour des raisons de cohérence entre les deux tables).

2.3. Années 1970-1980. Les théoriciens du Modèle Relationnel sont-ils en phase avec Codd ?

On a brièvement noté que des auteurs comme Roth, Korth et Silberschatz proposèrent une extension du Modèle Relationnel afin de réintroduire le principe des relations emboîtées les unes dans les autres (relations NF²).

Mais de façon générale, les théoriciens du relationnel ont fourni dans leurs écrits des définitions de la 1NF conformes à celle de Codd, y compris du reste ceux qui viennent d'être cités. Par exemple :


- « Un schéma de relation R est en première forme normale (1NF) si les domaines de l'ensemble des attributs de R sont atomiques. Un domaine est atomique si ses éléments sont indivisibles. » [Korth 1986]
- « Une relation R est en première forme normale (1NF) si et seulement si tous les domaines sous-jacents contiennent uniquement des valeurs atomiques. » [Date 1986]. Plus tard, Date modifiera sa définition, cf. paragraphe 2.7.
- « Une relation est en première forme normale si tout attribut contient une valeur atomique. » [Gardarin 1988]
- « Une relation est dite “normalisée” ou en “première forme normale” (1NF) si aucun attribut qui la compose n'est lui-même une relation, c'est-à-dire, si tout attribut est atomique (non décomposable). » [Miranda 1986]

Codd lui-même reprit le concept omniprésent d'atomicité et écrivit en 1990, dans son ultime ouvrage de référence traitant du Modèle Relationnel de Données [Codd 1990] :

« Les valeurs des domaines sur lesquels chaque relation est définie doivent être **atomiques** [...] »

D'autres considérèrent, à juste titre, que les relations sont *de facto* normalisées 1NF parce qu'ils ne traitent par définition que de domaines dont les valeurs ne sont pas des ensembles (Delobel et Adiba, Ullman, ...)

Par exemple, Jeff Ullman se contente d'écrire (cf. [Ullman 1982], « Normal forms for relation schemes ») :

 « Nous n'utilisons pas de domaines dont les valeurs sont des ensembles, et nous pouvons donc nous dispenser de mentionner la première forme normale. Nous considérons en effet que “relation” est synonyme de “relation en première forme normale.” »

2.4. L'esprit et la lettre

La définition de la 1NF par Ted Codd est précise. Mais on se situe parfois à la limite quant à son interprétation par les utilisateurs. Par exemple, le catalogue de DB2 for z/OS comporte une table appelée SYSCOPY, utilisée pour les opérations de *recovery*, comportant une colonne DSVOLSER de type VARCHAR(1784), utilisée pour stocker la liste des numéros à 6 chiffres (séparés par des virgules) des disques hébergeant les données d'une table. L'instruction SQL suivante est conforme du point de vue de la normalisation :

```
INSERT INTO SYSCOPY (... , DSVOLSER, ...) VALUES (... , "000701,010002,...,314116", ...) ;
```

En effet, au sens de Codd, la chaîne de caractères "000701,010002,...,314116" est **atomique**, car c'est une valeur tirée du domaine VARCHAR(1784) lequel, à la lettre, est bien atomique, en vertu de quoi la table SYSCOPY respecte la 1NF. Évidemment, si on veut savoir quelles tables (en fait quels *table spaces*) sont hébergées par le disque "314116", il suffit en SQL de coder par exemple :


```
SELECT DISTINCT TSNAME
FROM   SYSCOPY
WHERE  DSVOLSER LIKE ("%314116%") ;
```

Et pourtant, on pourrait discuter sans fin du bien-fondé de l'organisation de cette liste de numéros. En tout cas, on peut examiner cet exemple à la lumière de ce qu'a écrit Serge Miranda [Miranda 1988] :

« Une relation est dite normalisée (ou en “première forme normale” notée 1NF) si chaque valeur d'attribut est atomique (c'est-à-dire n'est pas un ensemble ou une liste) ; tout attribut d'une relation 1NF doit donc être monovalué. »

Ou du jugement sans appel de Chris Date ([Date 2007a], page 129) :

« Une colonne (attribut) C constitue un “groupe répétitif” si étant définie sur le domaine D, les valeurs légales pouvant apparaître dans C sont des ensembles (ou des listes, des tableaux, ou ...) de valeurs du domaine D. Ces groupes répétitifs sont définitivement proscrits du Modèle Relationnel. »

 Ne nous méprenons pas, un attribut peut faire référence à un domaine de chaînes de caractères, d'intervalles, de polygones, de tableaux, de relations, de ce que l'on veut. Ce qui est proscrit ici est bien le caractère **répétitif** des valeurs au sein d'un attribut faisant référence à un domaine (type) donné.

En tout état de cause, la table SYSCOPY ne contrevient pas à la 1NF. Une façon d'éviter toute équivoque consisterait à le faire sciemment, comme le permet PostgreSQL, grâce aux tableaux de valeurs. Au lieu d'écrire :

```
CREATE TABLE SYSCOPY
(DBNAME      CHAR(8)          NOT NULL,
TSNAME       CHAR(8)          NOT NULL,
...
DSVOLSER     VARCHAR(1784)    NOT NULL,
...) ;
```

Avec PostgreSQL il suffirait d'écrire, en appelant un chat un chat :

```
CREATE TABLE SYSCOPY
(DBNAME      CHAR(8)          NOT NULL,
TSNAME       CHAR(8)          NOT NULL,
...
DSVOLSER     TEXT [1]         NOT NULL,
...) ;
```

Puis :

```
INSERT INTO SYSCOPY (DBNAME, TSNAME, DSVOLSER, ...)
VALUES ("fsmrelb1", "fsmrelt1", ..., ARRAY ["000701", "010002", ..., "314116"], ...);
```

Il est quand même écrit dans la documentation de PostgreSQL :

« Chercher des éléments particuliers dans un tableau peut être le signe d'une **mauvaise conception** de la base de données. Étudiez la mise en place d'une table séparée dont chaque ligne soit affectée à chaque élément du tableau. La recherche sera plus simple et cela sera plus adapté si ces éléments sont nombreux. »

Qui plus est, l'affectation d'une **séquence** à des éléments contrevient à l'esprit du Modèle Relationnel.

2.5. L'atomicité : un critère absolu ?

Revenons sur les définitions données ci-dessus de la 1NF et dans lesquelles l'accent est donc mis sur l'atomicité. Comme l'a écrit Codd, *Atomique* signifie non décomposable, **pour autant que le système est concerné** : si une donnée réputée atomique cache en réalité une structure complexe, le système n'en a pas connaissance. Il en va ainsi de la colonne DSVOLSER de la table SYSCOPY, dans laquelle se cache une liste comme nous l'avons observé précédemment.

Examinons maintenant le contenu de la table Membre de la Figure 2.4, réputée en 1NF :

MbrId	Pseudonyme	DateInscription	Localisation	AdrCourriel
1	Mezig	01/04/2003	Relationland	mez@tutor-d.fr
2	Cezig	02/04/2003	Alpha Centauri	cez@alphacent.zz
3	Tezig	02/04/2003	Tahiti	tez@machin.pf
...

Figure 2.5 - Valeurs atomiques

A l'intersection de chaque ligne et de chaque colonne, on a exactement une valeur et celle-ci est **atomique**, non décomposable en l'état par le SGBD. Ainsi, la date d'inscription d'un membre est définie comme prenant ses valeurs dans le domaine DATE : ce domaine fait partie des domaines de base proposés par le système, et si ce dernier n'a pas *a priori* la connaissance sémantique de ce que à quoi on a accès au niveau subatomique, à savoir le jour, le mois, l'année, il met néanmoins à notre disposition des fonctions *ad hoc* correspondantes, telles DAY, MONTH, YEAR. De même, si l'on définissait une colonne CodePostal de type CHAR(5) pour les codes postaux des communes, le système n'aurait pas la connaissance sémantique de la structure interne de ces codes, à savoir par exemple que les deux premiers caractères représentent le numéro du département du bureau distributeur du courrier de la commune (les trois premiers caractères en France d'outre-mer). Cependant, le système nous permet, là encore, d'accéder au niveau subatomique, au moyen de la fonction SUBSTRING ou du prédicat LIKE.

Observons encore que la notion d'atomicité est tout à fait relative et élastique, même si « atomique » veut dire : non décomposable, indivisible. Considérons par exemple la colonne MbrId, qui est du type INTEGER. Quid si MbrId vaut 42 ? La valeur 42 est-elle atomique ? On pourrait pinailler et répondre négativement, car 42 est décomposable en facteurs premiers : $42 = 1 \times 2 \times 3 \times 7$. De la même façon, la chaîne de caractères "Developpez.com" peut être considérée comme étant composée de la séquence de caractères : "D,e,v,e,l,o,p,p,e,z,,c,o,m". Pour ne pas s'embarquer dans la controverse on peut préférer utiliser un autre adjectif, en remplaçant « atomique » par « **scalaire** » (voire « **encapsulé** », mais ce terme est connoté Orienté Objet) signifiant que le type (domaine), la colonne, la valeur, la variable, n'ont pas de composants directement visibles, mais accessibles, grâce à des opérateurs *ad hoc*, fournis avec le type, point barre. Par contraste, l'en-tête d'une relation n'est pas scalaire, puisque ses composants (ses attributs) sont visibles par l'utilisateur : la relation et le tuple (n-uplet) sont à ce jour **les seuls objets** qui, selon la théorie relationnelle, font légalement l'objet de types **non scalaires** (d'où un aménagement de la définition de la 1NF, cf. paragraphe 2.7). En tout état de cause, les types ARRAY et MULTISSET de SQL n'en font pas partie, mais pourraient très bien être pris en compte comme cela est précisé dans les remarques concernant Tutorial D (cf. le paragraphe A en annexe).

2.6. Début des années quatre-vingt-dix. Les RVA (attributs dont les valeurs sont des relations)

Codd mit sa casquette de logicien quand il écrivit en 1970 qu'un calcul du premier ordre suffit pour manipuler toutes les relations possibles, dans tous les sens et jusqu'à plus soif (autrement dit en vertu des principes de **complétude** et de **fermeture**), alors que l'année précédente il parlait sur la base d'un calcul du deuxième ordre (cf. le paragraphe C en annexe). L'*homo relationalis* aurait aimé en savoir plus sur ce revirement basé tout d'abord sur la simplicité de la représentation des données. Malheureusement, le père du Modèle Relationnel nous a quittés et l'on ne peut que conjecturer quand on lit : « *the possibility of eliminating nonsimple domains appears worth investigating* ».

On a vu par ailleurs des gens comme Roth, Korth et Silberschatz se comporter en « hérétiques » et proposer d'étendre le Modèle Relationnel grâce notamment à de nouveaux opérateurs (en particulier NEST/UNNEST), permettant d'atomiser récursivement des relations NF² (Non First Normal Form), relations dont les valeurs peuvent être des relations, à une profondeur quelconque. De leur côté, les gardiens du phare, Date et Darwen, n'ont pas manqué eux aussi d'étudier la possibilité de manipuler des relations contenant d'autres relations, mais tout en respectant la 1NF (aménagée).

Au début des années quatre-vingt-dix, dans un article intitulé « *Relation-Valued Attributes or Will the Real First Normal Form Please Stand Up?* » [Date 1992], D & D se sont livrés à une réflexion approfondie sur la véritable nature de la première forme normale :

« Nous examinons la véritable nature de la première forme normale, et avançons qu'il y a un certain nombre d'avantages à autoriser les attributs des relations à contenir des valeurs qui soient à leur tour (qui encapsulent) des relations. »

Leur constat fut — pour faire court — qu'à condition de laisser tomber la récursivité (cf. [Date 2007a], pages 126-127) et d'en rester aux opérateurs déjà existants, on pouvait utiliser avec profit des **RVA** (*relation-valued attributes*), sans avoir à étendre le Modèle Relationnel (évolution, oui, mais révolution, non), tout en restant dans le cadre de la première forme normale, dont la définition évolue malgré tout (cf. paragraphe 2.7), comme on l'a déjà brièvement évoqué. Par scrupule, Date a consulté les logiciens, mais personne n'a pu lui montrer que l'utilisation des RVA nécessitait d'en passer par une logique du deuxième ordre (cf. [Date 2007b], page 394). En passant, insistons sur le fait que les RVA ne contiennent que des **valeurs** (relations), jamais de **variables** (relvars), ces choses que le deuxième ordre permet pour sa part de quantifier.

Pour traiter des RVA, rappelons que, outre les opérateurs de base bien connus : RESTRICT, PROJECT, JOIN etc., D & D ont utilisé deux opérateurs faisant partie depuis une vingtaine d'années du Modèle Relationnel (et provenant du langage relationnel ISBL, lequel date des années soixante-dix) : **RENAME** et **EXTEND**. On trouvera dans un article rédigé par Darwen en 2005 (« [HAVING A Blunderful Time](#) ») l'importance du rôle joué par ces deux opérateurs, que nous allons être amenés à utiliser (concernant leur notation, se reporter au paragraphe B en annexe).

Exemple

Considérons les trois variables relationnelles sans prétention, qui constituent les piliers de la base de données relationnelle « Fournisseurs et Pièces » chère à Chris Date (base de données que nous francisons ici). La variable F est utilisée pour décrire les fournisseurs, la variable P concerne les pièces et la variable FP des liens qui unissent fournisseurs et pièces (« Quels fournisseurs ont livré quelles pièces », « Quelles pièces ont été livrées par quels fournisseurs »).

F	<u>Four_No</u>	Four_Nom	Statut	Ville	FP	<u>Four_No</u>	<u>Piece_No</u>	Quantite
	S1	Salsa	20	Lille		S1	P1	300
	S2	Jean	10	Paris		S1	P2	200
	S3	Bernard	30	Paris		S1	P3	400
	S4	Catherine	20	Lille		S1	P4	200
	S5	Alain	30	Arles		S1	P5	100
P	<u>Piece_No</u>	Piece_Nom	Couleur	Poids		S1	P6	100
	P1	Ecrou	Rouge	12,0		S2	P1	300
	P2	Boulon	Vert	17,0		S2	P2	400
	P3	Vis	Bleu	17,0		S3	P2	200
	P4	Vis	Rouge	14,0		S4	P2	200
	P5	Clou	Bleu	12,0		S4	P4	300
	P6	Rouage	Rouge	19,0		S4	P5	400

Figure 2.6 - Base de données des fournisseurs, des pièces et des livraisons

On trouve en annexe (paragraphe B), l'exemple le plus simple qui soit de l'utilisation de l'opérateur EXTEND pour produire à partir d'une relation R une relation R' dont l'en-tête est celui de R, augmenté d'un attribut.

Mais on peut aller beaucoup plus loin et produire une relation dont un attribut prend des valeurs qui sont des relations. Considérons en ce sens l'expression suivante, écrite en Tutorial D, utilisée pour présenter chaque fournisseur avec un inventaire des pièces qu'il fournit, le résultat étant illustré à l'aide de la Figure 2.7 :

WITH

```
(F RENAME (Four_No AS Four_No_Bis) AS T1),
(EXTEND T1 ADD (FP WHERE Four_No = Four_No_Bis) {Piece_No, Quantite} AS Piece_Qte) AS T2 :
T2 RENAME (Four_No_Bis AS Four_No) AS F_EXT
```

La construction

WITH <liste d'opérations> : <expression relationnelle>

permet de fournir sous forme d'une liste d'éléments séparés par des virgules, les opérations que l'on souhaite exécuter en séquence, pour produire au final la relation définie par <expression relationnelle>. La finalité de la liste étant de simplifier l'organisation des opérations qui, à défaut, pourraient devenir compliquées à exprimer.

En l'occurrence, on produit d'abord une relation T1 qui ne diffère de la relation F que par le nom de l'attribut Four_No_Bis. On procède ensuite à l'extension de T1 en récupérant dans FP l'ensemble des tuples pour lesquels la valeur de l'attribut Four_No est égale à celle de l'attribut Four_No_Bis de la relation T1, puis en effectuant une projection sur les attributs Piece_No et Quantite et en nommant Piece_Qte l'attribut (du type RELATION) obtenu par extension. La relation obtenue est nommée T2. Le résultat final F_EXT est celui de T2, en renommant pour finir Four_No_Bis en Four_No.

La représentation graphique de la relation F_EXT est la suivante :

F_EXT	<u>Four_No</u>	Four_Nom	Statut	Ville	Piece_Qte	
	S1	Salsa	20	Lille	<u>Piece_No</u>	Quantite
					P1	300
					P2	200
					P3	400
					P4	200
					P5	100
					P6	100
	S2	Jean	10	Paris	<u>Piece_No</u>	Quantite
					P1	300
					P2	400
	S3	Bernard	30	Paris	<u>Piece_No</u>	Quantite
					P2	200
	S4	Catherine	20	Lille	<u>Piece_No</u>	Quantite
					P2	200
					P4	300
					P5	400
	S5	Alain	30	Arles	<u>Piece_No</u>	Quantite

Figure 2.7 - Relation avec attribut à valeur relation (RVA)

Dans la relation F_EXT, les attributs Piece_No et Quantite ont été remplacés par le seul attribut Piece_Qte. Les autres attributs n'ont pas changé. Chaque valeur prise par l'attribut Piece_Qte est une relation (donc en-tête compris), une relation par attribut et par tuple de F_EXT : chaque tuple de F_EXT contient exactement une valeur pour chacun de ses attributs et cette relation est en première forme normale (cf. paragraphe 2.7). Noter que, pour le fournisseur S5, le corps de la relation {Piece_No, Quantite} est l'ensemble vide {}.

En Tutorial D, on pourrait du reste définir une relvar F_EXT :

```
VAR F_EXT BASE RELATION
{
  Four_No INTEGER,
  Four_Nom CHAR,
  Statut INTEGER,
  Ville CHAR,
  Piece_Qte RELATION {Piece_No INTEGER, Quantite INTEGER}
} KEY {Four_No} ;
```

Intérêt des RVA

Une relation comme F_EXT présente des avantages indéniables :

- Dans la Figure 2.7, le fait que le fournisseur S5 (Alain) n'a pas livré de pièces est représenté par l'ensemble vide au sein de l'attribut Piece_Qte de la relation F_EXT, alors que — en se plaçant dans un contexte SQL — si l'on utilisait une jointure externe (LEFT OUTER JOIN), le résultat ci-dessous serait pollué par le bonhomme NULL et ne pourrait même pas être doté d'une clé, en l'occurrence la paire {Four_No, Piece_No} : F_EXT ne pourrait donc être une relation et le principe de fermeture ne serait pas respecté (*horresco referens* !) :

Four_No	Four_Nom	Statut	Ville	Piece_No	Quantite
S1	Salsa	20	Lille	P1	300
S1	Salsa	20	Lille	P2	200
S1	Salsa	20	Lille	P3	400
S1	Salsa	20	Lille	P4	200
S1	Salsa	20	Lille	P5	100
S1	Salsa	20	Lille	P6	100
S2	Jean	10	Paris	P1	300
S2	Jean	10	Lille	P2	400
S3	Bernard	30	Paris	P2	200
S4	Catherine	20	Lille	P2	200
S4	Catherine	20	Lille	P4	300
S4	Catherine	20	Lille	P5	400
S5	Alain	30	Arles	NULL	NULL

Figure 2.8 - LEFT OUTER JOIN de F et FP

- En comparant la Figure 2.7 et la Figure 2.8, on se rend compte que dans le premier cas, la redondance est absente, alors qu'elle foisonne dans le second cas.
- La Figure 2.7 correspond à une relation peut-être plus proche de la vision qu'ont certains utilisateurs (ceux pour lesquels, sémantiquement parlant, la relation FP de la Figure 2.6 est une propriété multivaluée de la relation F, plutôt que de la relation P).

Dans leur article mentionné ci-dessus « *Relation-Valued Attributes or Will the Real First Normal Form Please Stand Up?* », D & D fournissent nombre d'observations supplémentaires portant sur l'intérêt des RVA.

Opérateurs GROUP et UNGROUP

Notons en passant que D & D ont prévu deux opérateurs non indispensables (en effet, ce sont des combinaisons d'autres opérateurs), mais permettant d'élever d'un cran le niveau d'abstraction (donc de nous simplifier la vie, ainsi que celle de l'optimiseur) pour effectuer des requêtes dans lesquelles interviennent des RVA, à savoir GROUP et UNGROUP. Ainsi, pour obtenir l'équivalent RVA de la relvar FP de la Figure 2.6 :

```
FP GROUP {Piece_No, Quantite} AS Piece_Qte
```

La relation obtenue est représentée dans la Figure 2.9. Inversement, si FPQ désigne cette relation, on peut lui faire subir une opération de dégroupement et retrouver FP :

```
FPQ UNGROUP Piece_Qte
```

Four_No	Piece_Qte	
S1	<u>Piece_No</u>	Quantite
	P1	300
	P2	200
	P3	400
	P4	200
	P5	100
	P6	100
S2	<u>Piece_No</u>	Quantite
	P1	300
	P2	400
S3	<u>Piece_No</u>	Quantite
	P2	200
S4	<u>Piece_No</u>	Quantite
	P2	200
	P4	300
	P5	400

Figure 2.9 - Groupement de FP par Four_No

Inconvénients des RVA

La relation F_EXT (Figure 2.7) est la source d'un phénomène d'**asymétrie** fort gênant, empoisonnant la vie des utilisateurs d'un SGBD hiérarchique comme IMS/DL1, avec lequel on représente *de facto* les données à la manière des RVA.

Par exemple, si l'on considère les requêtes :

1. Quels fournisseurs ont livré la pièce P2 ?
2. Quels sont les pièces livrées par le fournisseur S2 ?

Dans un contexte symétrique, elles sont traduites ainsi (en Tutorial D) :

1. (FP WHERE Piece_No = Piece_No ('P2')) {Four_No}
2. (FP WHERE Four_No = Four_No ('S2')) {Piece_No}

Tandis que dans un contexte asymétrique, elles donnent lieu à celles-ci :

1. ((F_EXT UNGROUP (Piece_Qte)) WHERE Piece_No = Piece_No ('P2')) {Four_No}
2. ((F_EXT WHERE Four_No = Four_No ('S2')) UNGROUP ((Piece_Qte)) {Piece_No}

Quant aux contraintes du genre intégrité référentielle, les choses peuvent se compliquer. Si définir une clé étrangère entre les relvars FP et P est chose simple :

```
FOREIGN KEY {Piece_No} REFERENCES P (Piece_No)
```

Entre F_EXT et P, ça l'est moins :

```
(F_EXT UNGROUP (Piece_Qte)) {Piece_No} ⊆ P {Piece_No}
```

Considérons maintenant les opérations de mise à jour. Exemple :

1. Ajouter dans la base de données le fait que le fournisseur S5 a livré la pièce P5 en quantité 100.
2. Ajouter dans la base de données le fait que le fournisseur S2 a livré la pièce P5 en quantité 200.

Dans un contexte symétrique, elles sont traduites ainsi :

1. INSERT FP RELATION {TUPLE {Four_No Four_No ('S5'),
Piece_No Piece_No ('P5'),
Quantite Quantite (100)}} ;
2. INSERT FP RELATION {TUPLE {Four_No Four_No ('S2'),
Piece_No Piece_No ('P5'),
Quantite Quantite (200)}} ;

Dans un contexte asymétrique, l'affaire se corse :

1. INSERT F_EXT RELATION {TUPLE {Four_No Four_No ('S5'),
Piece_Qte RELATION {TUPLE {Piece_No Piece_No ('P5'),
Quantite Quantite (100)}}}} ;
2. UPDATE F_EXT WHERE Four_No = Four_No ('S2')
(INSERT Piece_Qte RELATION {TUPLE {Piece_No Piece_No ('P5'),
Quantite Quantite (200)}}) ;

Etc.

RVA ou pas RVA ?

Si l'on se limite à l'aspect structurel des choses, les relvars « matriochkas » sont très séduisantes, mais le phénomène d'asymétrie inhérent nous complique singulièrement la vie quand il s'agit de mettre à jour ces relvars au moyen de l'algèbre relationnelle, ou encore d'en garantir l'intégrité. A cet égard, quel soulagement quand, au milieu des années quatre-vingts, on put passer d'IMS/DL1 à DB2 et que l'on n'eut plus à polémiquer pour savoir si c'était l'entité-type CLIENT qui absorbait l'entité-type CONTRAT ou inversement (bon d'accord, avec les « relations logiques » on s'en sortait, mais c'était quand même un peu le parcours du combattant...)

2.7. Pour conclure avec la première forme normale

Si l'on reprend les définitions formelles données par Chris Date dans [Date 2004] au chapitre 6, « Relations », et reprises ici en annexe (paragraphe A), le bilan concernant les relations est le suivant :

- (a) Tout tuple contient exactement une valeur pour chacun de ses attributs.
- (b) Il s'ensuit que dans chaque relation, chaque tuple contient exactement une valeur pour chacun de ses attributs.
- (c) Une relation qui vérifie cette propriété est dite **normalisée**, ou de façon équivalente, est en **première forme normale** (1NF).
- (d) Il s'ensuit que toutes les relations en conformité avec (a) sont en **1NF**.

CQFD. Si par le passé Date a donné comme tout le monde une définition de la 1NF impliquant l'atomicité (cf. paragraphe 2.3) et donc l'illégalité des RVA, après avoir étudié la vraie nature des types (domaines), il a été amené à remplacer la contrainte de l'atomicité par celle, moins restrictive, de l'unicité de la valeur prise dans chaque tuple par chaque attribut : dans le contexte du Modèle Relationnel de Données, chaque relation est ainsi *de facto* en 1NF.

Maintenant, une relation en 1NF est-elle pour autant parée de toutes les vertus ? L'étude des autres formes normales (2NF et à suivre) montrera qu'il n'en est pas forcément ainsi (loin s'en faut !) et, comme nous l'avons mentionné précédemment, on pourra être amené à couper en deux cette relation, de manière rigoureuse, grâce aux moyens que nous ont fournis Codd, Boyce, Heath, Rissanen, Fagin, Armstrong et autres chercheurs.

2.8. Le bêtisier. Les définitions non conformes de la 1NF

Si l'on explore le Web, on trouve des dizaines de milliers d'occurrences faisant mention de l'expression « first normal form ». Les définitions farfelues qu'on y trouve abondent. D'autres présentent un caractère de sérieux certain, mais sont fausses. Voici deux exemples caractéristiques relevés dans la littérature.

Un 1er exemple

Le concept de « groupe répétitif » accompagne souvent celui d'atomicité, et si l'on cherche sur le Web des expressions telles que « repeating groups » ou « groupes répétitifs », on constate qu'il n'est pas rare qu'elles soient incorporées dans la définition de la première forme normale, et c'est à juste titre. Le problème est que cette fois-ci, concernant ces groupes répétitifs, on trouve couramment une interprétation totalement différente de celle qu'en ont faite Serge Miranda, Chris Date et autres théoriciens ou chercheurs (cf. [supra](#), paragraphe 2.4).

A titre d'exemple, voici la définition de la 1NF selon DB2 for z/OS (Cf. « Entity normalization » (*sic* !) in *DB2 Version 9.1 for z/OS, Administration Guide*), définition dans laquelle l'auteur se vautre de A à Z, je traduis :

« Une entité relationnelle satisfait à la contrainte de première forme normale si chaque instance d'entité contient une seule valeur, jamais d'attributs qui se répètent. Les attributs répétitifs, souvent appelés groupes répétitifs, sont des attributs distincts, mais il s'agit fondamentalement du même. Dans une entité conforme à la première forme normale, chaque attribut est indépendant et unique quant à sa signification et à son nom. »

A la nouvelle interprétation près de ce qu'est un groupe répétitif (ensemble d'attributs et non pas de valeurs au sein d'un attribut), cette définition est plutôt absconse et pour être sûr d'en comprendre le sens, il n'est pas inutile de considérer l'exemple qui l'accompagne et qui en illustre la fausseté. En effet, l'auteur écrit (je traduis à nouveau) :

« Supposons qu'une entité contienne les attributs suivants

EMPLOYEE_NUMBER

JANUARY_SALARY_AMOUNT, FEBRUARY_SALARY_AMOUNT, MARCH_SALARY_AMOUNT

Cette situation viole la contrainte de première forme normale, parce que JANUARY_SALARY_AMOUNT, FEBRUARY_SALARY_AMOUNT, and MARCH_SALARY_AMOUNT sont essentiellement le même attribut, EMPLOYEE_MONTHLY_SALARY_AMOUNT. »

Certes, les trois attributs incriminés doivent passer à la trappe et sont à remplacer par le seul attribut EMPLOYEE_MONTHLY_SALARY_AMOUNT, mais ceci relève seulement de l'art de la **modélisation conceptuelle** et n'a strictement rien à voir avec la 1NF. En effet, bien que l'auteur ait omis de préciser le domaine de référence commun aux trois attributs prétendument peccamineux, ainsi qu'à l'attribut EMPLOYEE_MONTHLY_SALARY_AMOUNT, on peut supposer qu'il s'agit de celui des entiers (ou des réels, peu importe), scalaire donc (cf. paragraphe 2.5), et dans ces conditions la table EMPLOYEE_NUMBER est en 1NF.



N.B. La documentation que j'ai citée et dont est tiré cet exemple mériterait d'être relue sérieusement. Par exemple, l'expression « *si chaque instance d'entité contient une seule valeur* », peut être littéralement réécrite ainsi : « s'il n'existe pas dans la table deux lignes ayant même valeur », sous-entendu si une clé primaire a été définie. Bref, en plus de donner une définition fausse, l'auteur amalgame de façon floue et absconse des concepts indépendants les uns des autres et ne relevant aucunement de la 1NF. (Inutile de le lui faire observer, je m'en suis chargé et j'ai même eu une réponse du Team Lead : « *I have opened a defect against this documentation and will be working with development to update the information as soon as possible* ». Ce que j'interprète volontiers ainsi : *You can always run, my little rabbit...*)

Un 2e exemple

Il existe aussi des définitions qui peuvent conduire à des aberrations et à des contradictions. Voici un exemple tiré d'un ouvrage dont je ne nommerai évidemment pas l'auteur (agrégé de ceci, DEA de cela, etc.), ouvrage dans lequel on peut lire ces lignes :

« Un tableau est en première forme normale si toutes ses colonnes sont élémentaires et s'il admet au moins une clef. »

« Tableau » est en l'occurrence synonyme de « relation » et « élémentaire » synonyme d'« atomique ». La 1re affirmation exprimée dans cet énoncé est correcte, mais la 2e : « *s'il admet au moins une clef* » doit être évacuée¹. En effet, du point de vue du Modèle Relationnel, la nécessité de l'existence d'une clé ne relève absolument pas de la normalisation, mais se règle en amont : c'est une conséquence de la propriété des relations, selon laquelle celles-ci ne peuvent pas contenir de n-uplets doublons : une relation coddienne est un ensemble fini, or un ensemble ne contient pas d'éléments en double. A défaut, l'algèbre relationnelle fournirait des résultats faux.

On pourrait se dire : D'accord, la nécessité d'une clé ne relève pas de la normalisation, mais ces définitions ne sont pas source de dangers. Malheureusement, sur la base de cette définition erronée, notre clerc bien imprudent aboutit à des conclusions délirantes du genre :

« Un Tableau en Quatrième Forme Normale n'est pas en Troisième Forme Normale, ni même en Deuxième, tout simplement parce qu'il n'est pas en Première Forme Normale. En effet, la condition essentielle pour qu'un Tableau soit en Première Forme Normale est l'existence d'une Clef. »

En effet, il est parti de sophismes, d'énormités du genre :

« Il n'y a pas de Clef dans un Tableau en Quatrième Forme Normale, et celui-ci n'est donc pas en Première Forme Normale. »

Et j'en passe.

La ligne jaune est franchie, notre clerc administre la preuve de sa méconnaissance des fondements mêmes du Modèle Relationnel de Données, doublée d'une dose d'inconscience rare. Pour faire bonne mesure, tout en oubliant de fournir la définition de la 4NF (et pour cause), il va jusqu'à affirmer que des auteurs aussi incontestables que Delobel et Adiba seraient les seuls à avoir démontré que celle-ci implique la BCNF, or Fagin (père de la 4NF) l'avait déjà fait en 1977. Qui plus est, à partir de ses prémisses abracadabrantes, le clerc va jusqu'à « prouver » que la démonstration de Delobel et Adiba (cf. paragraphe 4.14 ci-dessous) « ne démontre rien »... Mais on peut supposer que, si d'aventure, ces éminents chercheurs ont parcouru la prose les mettant en question, ils l'auront fait « d'un derrière distrait » comme disait Flaubert.

¹ Curieusement, des auteurs bien connus des cercles merisiens incorporent eux aussi à la définition de la 1NF la nécessité de l'existence d'une clé. Citons [Morejon 1992] :

« Une relation est en 1NF si :

- elle possède une clé
- tous ses attributs sont atomiques. »

Ou encore [RoMo 1989] qui tricote allègrement concepts merisiens et relationnels (page 82 de l'ouvrage) :

« Pour répondre à la 1FN tout individu doit posséder un identifiant. »

3. Forme normale de Boyce-Codd, deuxième et troisième formes normales

3.1. Les états d'âme provoqués par la première forme normale

3.1.1. Une tentative de normalisation en 1NF

Considérons à nouveau la table Membre de la Figure 2.3, qui pourrait représenter une ébauche de la table des membres de DVP. Au sens de Codd, cette table n'est pas normalisée en 1NF et pour la conserver sans la casser en deux, tout en normalisant, on pourrait la structurer ainsi (certains noms d'attributs ont été modifiés, pour des contraintes de représentation graphique) :

```
CREATE TABLE Membre
(
    MbrId          INTEGER          NOT NULL
  , Pseudo        VARCHAR(16)      NOT NULL
  , DateInscr     DATE              NOT NULL
  , Localisation  VARCHAR(48)      NOT NULL
  , AdrCourriel   VARCHAR(48)      NOT NULL
  , MessId        INTEGER          NOT NULL
  , MessDate      TIMESTAMP        NOT NULL
  , MessTexte     VARCHAR(MAX)     NOT NULL
  , Forum         CHAR(8)          NOT NULL
  , CONSTRAINT MbrPk PRIMARY KEY (MbrId, MessId)
  , CONSTRAINT CK2 UNIQUE (Pseudo, MessId)
  , CONSTRAINT CK3 UNIQUE (AdrCourriel, MessId)) ;
```

Figure 3.1 - Table Membre, normalisée en 1NF

On notera au passage qu'il a fallu faire évoluer la composition des contraintes MbrPk, CK2 et CK3. La clé primaire de la table est composée de la paire {MbrId, MessId} (attributs soulignés ci-dessous).

<u>MbrId</u>	Pseudo	DateInscr	Localisation	AdrCourriel	<u>MessId</u>	MessDate	MessTexte	Forum
31411	fsmrel	08/09/2006	Relationland	fsmrel@xxx.re	1	08/09/2006	J'ai un problème ...	Schéma
31411	fsmrel	08/09/2006	Relationland	fsmrel@xxx.re	2	21/09/2006	J'aimerais savoir ...	Merise
31411	fsmrel	08/09/2006	Relationland	fsmrel@xxx.re	3	01/02/2007	C'est quoi SQL ? ...	SQL
31411	fsmrel	08/09/2006	Relationland	fsmrel@xxx.re	4	02/03/2008	Rien ne marche, ...	Schéma
40001	Albert	01/04/2008	Paname	albert@yyy.zz	1	01/04/2008	Bonjour à tous, ...	SQL
40001	Albert	01/04/2008	Paname	albert@yyy.zz	2	02/04/2008	Merci ! Merci ! ...	SQL
40001	Albert	01/04/2008	Paname	albert@yyy.zz	3	02/04/2008	Bonjour à tous, ...	DB2
40001	Albert	01/04/2008	Paname	albert@yyy.zz	4	03/04/2008	Je rame, ...	DB2
10000	Marc	11/03/2002	Papeete	marc@zzz.pf	1	11/03/2002	Hello World ! ...	Merise
10000	Marc	11/03/2002	Papeete	marc@zzz.pf
10000	Marc	11/03/2002	Papeete	marc@zzz.pf	20000	12/03/2002	Le ciel est bleu, ...	Merise

Figure 3.2 - Table Membre, normalisée en 1NF (corps de la table)

3.1.2. Conséquences de la normalisation : des redondances à profusion

Au vu du contenu de la table de la Figure 3.2, on peut constater la présence de **redondances** (en rouge) ne faisant que polluer la table. On apprend ainsi quatre fois — c'est-à-dire autant de fois qu'il a échangé des messages — que le membre 31411 a pour pseudonyme fsmrel, qu'il s'est inscrit le 08/09/2006, qu'il vit dans le Relationland et qu'il a pour

adresse fsmrel@xxx.re. Pour sa part, Marc en est à 20 000 messages : la normalisation en 1NF de la table Membre a provoqué une inflation déraisonnable de redondances, faisant que la table prend de l'embonpoint.

Exemple :

Coût du stockage (DB2 for z/OS V8). Par référence aux tables telles qu'elles sont décrites dans la Figure 2.4 et la Figure 3.1, et sur la base d'un taux de compression d'environ 75% pour la table Membre et de 90% pour les autres tables, le coût du stockage est de l'ordre de celui-ci (en notant que dans le cas du couple de tables {Membre, Message}, l'index primaire de la table Message, de clé {MbrId, MessageId} sert aussi pour la clé étrangère {MbrId}) :

Objet	Nb lignes	Coût (Mo)	Niveaux index	(1 Mo = 2 ²⁰ octets)
Table Membre	5 000	0,20		
Index primaire		0,08	2	
Index unique (Pseudo)		0,14	2	
Index unique (AdrCourriel)		0,33	3	
Total		0,75		
Table Message	100 000	23,97		
Index primaire		1,68	3	
Total		25,65		
Total Membre + Message		26,40		
Table unique	100 000	25,46		
Index primaire		1,68	3	
Index Pseudo		3,03	3	
Index AdrCourriel		6,68	3	
Total table unique		36,85		

Coûts de stockage (DB2)

Contrairement à ce que l'intuition pourrait faire croire, dans le cas de la table unique (figure 3.1), le coût de stockage est supérieur à celui des tables Membre + Message de la figure 2.4.

On peut s'interroger sur le nombre de lectures/écritures en cache ou sur disque nécessaires pour, par exemple, modifier la localisation ou l'adresse de courriel de quelqu'un qui, comme Marc, a rédigé 20 000 messages. Dans le cas de l'adresse de courriel, ne pas oublier que ces lectures/écritures concernent aussi l'index utilisé pour garantir l'unicité des couples {MbrId, AdrCourriel}. Sans oublier non plus les désorganisations qui s'ensuivent, l'encombrement des fichiers journaux, les contentions entre transactions concurrentes, etc.

Tout cela, indépendamment d'une catégorie de problèmes bien connus de mise à jour, inhérents à la non normalisation, très tôt pointés par Codd et repris par tous les auteurs, problèmes évoqués dans le paragraphe 3.1.3 qui suit.

Quant aux performances concernant la lecture, si avec la table unique on évite une jointure, un prototypage en ce sens et des mesures poussées pourraient bien montrer qu'à part quelques requêtes pour lesquelles on ferait des économies (de bouts de chandelle à dire vrai), certains coûts sont cachés et peuvent être élevés (tris déclenchés par l'emploi de la clause SQL DISTINCT, etc.) sans compter que le nombre de certains niveaux d'index augmente. Se reporter au paragraphe 3.8 pour juger de prétendus méfaits dont certains accusent bien à tort la jointure, sans avoir vérifié.

Ce qui se passe réellement dans la soute n'a pas grand-chose à voir avec ce que l'on peut imaginer du haut de la dunette...

3.1.3. Les difficultés de mise à jour (Insert, Delete, Update)

L'organisation de la table Membre (Figure 3.1) pose quelques problèmes concernant les mises à jour.

Insert : On ne peut pas prendre en compte l'inscription des membres tant qu'ils n'ont pas envoyé un premier message (intégrité d'entité oblige, cf. paragraphe 3.2.6 : l'attribut MessId participe à la clé primaire et ne peut être marqué « null »).

Delete : Si l'on veut supprimer les messages de fsmrel, on est carrément obligé de supprimer ce dernier (intégrité d'entité oblige encore).

Update : Suite à l'exécution d'une requête mal codée, rien n'empêche de modifier le pseudonyme d'un membre seulement dans certaines lignes qui le concernent et « d'oublier » de le faire dans d'autres lignes. Bien sûr, à coups de contraintes (triggers SQL en général), on peut contrôler qu'un membre n'a qu'un pseudonyme, qu'une date d'inscription, etc. Mais quelle surcharge de travail pour le développeur et le DBA, et quel surcoût en matière de consommation de ressources pour le SGBD chargé de garantir les contraintes...

3.1.4. Quelle alternative ?

Que faire ? Se résigner à mettre en oeuvre la table Membre (Figure 3.1) respectant la 1NF ? Revenir à la situation initiale, en conservant une structure calquée sur celle de la relation de la Figure 2.3 ? Encore faudrait-il que le SGBD accepte les attributs à valeurs relations (RVA). Évidemment, aux difficultés près posées par ce genre d'attributs (cf. paragraphe 2.6), les redondances disparaissent et la table subit une sérieuse et bénéfique cure d'amaigrissement ; désormais (grâce à l'ensemble vide) une inscription peut être effectuée sans qu'il soit besoin d'attendre que le nouvel inscrit (rismo) ait envoyé son premier message ; de même, on peut supprimer tous les messages de Marc (sitôt dit, sitôt fait) sans que celui-ci disparaisse pour autant, etc., suite à quoi la table aurait l'allure suivante (structure et valeur) :

MbrId	Pseudo	DateInscr	Localisation	AdrCourriel	Messages																				
31411	fsmrel	08/09/2006	Relationland	fsmrel@xxx.re	<table><tr><th>MessId</th><th>MessDate</th><th>MessTexte</th><th>Forum</th></tr><tr><td>1</td><td>08/09/2006</td><td>J'ai un problème ...</td><td>Schéma</td></tr><tr><td>2</td><td>21/09/2006</td><td>J'aimerais savoir ...</td><td>Merise</td></tr><tr><td>3</td><td>01/02/2007</td><td>C'est quoi SQL ? ...</td><td>SQL</td></tr><tr><td>4</td><td>02/03/2008</td><td>Rien ne marche, ...</td><td>Schéma</td></tr></table>	MessId	MessDate	MessTexte	Forum	1	08/09/2006	J'ai un problème ...	Schéma	2	21/09/2006	J'aimerais savoir ...	Merise	3	01/02/2007	C'est quoi SQL ? ...	SQL	4	02/03/2008	Rien ne marche, ...	Schéma
MessId	MessDate	MessTexte	Forum																						
1	08/09/2006	J'ai un problème ...	Schéma																						
2	21/09/2006	J'aimerais savoir ...	Merise																						
3	01/02/2007	C'est quoi SQL ? ...	SQL																						
4	02/03/2008	Rien ne marche, ...	Schéma																						
40001	Albert	01/04/2008	Paname	albert@yyy.zz	<table><tr><th>MessId</th><th>MessDate</th><th>MessTexte</th><th>Forum</th></tr><tr><td>1</td><td>01/04/2008</td><td>Bonjour à tous, ...</td><td>SQL</td></tr><tr><td>2</td><td>02/04/2008</td><td>Merci ! Merci ! ...</td><td>SQL</td></tr><tr><td>3</td><td>02/04/2008</td><td>Bonjour à tous, ...</td><td>DB2</td></tr><tr><td>4</td><td>03/04/2008</td><td>Je rame, ...</td><td>DB2</td></tr></table>	MessId	MessDate	MessTexte	Forum	1	01/04/2008	Bonjour à tous, ...	SQL	2	02/04/2008	Merci ! Merci ! ...	SQL	3	02/04/2008	Bonjour à tous, ...	DB2	4	03/04/2008	Je rame, ...	DB2
MessId	MessDate	MessTexte	Forum																						
1	01/04/2008	Bonjour à tous, ...	SQL																						
2	02/04/2008	Merci ! Merci ! ...	SQL																						
3	02/04/2008	Bonjour à tous, ...	DB2																						
4	03/04/2008	Je rame, ...	DB2																						
10000	Marc	11/03/2002	Papeete	marc@zzz.pf	<table><tr><th>MessId</th><th>MessDate</th><th>MessTexte</th><th>Forum</th></tr><tr><td></td><td></td><td></td><td></td></tr></table>	MessId	MessDate	MessTexte	Forum																
MessId	MessDate	MessTexte	Forum																						
54321	rismo	01/09/2008	Papeete	rismo@xyz.fr	<table><tr><th>MessId</th><th>MessDate</th><th>MessTexte</th><th>Forum</th></tr><tr><td></td><td></td><td></td><td></td></tr></table>	MessId	MessDate	MessTexte	Forum																
MessId	MessDate	MessTexte	Forum																						

Figure 3.3 - Table Membre, avec l'attribut Messages dans une configuration RVA

3.1.5. Approche conceptuelle de la solution

Il y a bien sûr une alternative. Celui qui a l'habitude de pratiquer l'art de la modélisation conceptuelle, qu'il s'agisse de produire des MCD ou des diagrammes de classes, n'hésitera guère et fournira un diagramme qui, par dérivation, donnera lieu à des tables normalisées en 1NF, mais dépourvues d'attributs RVA, ce qui tombe à pic quand le SGBD ne permet pas la mise en oeuvre de tels attributs. Disons que l'on procède en l'occurrence selon une démarche dite synthétique, tandis que dans une étape suivante, on vérifiera la qualité du résultat, selon une démarche dite

analytique, mettant en jeu les techniques de normalisation qui permettront de s'assurer que l'architecture de la base de données est valide, une fois éliminées les redondances. En fait, on pourra toujours améliorer cette architecture pour éviter par exemple la présence du bonhomme NULL, ou pour anticiper quant à la performance des applications, mais ceci sort du cadre de la normalisation. Quoi qu'il en soit, le concepteur modélisera quelque chose comme ceci :

MCD (Notation IEF, Information Engineering Facility)

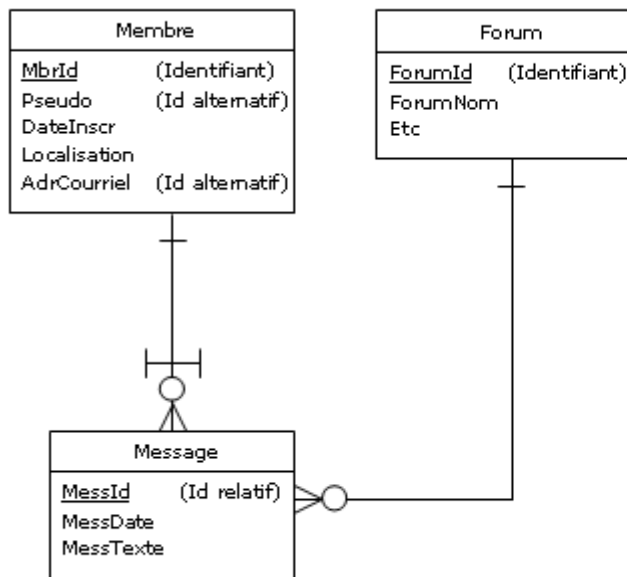


Diagramme de classes

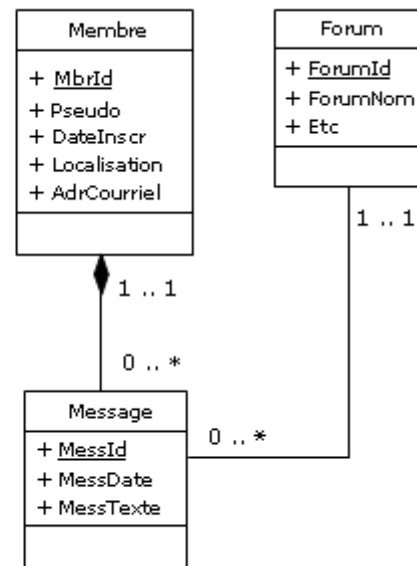


Figure 3.4 - Membres des forums - vision conceptuelle

Ce qui donnera lors de l'étape de dérivation un diagramme logique tel que celui-ci :

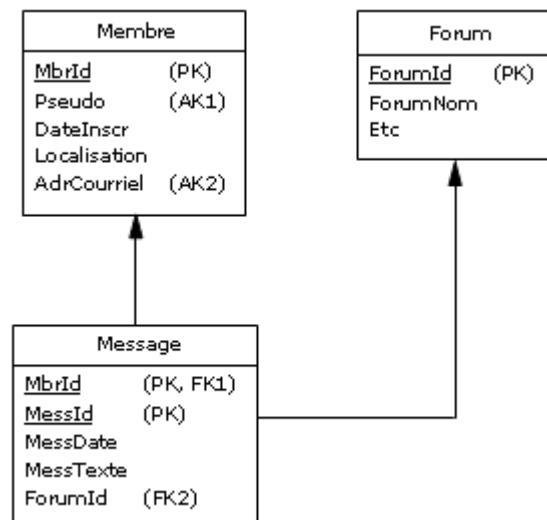


Figure 3.5 - Membres des forums - diagramme logique

Autrement dit, lors de la génération du code SQL correspondant, on retrouvera la structure des tables Membre et Message de la Figure 2.4. Vous me direz : que de détours pour revenir en quelque sorte à la case Départ ! Mais il fallait bien voir qu'il y a plus d'une façon de modéliser avant de décider d'une stratégie. Et si l'on opte pour les structures des tables de la Figure 2.4, il ne restera plus qu'à démontrer qu'elles sont en cinquième forme normale — vaste programme, mais réalisable sans grand effort dans le cas de cet exemple, grâce aux théorèmes de Date et Fagin (cf. paragraphe 5.8) :

- Si une relvar est en 3NF et si chacune de ses clés candidates ne comporte qu'un seul attribut, alors cette relvar est en 5NF.
- Si une relvar est en BCNF et comporte au moins un attribut non-clé, alors cette relvar est en 5NF.

3.2. L'approche analytique, ses outils

3.2.1. Remarque préliminaire

Notre défi est donc de produire une structure de base de données qui soit valide, évolutive, manipulable, c'est-à-dire débarrassée des redondances dénoncées précédemment. La théorie relationnelle nous en donne les moyens. Elle met à notre disposition des outils dont je cite certains : dépendance fonctionnelle, dépendance multivaluée, dépendance de jointure, clé candidate, surclé, théorème de Heath, théorème de Fagin-Date, opérateurs de projection et de jointure (naturelle), axiomes d'Armstrong. Grâce à ces outils on pourra s'assurer que les tables sont normalisées en 2NF, 3NF, etc. Et par une pratique synthéto-analytique rappelant l'art du yoyo, on vérifiera la modélisation conceptuelle en conséquence.

Je propose ici de décrire les outils les plus indispensables, en commençant par les plus simples et qui permettent de s'assurer qu'une table est en forme normale de Boyce-Codd (BCNF), à savoir les dépendances fonctionnelles. Par ailleurs, chronologie oblige, il est d'usage de commencer par étudier la 2NF et la 3NF avant de passer à la BCNF : je procèderai néanmoins dans l'ordre inverse, car la 2NF et la 3NF ne présentent plus guère aujourd'hui d'intérêt qu'historique.

A noter que j'utiliserai ici la terminologie du Modèle Relationnel plutôt que celle du Modèle SQL, mais il est facile de traduire, en remplaçant relvar et relation par table, tuple par ligne, etc.

3.2.2. Dépendance fonctionnelle (DF)

Il est un concept capital sans lequel nous ne pourrions pas traiter de la normalisation, car non seulement il est impliqué systématiquement dans les énoncés des formes normales, mais en outre, il représente un outil décisif pour normaliser les relations : il s'agit de la dépendance fonctionnelle. C'est en quelque sorte notre rasoir d'Ockham¹.

Une dépendance fonctionnelle (DF) est une instruction de la forme :

$$X \rightarrow Y$$

où X et Y sont deux sous-ensembles d'attributs de l'en-tête d'une relvar R, et satisfaisant à la règle : pour une valeur de X, correspond **exactement** une valeur de Y, c'est-à-dire que si deux tuples ont la même valeur v_X pour X, alors ils ont aussi la même valeur v_Y pour Y.

On dit encore que Y est fonctionnellement dépendant de X, ou que X détermine fonctionnellement Y. X est appelé le **déterminant** de la DF et Y le **dépendant**.

Pour signifier que X ne détermine pas Y, on peut écrire : $X \nrightarrow Y$.

Considérons par exemple la relvar FPV, extension de la relvar FP décrite dans la Figure 2.6 et dont l'en-tête est enrichi, juste pour les besoins de la cause, d'un attribut Ville, utilisé pour savoir dans quelle ville est localisé un fournisseur :

FPV	<u>Four_No</u>	<u>Piece_No</u>	Quantite	Ville
	S1	P1	300	Lille
	S1	P2	200	Lille
	S1	P3	400	Lille
	S1	P4	200	Lille
	S1	P5	100	Lille
	S1	P6	100	Lille
	S2	P1	300	Paris
	S2	P2	400	Paris
	S3	P2	200	Paris
	S4	P2	200	Lille
	S4	P4	300	Lille
	S4	P5	400	Lille

Figure 3.6 - Relvar des liens fournisseurs - pièces (enrichie de l'attribut Ville)

¹ Guillaume d'Ockham (ou d'Occam), moine anglais du XIVe siècle, à qui l'on doit ce sage conseil de parcimonie : *pluralitas non est ponenda sine necessitate*, que nous pouvons interpréter ici comme : *Ne multipliez pas les redondances au-delà du nécessaire.*

La relvar FPV vérifie un certain nombre de DF. Ainsi, pour prendre en compte une règle de gestion stipulant qu'un fournisseur est localisé dans une et une seule ville, on écrit :

$\{\text{Four_No}\} \rightarrow \{\text{Ville}\}$

La paire $\{\text{Four_No}, \text{Piece_No}\}$ étant clé candidate (cf. paragraphe 3.2.3), on a aussi les DF suivantes (entre autres) :

$\{\text{Four_No}, \text{Piece_No}\} \rightarrow \{\text{Quantite}\}$

$\{\text{Four_No}, \text{Piece_No}\} \rightarrow \{\text{Ville}\}$



(L'utilisation des accolades est là pour rappeler que le déterminant et le dépendant d'une DF ne sont pas des attributs, mais des **ensembles** d'attributs, au sens de la théorie des ensembles).

Avant de poursuivre, procédons à une classification des dépendances fonctionnelles :

✓ **Dépendance fonctionnelle triviale**

Soit R une relvar et X et Y deux sous-ensembles quelconques d'attributs de l'en-tête de R. La dépendance fonctionnelle $X \rightarrow Y$ est dite **triviale** si Y est inclus dans X (inclusion au sens large).

La relvar FPV comporte donc (entre autres) les dépendances triviales suivantes :

$\{\text{Four_No}\} \rightarrow \{\text{Four_No}\}$

$\{\text{Four_No}, \text{Piece_No}\} \rightarrow \{\text{Four_No}\}$

$\{\text{Four_No}, \text{Piece_No}\} \rightarrow \{\text{Piece_No}\}$

$\{\text{Ville}, \text{Quantite}\} \rightarrow \{\text{Ville}\}$

Etc.

Malgré son caractère ... trivial, la DF triviale est importante, car elle fait l'objet du 1er axiome d'Armstrong (cf. paragraphe E en annexe), et elle intervient dans l'énoncé de la BCNF.

✓ **Dépendance fonctionnelle partielle**

Soit R une relvar, X un sous-ensemble d'attributs de R et C un attribut quelconque de R. La dépendance fonctionnelle $X \rightarrow \{C\}$ est **partielle** si elle n'est pas triviale et s'il existe Y strictement inclus dans X tel que $Y \rightarrow \{C\}$.

Ainsi, la relvar FPV comporte la dépendance fonctionnelle non triviale

$\{\text{Four_No}, \text{Piece_No}\} \rightarrow \{\text{Ville}\}$

laquelle est partielle, car il existe par ailleurs la dépendance fonctionnelle

$\{\text{Four_No}\} \rightarrow \{\text{Ville}\}$

Ce type de DF intervient dans l'étude de la 2NF.

✓ **Dépendance fonctionnelle irréductible à gauche (ou élémentaire ou totale)**

Soit R une relvar, X un sous-ensemble quelconque d'attributs de R et C un attribut quelconque de R. La dépendance fonctionnelle $X \rightarrow \{C\}$ est dite **irréductible à gauche** (ou encore **totale**, ou **élémentaire**) si elle n'est ni triviale ni partielle, c'est-à-dire s'il n'existe pas Y strictement inclus dans X tel que $Y \rightarrow \{C\}$.

Ainsi, la relvar FPV comporte la dépendance fonctionnelle irréductible à gauche :

$\{\text{Four_No}, \text{Piece_No}\} \rightarrow \{\text{Quantite}\}$

En effet, une quantité de pièces n'a de sens que pour un fournisseur et un type de pièce donnés : pour un fournisseur on peut avoir différentes quantités de pièces et il n'existe donc pas de DF $\{\text{Four_No}\} \rightarrow \{\text{Quantite}\}$. De la même façon, il n'existe pas de DF $\{\text{Piece_No}\} \rightarrow \{\text{Quantite}\}$.

Autre exemple : Au paragraphe E.6.2 en annexe, on montre qu'étant données la relvar R{A, B, C, D} et l'ensemble de DF : $\{\{A\} \rightarrow \{B, C\}, \{B\} \rightarrow \{C\}, \{A\} \rightarrow \{B\}, \{A, B\} \rightarrow \{C\}, \{A, C\} \rightarrow \{D\}\}$, la DF $\{A, C\} \rightarrow \{D\}$ est réductible.

Ce type de DF intervient essentiellement dans l'étude de la 2NF (et dans telle ou telle définition de la BCNF).



L'expression *dépendance fonctionnelle totale* est celle utilisée par Ted Codd : *full functional dependence* [Codd 1971]. L'expression *dépendance fonctionnelle irréductible à gauche* (*left-irreducible functional dependence*) apparaît chez Date dans la 6e édition de *An Introduction to Database Systems*. Le qualificatif *irréductible à gauche* est plus précis que celui de *totale* (et d'*élémentaire*) et fait intervenir, à juste titre, la fermeture d'un ensemble de DF (cf. paragraphes E.1 et E.6.2 en annexe).

3.2.3. Clé candidate

Une clé candidate (clé pour abrégé) est un sous-ensemble d'attributs K de l'en-tête d'une relvar R, respectant les deux contraintes suivantes :

- ✓ **Unicité.** Deux tuples distincts de R ne peuvent avoir simultanément même valeur de K.
- ✓ **Irréductibilité** (ou minimalité). Il n'existe pas de sous-ensemble strict de K garantissant la règle d'unicité.

(En notant qu'en vertu de la propriété d'unicité, la DF : $K \rightarrow \{A\}$ est vérifiée pour chaque attribut A de R.)

Ainsi, le sous-ensemble représenté par la paire {Four_No, Piece_No} est une clé candidate de la relvar FPV (clé, pour faire court). Considérons en effet les DF vues précédemment :


$\{Four_No, Piece_No\} \rightarrow \{Four_No\}$	(DF triviale)
$\{Four_No, Piece_No\} \rightarrow \{Piece_No\}$	(DF triviale)
$\{Four_No, Piece_No\} \rightarrow \{Quantite\}$	(DF irréductible à gauche)
$\{Four_No, Piece_No\} \rightarrow \{Ville\}$	(DF partielle)

Du fait de ces DF, on constate tout d'abord que la paire {Four_No, Piece_No} détermine fonctionnellement **chaque** attribut de FPV, ce qui répond à la règle d'unicité.

Ensuite, cette paire n'est pas réductible. En effet, les sous-ensembles {Four_No} et {Piece_No} qui la composent ne sont pas assujettis, pris isolément, à respecter la contrainte d'unicité (de fait, le fournisseur S1 apparaît légalement dans 6 tuples de la relvar FPV et la pièce P2 dans 4 tuples) et ne peuvent donc pas faire à eux seuls l'objet de clés candidates.

Puisqu'elle elle satisfait aux contraintes d'unicité et d'irréductibilité, la paire {Four_No, Piece_No} est clé candidate.

Les triplets {Four_No, Piece_No, Quantite} et {Four_No, Piece_No, Ville} garantissent la règle d'unicité, mais sont réductibles car le sous-ensemble {Four_No, Piece_No} inclus dans chacun d'eux garantit aussi la règle : ces triplets ne peuvent être clés candidates, pas plus que le quadruplet {Four_No, Piece_No, Quantite, Ville} réductible lui aussi.

 On observera qu'une relvar peut posséder plus d'une clé candidate. C'est le cas par exemple de la table Membre telle qu'elle est décrite dans la Figure 2.4 et dans sa version obèse (Figure 3.1), où la table est dotée des trois clés suivantes : {MbrId, MessId}, {Pseudo, MessId}, {AdrCourriel, MessId}.

3.2.4. Surclé

Le concept de surclé intervient souvent dans la théorie de la normalisation, il est donc important d'en donner la définition. La surclé est un surtype de la clé candidate :

Une surclé est un sous-ensemble d'attributs K de l'en-tête d'une relvar R, respectant la contrainte d'unicité imposée aux clés candidates, mais pas nécessairement celle d'irréductibilité.

La paire {Four_No, Piece_No}, les triplets {Four_No, Piece_No, Quantité}, {Four_No, Piece_No, Ville} et le quadruplet {Four_No, Piece_No, Quantité, Ville} constituent autant de surclés pour FPV.

On retiendra donc que :

- Un sous-ensemble K d'une relvar R est une surclé de R si et seulement si **chaque** attribut C de R en dépend fonctionnellement (trivialement, partiellement ou totalement).
- Une clé candidate est une surclé spécialisée, car elle est astreinte à respecter le principe d'irréductibilité outre celui d'unicité.

3.2.5. Sous-clé

Une sous-clé est un sous-ensemble (non strict) d'une clé candidate.

La paire {Four_No, Piece_No} et les singletons {Four_No} et {Piece_No} sont des sous-clés de la relvar FPV. Même chose pour l'ensemble vide {}.

On peut noter qu'une clé candidate est à la fois surclé et sous-clé.

3.2.6. Clé primaire

Dans le contexte du Modèle Relationnel lui-même, le concept de clé primaire est aujourd'hui obsolète (rasoir d'Ockham oblige). Mais du point de vue SQL, il est toujours bien présent : dans ces conditions, si une table T possède plusieurs clés candidates, on en privilégie une pour être **primaire**, et si T ne possède qu'une seule clé candidate, celle-ci l'est *de facto*. L'intention est non seulement d'« identifier » chaque tuple de T, mais aussi d'établir des ponts entre T et d'autres tables, garantir l'intégrité référentielle par le jeu des liens entre clés primaires et étrangères [Codd 1970]. Les clés candidates non retenues sont dites **alternatives** ou alternées, de rechange... (*Alternate keys*).

Avec SQL donc, même si cela est *a priori* arbitraire, il faut faire appel à quelques critères aidant à fixer son choix :

- ✓ Il est impératif qu'une clé primaire soit **stable**, c'est-à-dire qu'elle ne change jamais de valeur. En effet, du fait des contraintes référentielles impliquant clés primaires et étrangères, le remplacement d'une valeur de clé primaire peut déclencher une cascade de remplacements de valeurs de clés primaires/étrangères dans de nombreuses autres tables et finir par pénaliser très sensiblement l'activité d'une production informatique (accès physiques aux données, phénomènes de blocage, désorganisation des données au niveau physique, etc., et n'oublions pas que pour la majorité des SGBDR, la structure d'un tuple induit celle d'un enregistrement physique, hélas !)

Par exemple, si la table des fournisseurs comporte le numéro de SIREN en tant qu'attribut, il faut éviter de faire participer celui-ci à la clé primaire de la table. En effet, ce numéro est attribué par l'INSEE qui, tous les mois, fournit la liste des numéros erronés et à remplacer. Plus généralement, il ne faut pas utiliser des informations dont les valeurs sont fournies par un organisme extérieur à l'entreprise et susceptibles de changer, à l'instar de ces numéros.

De la même façon, il faut éviter d'utiliser pour les clés primaires des données dont les valeurs sont **significatives**, du genre : les deux premiers caractères représentent telle information, les trois suivants telle autre information, cela relativement aux deux premiers... Sinon, on en arrive à figer un système qui pourtant doit s'adapter aux évolutions propres à l'entreprise ou imposées par un système externe (l'ISBN codifié par l'ISO, l'EAN13 ou l'EAN8, les codes bancaires présents sur la ligne CMC7 des chèques, etc.)

Ainsi, une clé primaire ne doit pas pouvoir changer de valeur et ne doit donc pas être significative : elle doit être définie en accord avec l'administrateur de la base de données, lequel aura à terme le bébé sur les bras. En général, on en vient à définir un attribut artificiel de type entier, que l'on incrémente ou que l'on hache (tâches confiées généralement au SGBDR), ou encore de type TIMESTAMP, etc. (*Pour des raisons de lisibilité et ne pas décourager le lecteur, nous dérogerons à cette règle en ce qui concerne certains exemples qui émaillent cet article cf. par exemple le paragraphe 3.7, mais dans un contexte de production, on ne déroge pas*).

- ✓ Une clé primaire doit respecter l'**intégrité d'entité** au sens de Codd, c'est-à-dire qu'aucun de ses composants ne peut être **marqué** « NULL ».

Il est évident que si aucune clé candidate ne répond aux critères que l'on vient d'énumérer, alors il faudra inventer de toutes pièces une clé primaire qui soit conforme, en définissant à cet effet un attribut parfaitement **artificiel**. Ceci est à rapprocher de ce qu'a écrit Yves Tabourier au sujet des propriétés des entités-types (ou individus-types) [Tabourier 1986] page 80 :

« ... la fonction d'une propriété est de décrire les objets (et les rencontres), alors que l'identifiant ne décrit rien. Son rôle fondamental est d'être sûr de distinguer deux jumeaux parfaits, malgré des descriptions identiques.

L'expérience montre d'ailleurs que l'usage des "identifiants significatifs" (ou "codes significatifs") a pu provoquer des **dégâts tellement coûteux** que la sagesse est d'éviter avec le plus grand soin de construire des identifiants décrivant les objets ou, pis encore, leurs liens avec d'autres objets... »

On ne saurait mieux dire.

Observations concernant les clés primaires

Pour leur part, les continuateurs de l'oeuvre de Codd, à savoir Date & Darwen, estiment que le choix d'une clé candidate pour être clé primaire, relève de considérations d'ordre purement psychologiques car, en quelque sorte, « elle serait plus égale que les autres ». Reste son rôle en relation avec l'intégrité référentielle : que ce soit dans le cadre du Modèle Relationnel ou du Modèle SQL, une clé étrangère pouvant être en relation avec n'importe quelle clé candidate servant de référence, on pourrait simplement exiger que la clé référencée vérifie les critères précédemment énoncés (stabilité, absence de signification, marquage « null » interdit). Une réflexion attentive montre qu'il faut se rendre aux arguments développés par D & D dans [Date 1995] : les deux concepts de clé primaire et de clé alternative peuvent être évacués sans rien remettre en cause au plan fonctionnel. C'est pourquoi Tutorial D (le langage de D & D) les ignore, au bénéfice du seul concept de clé (sous-entendu candidate). Mais, dans le contexte de la modélisation, il peut être utile de raisonner en termes de clé primaire (comme interprétation de l'identifiant d'une entité-type d'un modèle conceptuel de données).

Les concepts de clé primaire et de clé alternative pourraient donc théoriquement être évacués de SQL, au bénéfice du seul concept de clé (toujours sous-entendu candidate), mais il y a malgré tout la réalité des bases de données en production, et l'on ne peut donc pas supprimer du langage un mot-clé omniprésent, simplement en claquant des doigts.

3.3. Forme normale de Boyce-Codd (BCNF)

On a besoin de techniques permettant d'éviter les anomalies de mise à jour (cf. paragraphe 3.1.3), en décomposant une relvar R en d'autres relvars R1, R2, ..., Rn, elles-mêmes dépourvues de ces anomalies. Décomposer signifie, réciproquement, savoir recomposer R. Pour cela, on met en oeuvre un procédé de normalisation, s'appuyant fondamentalement sur les relations qu'il peut y avoir entre clés et dépendances fonctionnelles.

Il faut d'abord être à même de vérifier si une relvar est à décomposer. L'énoncé fondamental permettant de s'en assurer porte le nom de forme normale de Boyce-Codd (BCNF pour *Boyce-Codd Normal Form*), cf. [Fagin 1977], [Ullman 1982], [Date 2008].

3.3.1. Énoncé de la BCNF

Énoncé de Boyce et Codd [Codd 1974], où la BCNF est encore nommée 3NF :

A relation R is in third normal form if it is in first normal form and, for every attribute, collection C of R, if any attribute not in C is functionally dependent on C, then all attributes in R are functionally dependent on C.

Énoncé de Date ([Date 2008]), plus au goût du jour que le précédent, je traduis :

Une relvar R est en forme normale de Boyce-Codd (BCNF) si et seulement si pour **chaque** dépendance fonctionnelle non triviale $A \rightarrow B$ qui doit être vérifiée par R, A est une surclé de R.


(A et B sont des sous-ensembles d'attributs de l'en-tête de la relvar R).

Variante 1, due à Carlo Zaniolo (cf. [Date 2004]), je traduis :

Soit R une relvar, X un sous-ensemble d'attributs de l'en-tête de R et A un attribut de cet en-tête. R est en forme normale de Boyce-Codd (BCNF) si et seulement si, pour **chaque** dépendance fonctionnelle $X \rightarrow \{A\}$ qui doit être vérifiée par R, au moins une des conditions suivantes est satisfaite :

1. A est un élément de X (cette dépendance fonctionnelle est donc triviale).
2. X est une surclé de R.

On peut ainsi constater que la relvar FPV (cf. Figure 3.6) n'est pas en forme normale de Boyce-Codd, puisqu'il existe la dépendance fonctionnelle non triviale $\{\text{Four_No}\} \rightarrow \{\text{Ville}\}$ dont le déterminant $\{\text{Four_No}\}$ ne constitue pas une surclé, car ne respectant pas la contrainte d'unicité (cf. paragraphe 3.2.3). Pour normaliser la relvar FPV, on se servira du théorème de Heath (voir ci-dessous).

 Petit problème en passant : pour prouver qu'une relvar R est en BCNF il faut d'abord dresser l'inventaire complet des DF non triviales satisfaites par R, et vérifier que chacune d'elles est une surclé. Or, une DF est de la forme $A \rightarrow B$, A et B étant des sous-ensembles des attributs de R. Comme à un ensemble de n éléments correspond 2^n sous-ensembles (compte tenu de l'ensemble vide), la limite supérieure du nombre de DF est égale à 2^{2n} . Certains chiffres découragent... Heureusement, on dispose d'un algorithme permettant de s'assurer assez rapidement du respect de la BCNF (cf. paragraphe E.5 en annexe).

Variante 2 de l'énoncé de la BCNF (basée sur les dépendances fonctionnelles irréductibles à gauche) :

Soit R une relvar, X un sous-ensemble d'attributs de l'en-tête de R et A un attribut de cet en-tête. R est en forme normale de Boyce-Codd (BCNF) si et seulement si, pour chaque dépendance fonctionnelle $X \rightarrow \{A\}$ qui doit être vérifiée par R, une des conditions suivantes est satisfaite :

1. Cette dépendance fonctionnelle est triviale (A est un élément de X).
2. Cette dépendance fonctionnelle est irréductible à gauche et X est une clé candidate de R.

Petite consolation concernant la chasse aux DF : La recherche peut être limitée aux DF irréductibles à gauche. Il reste que la méthode décrite au paragraphe E.6.2 sera encore d'un grand secours, par exemple pour prouver qu'une DF est irréductible. A noter que cette définition a pour origine une de celles qui sont proposées dans [Date 2004] :

« A relvar is in BCNF if and only if every nontrivial, left-irreducible FD has a candidate key as its determinant. »
(Une relvar est en BCNF si et seulement si le déterminant de chaque DF non triviale, irréductible à gauche est une clé candidate de la relvar.)

3.3.2. Théorème de Heath

On doit à Ian Heath un théorème très important (1971), que l'on mettra à profit pour normaliser :

Soit la relvar R {A, B, C} dans laquelle A, B et C sont des sous-ensembles d'attributs de R. Si R satisfait à la dépendance fonctionnelle $A \rightarrow B$, alors R est égale à la jointure de ses projections sur {A, B} et {A, C}.

Autrement dit, si R1 et R2 représentent les relvars résultant de ces projections, alors $R = R1 \text{ JOIN } R2$ (préservation du contenu de la base de données). On dispose là d'un moyen très efficace pour décomposer une relvar non normalisée. A user sans modération (sauf à autoriser la présence du bonhomme NULL, cf. paragraphe D en annexe), mais tout en ayant à l'esprit la règle de préservation des dépendances :

Préservation des dépendances

On doit à Jorma Rissanen une règle importante, grâce à laquelle on évitera de perdre des DF :

Soit la relvar R {A, B, C} dans laquelle A, B et C sont des sous-ensembles d'attributs de R. Si R satisfait aux dépendances fonctionnelles $A \rightarrow B$ et $B \rightarrow C$, alors plutôt que de décomposer R en {A, B} et {A, C}, on décomposera R en {A, B} et {B, C}.



En effet, en procédant ainsi, R est encore égale à la jointure de ses projections sur {A, B} et {B, C}, et surtout, on ne perd pas en route la DF $B \rightarrow C$, qui est quand même la traduction formalisée d'une **règle de gestion des données**. Se reporter aussi au paragraphe E.7.2 où l'on traite plus avant de la préservation des dépendances.

3.3.3. Comment normaliser une relvar qui n'est pas en BCNF

Dans le cas de la relvar FPV (cf. Figure 3.6), pour évacuer le problème posé par la dépendance fonctionnelle $\{Four_No\} \rightarrow \{Ville\}$, si l'on a présent à l'esprit le théorème de Heath, il suffit de procéder ainsi :

1. Définir une relvar FV, qui soit la **projection** de la paire {Four_No, Ville} :

FV (Four_No, Ville)

(par application de la définition de la clé candidate (principes d'unicité et d'irréductibilité), la clé de FV est composée du seul attribut Four_No).

2. Définir une relvar FPQ qui soit la **projection** de FPV sur l'ensemble des attributs de FPV sauf Ville :

FPQ (Four_No, Piece_No, Quantite)

En sorte que par **jointure naturelle** de FV et FPQ on retrouve FPV :

FPV = FV JOIN FPQ

3.3.4. Décomposition par projection/jointure préservant l'information

Ainsi, le processus de normalisation met en jeu le mécanisme de **projection/jointure (naturelle)** :

- 1) Par projection d'une DF qui est la cause d'un viol de BCNF, on décompose une relvar R non normalisée en deux relvars R1 et R2 (mieux, sinon totalement) normalisés.

Et l'on sait que — en vertu du théorème de Heath — par jointure naturelle des relvars R1 et R2, on retrouve exactement la relation initiale, avec **préservation du contenu de la base de données**.

Ce dernier point est important, car le but de l'opération est de retrouver la relation initiale dans son intégrité : la normalisation par projection/jointure (*nonloss decomposition*) est une **décomposition préservant l'information**.

- 2) On recommence le processus tant qu'il y a encore à normaliser.

Application. Résultat de la normalisation de la relvar FPV (FPV = FV JOIN FPQ) :

FV	<u>Four_No</u>	Ville
	S1	Lille
	S2	Paris
	S3	Lyon
	S4	Lille

(On retrouve la relvar FP de la Figure 2.6, ainsi qu'un sous-ensemble de la relvar F).

FPQ	<u>Four_No</u>	<u>Piece_No</u>	Quantite
	S1	P1	300
	S1	P2	200
	S1	P3	400
	S1	P4	200
	S1	P5	100
	S1	P6	100
	S2	P1	300
	S2	P2	400
	S3	P2	200
	S4	P2	200
	S4	P4	300
	S4	P5	400

Figure 3.7 - Normalisation par décomposition de la relvar FPV

3.3.5. Normalisation et intégrité référentielle

Il est évident que lorsqu'on a normalisé en ayant tenu compte de la recommandation de Rissanen, l'intégrité référentielle doit à son tour être assurée. Ainsi, existe-t-il une contrainte de ce type entre les relvars FV et FPQ suite à la normalisation de la relvar FPV en BCNF (cf. ci-dessus) :

Version Tutorial D :

```
VAR FPQ BASE RELATION {
    Four_No    INTEGER
    , Piece_No  INTEGER
    , Quantite  INTEGER}
KEY {Four_No, Piece_No}
FOREIGN KEY {Four_No} REFERENCES FV {Four_No} ... ;
```

Version SQL :

```
CREATE TABLE FPQ (
    Four_No    INTEGER    NOT NULL
    , Piece_No  INTEGER    NOT NULL
    , Quantite  INTEGER    NOT NULL
    , CONSTRAINT FPQ_PK PRIMARY KEY (Four_No, Piece_No)
    , CONSTRAINT FPQ_AK1 FOREIGN KEY (Four_No) REFERENCES FV (Four_No) ... ) ;
```


3.3.6. Conséquences de la normalisation par projection - jointure

Une fois achevé le travail de normalisation, on observera que :


- ✓ Du point de vue de la modélisation conceptuelle, tout est plus cohérent.
- ✓ Les problèmes de mise à jour s'arrangent.
 - Ainsi, on peut désormais stocker dans la base de données le fait que le fournisseur S5 réside en Arles même s'il n'a effectué aucun envoi (comme dans le cas de la relvar F de la Figure 2.6, dont FV est une restriction/projection).
 - On peut aussi supprimer les envois d'un fournisseur, tout en conservant l'information comme quoi il réside dans telle ville.
 - On contraint un fournisseur à résider dans une seule ville, comme cela l'a été spécifié au niveau des règles de gestion : Nul besoin de définir une contrainte supplémentaire à cet effet.

3.4. Deuxième et troisième formes normales

3.4.1. BCNF versus 2NF et 3NF

Chronologiquement, la BCNF (présentée en 1974) fut précédée de deux autres formes normales, dites deuxième et troisième formes normales (2NF et 3NF), décrites initialement par Ted Codd (voir [Codd 1971] par exemple).

Au fil du temps, ces deux formes normales ont été, hélas ! parfois remaniées, faussées par des mains inexpérimentées. Quoi qu'il en soit, les deuxième et troisième formes normales sont moins efficaces : la BCNF permet d'éliminer des redondances face auxquelles ses deux sœurs aînées sont impuissantes. Et puis, il serait dommage de mémoriser trois définitions quand une seule suffit, celle de la BCNF, dont l'énoncé est conceptuellement plus simple et ne fait pas référence à la panoplie complète des dépendances fonctionnelles. Qui peut le plus peut le moins :

 Sur le terrain, on gagnera un temps précieux à vérifier directement la BCNF et à ne faire éventuellement appel à ses petites sœurs — moins efficaces — qu'en cas de problème, à savoir violer la BCNF pour éviter la perte d'une règle de gestion (cf. paragraphe 3.7).

3.4.2. Deuxième forme normale (2NF)

Dans son article de 1971, Ted Codd donne la définition suivante de la deuxième forme normale (2NF), dans laquelle nous conservons le terme *relation*, bien que celui de *relvar* y soit plus approprié :

Une relation R est en deuxième forme normale si elle est en première forme normale et si chaque attribut n'appartenant à aucune clé **candidate** de R est en dépendance totale de chaque clé candidate de R.

(« A relation R is in second form normal if it is in first normal form and every non-prime attribute of R is fully dependant on each candidate key of R. »)

Pour Codd, un *prime attribute*, est un attribut qui appartient à au moins une clé candidate de R, et *a contrario*, un *non-prime attribute* est un attribut qui n'appartient à aucune clé candidate de R (on peut préférer utiliser l'expression **attribut non clé**). Rappelons qu'une dépendance fonctionnelle est totale (ou irréductible à gauche ou élémentaire) si elle n'est ni triviale ni partielle (cf. paragraphe 3.2.2).

On peut ainsi constater que la relvar FPV (Figure 3.6) n'est pas en deuxième forme normale, car l'attribut Ville n'appartient pas à la clé candidate {Four_No, Piece_No}, mais en dépend partiellement puisque {Four_No} → {Ville}. Pour normaliser, on utilise une fois de plus le mécanisme de décomposition sans perte, par projection - jointure, en mettant à nouveau à profit le théorème de Heath.



Remarque importante :

Hélas, trois fois hélas ! La définition la plus courante donnée de la 2NF est la suivante :



Une relation R est en deuxième forme normale si elle est en première forme normale et si chaque attribut n'appartenant pas à la clé **primaire** de R est en dépendance totale de celle-ci.

Cette définition est bien sûr **incomplète**, car seule la clé primaire est prise en compte, tandis que les **clés alternatives** passent à la trappe. Il est probable qu'en se recopiant les uns les autres, les rédacteurs se sont inspirés de la définition proposée par Chris Date, mais sans tenir compte de l'avertissement que celui-ci a pourtant pris soin de donner au lecteur :

« Pour des raisons de simplicité, on supposera que chaque relvar a exactement une clé candidate, c'est-à-dire une clé primaire et aucune clé alternative. »

D'où la définition donnée par Date, valable **seulement** dans ce cas particulier :

A relvar is in 2NF if and only if it is in 1NF and every nonkey attribute is irreducibly dependant on the primary key.

Un attribut est non clé (*nonkey*) quand il n'appartient pas à la clé primaire de la relvar. Nombre de ceux qui ont recopié cette définition de Date n'ont malheureusement pas lu son avertissement et feraient bien de s'inspirer de [Bouzeghoub 1990], je cite : « Si une relation possède plusieurs clés, la définition doit être vérifiée pour chaque clé ».

A défaut, dans le cas général, cette définition de la 2NF qui ne tient compte que de la clé primaire est bien sûr trop faible : il est par exemple courant de mettre en oeuvre un attribut artificiel pour définir la clé primaire d'une table (ceci vaut au niveau conceptuel, *mutatis mutandis*), admettons. C'est ce qu'on a fait pour la relvar FPVX ci-dessous, supposée remplacer la relvar FP (cf. Figure 2.6), étendue de l'attribut Ville. Mais l'ajout de l'attribut PVX_Id, fait que la paire {Four_No, Piece_No} est ravalée au rang de clé alternative (clé primaire soulignée, clé alternative en italiques) :

FPVX	<u>PVX_Id</u>	<i>Four_No</i>	<i>Piece_No</i>	Quantité	Ville
	1	S1	P1	300	Lille
	2	S1	P2	200	Lille
	3	S1	P3	400	Lille
	4	S1	P4	200	Lille
	5	S1	P5	100	Lille
	6	S1	P6	100	Lille
	7	S2	P1	300	Paris
	8	S2	P2	400	Paris
	9	S3	P2	200	Lyon
	10	S4	P2	200	Lille
	11	S4	P4	300	Lille
	12	S4	P5	400	Lille

Figure 3.8 - La relvar FPVX, censée respecter 2e forme normale

PVX_Id étant le seul attribut entrant dans la composition de la clé primaire, tous les autres attributs sont en dépendance totale de celle-ci : FPVX respecte la deuxième forme normale incomplète ! Mais **la redondance et les problèmes de mise à jour et autres sont toujours bien présents...**

⇒ Bannissons la définition incomplète et utilisons celle de Codd (ou mieux, restons-en à la BCNF).

Par ailleurs, si la mise en oeuvre d'un attribut tel que PVX_Id s'avère nécessaire, ne l'envisageons que pour des relvars déjà normalisées en BCNF (ou de préférence en 5NF, car certaines erreurs pourraient encore se faufiler).

3.5. Dépendance transitive, dépendance directe

A son tour, la définition de la troisième forme normale fait intervenir un autre type de DF, à savoir la DF transitive. Soit deux sous-ensembles d'attributs X, Y appartenant à l'en-tête H d'une relvar R, et un attribut A appartenant également à H mais n'appartenant pas à Y. La dépendance fonctionnelle $X \rightarrow \{A\}$ est dite **transitive** si elle vérifie (cf. [Delobel 1982], « Normalité d'une relation ») :

- (1) $X \rightarrow Y$
- (2) $Y \rightarrow \{A\}$
- (3) $Y \not\rightarrow X$ (la dépendance fonctionnelle $Y \rightarrow X$ n'existe pas)
- (4) Y n'est pas inclus dans X (la dépendance fonctionnelle $X \rightarrow Y$ n'est donc pas triviale)

A *contrario* si {A} ne dépend pas transitivement de X, la dépendance fonctionnelle $X \rightarrow \{A\}$ est dite **directe**.

3.6. Troisième forme normale (3NF)

Dans son article de 1971, Ted Codd donne cette définition de la troisième forme normale (3NF) :

Une relation R est en troisième forme normale si elle est en deuxième forme normale et si chaque attribut n'appartenant à aucune clé **candidate** ne dépend directement que des clés candidates de R.

(« *A relation R is in third form normal if it is in second normal form and every non-prime attribute of R is non-transitively dependant on each candidate key of R.* »)

Rappel : pour Codd, un *prime attribute*, est un attribut qui appartient à au moins une clé candidate de R, et *a contrario*, un *non-prime attribute* est un attribut qui n'appartient à aucune clé candidate de R.

La traduction littérale de la définition de Codd ne serait pas très satisfaisante, d'où un léger aménagement. Peu importe, retenons ceci : pour que la 3NF soit respectée, un attribut n'appartenant pas à une clé candidate ne doit jamais dépendre directement d'un **sous-ensemble** d'attributs qui ne constitue pas lui-même une clé candidate.

Une définition plus intéressante est celle de Zaniolo, car elle montre que la BCNF implique la 3NF (il suffit en effet d'y supprimer la 3e condition pour retrouver la définition de la [BCNF](#)) :

Soit R une relvar, X un sous-ensemble d'attributs de l'en-tête de R et A un attribut de cet en-tête. R est en troisième forme normale si et seulement si, pour chaque dépendance fonctionnelle $X \rightarrow \{A\}$ qui doit être vérifiée par R, au moins une des conditions suivantes est satisfaite :

1. A est un élément de X (cette dépendance fonctionnelle est donc triviale).
2. X est une surclé de R.
3. A appartient à une clé candidate de R.

Bien entendu, il existe d'autres définitions de la troisième forme normale, dont la plus courante (hélas !) est la suivante :



Une relation R est en troisième forme normale si et seulement si :

1. Elle est en deuxième forme normale ;
2. Tout attribut n'appartenant pas à la clé **primaire** de R n'en dépend pas transitivement.

Comme dans le cas de la 2NF non coddienne, cette définition est incomplète (et même fausse dans le cas général, cf. paragraphe 3.9), puisque cette fois-ci encore elle est muette à propos des **clés alternatives**. Nombre de ceux qui ont recopié la définition de Chris Date ont une fois de plus oublié de tenir compte de son avertissement :

« *Pour des raisons de simplicité, on supposera que chaque relation a exactement une clé candidate, c'est-à-dire une clé primaire et aucune clé alternative.* »

A titre d'exercice, on peut montrer que la relvar FPVX de la Figure 3.8 enfreint la 3NF.

En effet, au sens de Codd, la relvar enfreint la 2NF, donc *a fortiori* la 3NF. Mais acceptons la définition incomplète de la 2NF, selon laquelle FPVX respecterait la 2NF. Toutefois, quelle que soit la définition que l'on utilise de la 3NF, on montre que FPVX enfreint la 3NF. De fait, puisque {PVX_Id} est clé primaire, on peut mettre en évidence les DF suivantes :

- DF1 : {PVX_Id} → {Four_No}
- DF2 : {PVX_Id} → {Piece_No}
- DF3 : {PVX_Id} → {Quantite}
- DF4 : {PVX_Id} → {Ville}

Pour être complet, comme {Four_No, Piece_No} est clé alternative, on a aussi :

DF5 : {Four_No, Piece_No} → {PVX_Id}

DF6 : {Four_No, Piece_No} → {Quantite}

DF7 : {Four_No, Piece_No} → {Ville}

En utilisant la règle d'union inférée des axiomes d'Armstrong (cf. paragraphe E en annexe), de DF1 et DF2, on produit la DF :

DF8 : {PVX_Id} → {Four_No, Piece_No}

Il existe aussi la DF à l'origine de l'infraction :

DF9 : {Four_No} → {Ville}

La relvar FPVX n'est pas en 3NF, puisqu'en fait DF4 peut être obtenue à partir de DF1 et DF9 (axiome d'Armstrong de transitivité), c'est-à-dire que l'attribut Ville qui n'appartient pas à la clé primaire {PVX_Id} de la relvar, dépend transitivement de cette clé, via le singleton {Four_No} qui lui-même n'est pas clé.

Conclusion :

a) N'utiliser que les définitions de Codd de la 2NF et de la 3NF (on peut aussi préférer Zaniolo), ou celles qui lui sont équivalentes (cf. par exemple [Delobel 1982], [Gardarin 1988]), et d'urgence mettre à la poubelle celles qui se limitent aux clés primaires.

b) Mieux encore : dans le travail de normalisation, au nom du rasoir d'Ockham, prendre l'habitude de laisser tomber la 2NF et la 3NF, au profit de la seule BCNF. On peut aussi arguer de ce choix si l'on est limité par le temps, un peu paresseux et doté d'une mémoire assez volatile pour ne pas retenir toutes les définitions.

Ainsi, dans le cas de l'exercice qui vient d'être proposé, sans perdre de temps à vérifier la 2NF puis la 3NF avant d'en arriver à la BCNF, le diagnostic est rapide et le remède simple et efficace :

Le déterminant {Four_No} de la DF non triviale DF9 n'étant pas une surclé de la relvar FPVX, celle-ci enfreint la BCNF et il faut la décomposer. Pour cela, on applique le théorème de Heath, sans oublier la règle de Rissanen de préservation des dépendances (cf. paragraphe 3.3.2).

Procédons ainsi, en représentant d'abord la relvar FPVX sous la forme suivante :

FPVX { {Four_No}, {Ville}, {PVX_Id, Piece_No, Quantite} } ;

Puisque l'on a la DF {Four_No} → {Ville}, la relvar peut être ainsi décomposée (avec préservation des DF) :

FPVXa {Four_No, Ville}

FPVXb {Four_No, PVX_Id, Piece_No, Quantite}

FPVXa n'est jamais que la projection sur les attributs Four_No et Ville de la relvar F (Figure 2.6), et comme elle n'en est qu'un sous-ensemble elle peut disparaître.

FPVXb n'est jamais que de la relvar FP (Figure 2.6) ayant subi une extension (ajout de l'attribut PVX_Id). En fait, l'attribut PVX_Id n'apporte strictement rien au plan de la modélisation et des opérations : au nom du rasoir d'Ockham, il doit passer à la trappe, à la suite de quoi on retrouve FP.

3.7. Un problème embarrassant de BCNF

On peut être confronté au dilemme suivant :

Si une relvar n'est pas en BCNF, par application du théorème de Heath, on sait la décomposer sans perte, mais il y a un hic : on pourrait ne pas respecter la règle de préservation des dépendances (cf. paragraphe 3.3.2 et annexe E.7.2), c'est-à-dire qu'en décomposant, on perdrait une DF et par voie de conséquence, la règle de gestion dont cette DF est issue ne serait plus garantie. Le remède peut être pire que le mal. Que faire ?

Illustrons ceci à l'aide d'un exemple proposé par Chris Date (cf. [Date 2004], page 370), en considérant la relvar EMP suivante (francisée) :

EMP	<u>Etudiant</u>	<u>Matiere</u>	Professeur
	Albert	Latin	M. Lebrun
	Albert	Grec	Mme Prévert
	Nanard	Latin	M. Lebrun
	Nanard	Grec	Mlle Martin

Figure 3.9 - Relvar EMP

Règles de gestion extraites du corpus des règles consignées dans le dossier de conception qui nous a été transmis :

- (R1) Pour chaque matière étudiée par un étudiant, celui-ci n'a qu'un professeur.
- (R2) Chaque professeur n'enseigne qu'une matière.
- (R3) Chaque matière peut être enseignée par plusieurs professeurs.
- (R4) Chaque professeur peut enseigner sa matière à plusieurs étudiants.
- ... Etc.

Traduction de ces règles sous forme de dépendances fonctionnelles :

- DF1 : {Etudiant, Matiere} → {Professeur}
- DF2 : {Professeur} → {Matiere}

La relvar EMP ne respecte pas la BCNF, car le déterminant {Professeur} de DF2 n'est pas une surclé, un professeur pouvant avoir plusieurs étudiants (par exemple, M. Lebrun a au moins deux étudiants, Albert et Nanard).

On normalise en appliquant le théorème de Heath sur la base de DF2, en décomposant donc ainsi la relvar EMP :

- PM {Professeur, Matière} ayant pour seule clé candidate {Professeur}.
- PE {Professeur, Etudiant} ayant pour seule clé candidate {Professeur, Etudiant}.

PM	<u>Professeur</u>	Matiere
	M. Lebrun	Latin
	Mme Prévert	Grec
	Mlle Martin	Grec

PE	<u>Professeur</u>	<u>Etudiant</u>
	M. Lebrun	Albert
	Mme Prévert	Albert
	M. Lebrun	Nanard
	Mlle Martin	Nanard

Figure 3.10 - Projection de EMP selon PM et PE

Ces deux relvars sont en BCNF, l'information est préservée, mais **on perd DF1, donc la règle de gestion (R1)**, et si Mlle Martin enseigne le grec à Nanard, rien n'empêcherait désormais que Mme Prévert lui enseignât aussi cette matière...

Alors faut-il violer la BCNF ?

On peut effectivement préférer ne pas décomposer la relvar EMP et se limiter au respect de la 3NF, mais en contrepartie il faudra prévoir une contrainte garantissant le respect de la dépendance fonctionnelle DF2, comme quoi un professeur n'enseigne qu'une matière.

Dans le style de Tutorial D :

```
VAR EMP BASE RELATION {Etudiant CHAR, Matiere CHAR, Professeur CHAR}
    KEY {Etudiant, Matiere};

CONSTRAINT EMPDF2
    FORALL x, y ∈ EMP (x.Professeur = y.Professeur ⇒ x.Matiere = y.Matiere) ;

(Quels que soient les tuples x et y de la relvar EMP, si x.Professeur = y.Professeur alors x.Matiere = y.Matiere)
```

Version SQL :

```
CREATE TABLE EMP (
    Etudiant      CHAR(16) NOT NULL
,   Matiere      CHAR(16) NOT NULL
,   Professeur   CHAR(16) NOT NULL
,   PRIMARY KEY (Etudiant, Matiere)) ;

CREATE ASSERTION EMPDF2 CHECK
    (NOT EXISTS (
        SELECT x.Matiere
        FROM   EMP AS x, EMP AS y
        WHERE  x.Professeur = y.Professeur
            AND x.Matiere <> y.Matiere)) ;
```

Rappel : En situation réelle, on utilise bien sûr des valeurs non significatives et invariantes pour les attributs composant les clés primaires (cf. paragraphe 3.2.6).

A défaut, à la place de l'assertion SQL, on peut utiliser une vue munie d'une clause WITH CHECK OPTION :

```
CREATE VIEW EMP_BIS (Etudiant, Matiere, Professeur) AS
    SELECT  x.Etudiant, x.Matiere, x.Professeur
    FROM    EMP AS x
    WHERE NOT EXISTS (
        SELECT *
        FROM   EMP AS y
        WHERE  x.Professeur = y.Professeur
            AND x.Matiere <> y.Matiere
        )
    WITH CHECK OPTION ;
```

Mais il faudra interdire l'accès à la table EMP pour les opérations de mise à jour et n'autoriser en l'occurrence que l'utilisation de la vue, système manquant un peu de souplesse. Quoi qu'il en soit, on peut aussi se rabattre sur un trigger SQL pour garantir la dépendance fonctionnelle DF2.

Exemple avec SQL Server 2005 :

```
CREATE TRIGGER EMPDF2Insert ON EMP INSTEAD OF INSERT AS
    SELECT x.Matiere
    FROM   Inserted x, EMP y
    WHERE  x.Professeur = y.Professeur
        AND x.Matiere <> y.Matiere ;
If @@Rowcount > 0
    Begin
        Raiserror ('Viol BCNF !',16,1)
    Return
    End
INSERT INTO EMP SELECT DISTINCT Etudiant, Matiere, Professeur
    FROM   Inserted ;
GO
```

Et bien entendu, il faudra prévoir un trigger pour contrôler les nouvelles valeurs prises par les attributs Matiere et Professeur lors des opérations d'**update**.

Concernant les opérations d'INSERT, on peut désormais violer la BCNF, puisque l'on sait faire respecter la DF {Professeur} → {Matiere}.



Pour le reste, il y a quand même un hic, car si Nanard n'étudie pas le grec mais le sanscrit, remplacer le tuple <"Nanard", "Grec", "Mlle Martin"> par le tuple <"Nanard", "sanscrit", "M. Antoun"> a pour effet que l'**on perd l'information** selon laquelle Mlle Martin enseigne le grec, si Nanard était son seul élève dans cette matière. Il sera donc prudent de commencer par le commencement, c'est-à-dire produire un MCD (ou un diagramme de classes) dans lequel figurent les entités-types Professeur, Matiere, Etudiant, une association-type entre Professeur et Matiere, une association-type ternaire, à savoir EMP, entre les trois entités-types et une CIF

(contrainte d'intégrité fonctionnelle) entre EMP et Professeur. Une fois cette tâche accomplie, alors seulement on pourra dériver le MCD : il s'agit là du processus normal de conception d'une base de données.

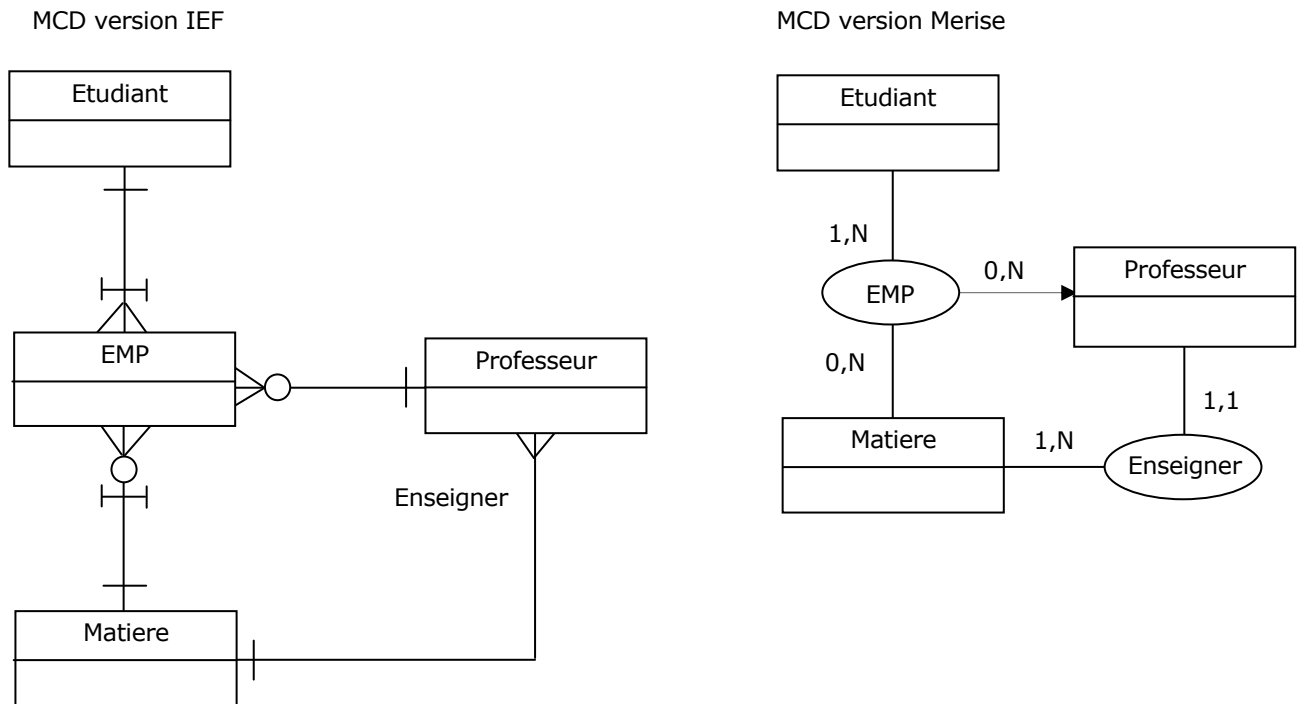


Figure 3.11 - Relvar EMP - Vision conceptuelle

En redescendant au niveau relationnel, on devra s'occuper des pièges qui seront inmanquablement tendus. Par exemple, si le corps de la relvar EMP contient le tuple <"Nanard", "Grec", "Mlle Martin">, après tout la matière enseignée par Mlle Martin peut très bien être le solfège... On part pour un trigger supplémentaire (ou une « surclé étrangère » inavouable entre EMP et Professeur...) pour s'assurer que la matière enseignée par un professeur aux élèves est bien celle que ce professeur est réputé enseigner.

Alors faut-il respecter la BCNF ?

En réaction à ce qui précède, normaliser et mettre en oeuvre les relvars PM {Professeur, Matière} et PE {Professeur, Etudiant} semble s'imposer, avec toutefois un contrôle à assurer pour la dépendance fonctionnelle DF1 {Etudiant, Matière} → {Professeur} puisque celle-ci est perdue après projection. Par contre, sémantiquement parlant, le MCD correspondant ne ressemble pas à grand-chose, et respecter la BCNF n'est finalement peut-être pas ici ce qu'il y a de mieux...

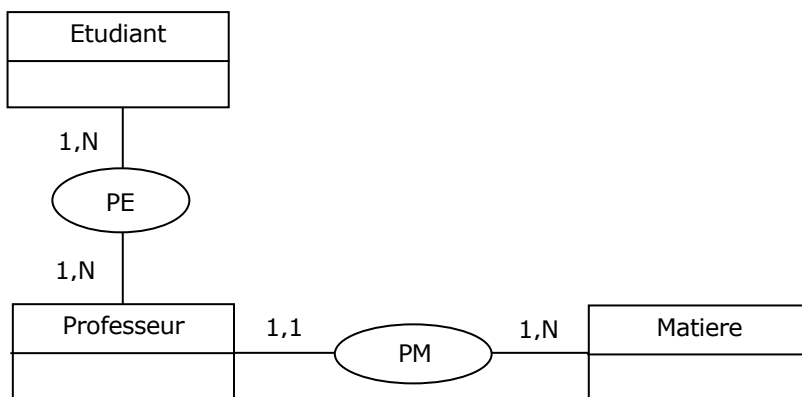


Figure 3.12 - Relvar EMP - Vision conceptuelle, variante

On est forcé d'en convenir. En tout cas, le code correspondant est le suivant :

Dans le style de Tutorial D :

```
VAR PM BASE RELATION {Professeur CHAR, Matiere CHAR}
    KEY {Professeur}

VAR PE BASE RELATION {Professeur CHAR, Etudiant CHAR}
    KEY {Professeur, Etudiant}
    FOREIGN KEY {Professeur} REFERENCES PM ;

CONSTRAINT EMPDF1
    FORALL x, y ∈ (PE JOIN PM) (x.Etudiant = y.Etudiant AND x.Matiere = y.Matiere
    ⇒ x.Professeur = y.Professeur) ;
```

Version SQL :

```
CREATE TABLE PM (
    Professeur    CHAR(16) NOT NULL
    , Matiere      CHAR(16) NOT NULL
    , PRIMARY KEY (Professeur)) ;

CREATE TABLE PE (
    Professeur    CHAR(16) NOT NULL
    , Etudiant     CHAR(16) NOT NULL
    , PRIMARY KEY (Professeur, Etudiant)
    , FOREIGN KEY (Professeur) REFERENCES PM) ;
GO
CREATE ASSERTION EMPDF1 CHECK
    (NOT EXISTS (
        SELECT X.Etudiant, X.Matiere, X.Professeur
        FROM   (SELECT PE.Etudiant, PM.Matiere, PM.Professeur
        FROM     PE INNER JOIN PM
                ON PE.Professeur = PM.Professeur) AS X
        INNER JOIN
        (SELECT PE.Etudiant, PM.Matiere, PM.Professeur
        FROM     PE INNER JOIN PM
                ON PE.Professeur = PM.Professeur) AS Y
        ON X.Etudiant = Y.Etudiant AND X.Matiere = Y.Matiere
        WHERE    X.Professeur <> Y.Professeur
    )) ;
```

Etc.

3.8. Retour sur la dénormalisation *a priori*

Le thème de la dénormalisation a été abordé au paragraphe 1.6. Voici comment en termes choisis on propage un mythe sans manifestement être allé vérifier comment se passent concrètement les choses. Je cite à nouveau le redoutable [RoMo 1989] qui écrit à la page 198 de l'ouvrage, sous le titre « Optimisation du MLD relationnel » :

« Une optimisation structurelle principale peut être effectuée au niveau logique, il s'agit de la création d'une certaine **redondance** en vue de diminuer le nombre d'actions élémentaires nécessaires à la satisfaction d'une requête en diminuant notamment le nombre de jointures entre tuples différents... »

Et l'auteur fournit un exemple de MLD dans lequel une table initialement en BCNF (en fait 5NF) viole désormais la 3NF, suite à injection d'une DF transitive (cf. paragraphe 3.5). Un avion (table Avion) est d'un certain type (table TypeAvion, porteuse d'un attribut NombreDePlaces) : la recommandation est de dupliquer l'attribut NombreDePlaces dans la table Avion. Cela part d'une bonne intention, l'auteur évoquant la possibilité « qu'il soit souvent demandé quelle est la capacité d'un avion ». Mais il aurait dû parler des effets secondaires et insister sur les précautions élémentaires à prendre avant de nous laisser nous embarquer dans ce genre de galère : prototyper et quantifier les performances, mettre en œuvre une contrainte sous-traitée au SGBD pour garantir en toute sécurité la cohérence des valeurs prises par l'attribut NombreDePlaces dans les deux tables concernées... Bref, l'imprudent aurait surtout dû

conclure que les concepteurs et les développeurs doivent s'en remettre aux DBA, lesquels sont outillés pour juger objectivement de l'(in)efficacité de la dénormalisation : ficher une zoubia pas possible pour se « satisfaire » d'un gain qui sera peut-être de l'ordre de la milliseconde ?

L'enfer est pavé de bonnes intentions. Allons-y d'un contre-exemple.

Dans une banque d'une ville méridionale où il fait bon vivre. On me soumet le problème suivant : Les chèques des clients sont regroupés en remises, lesquelles sont à leur tour regroupées en lots. Tous les chèques datés du tant doivent être supprimés de la table des chèques. Manque de chance, l'attribut de type Date utilisé pour l'opération de restriction ne figure pas dans la table Cheque, mais dans la table « grand-parente » Lot (attribut LotDate).

Ce qui est embêtant dans cette affaire, c'est que le traitement (hebdomadaire) dure près de 9 heures, pour une table de 2 millions de chèques (d'accord, ça se passait en 1993, mais quand même...) A l'unanimité moins une voix, celle de votre serviteur, le verdict tomba : « **C'est la faute à la 3NF**, il faut dé-nor-ma-lis-er ! »

Structure des tables :

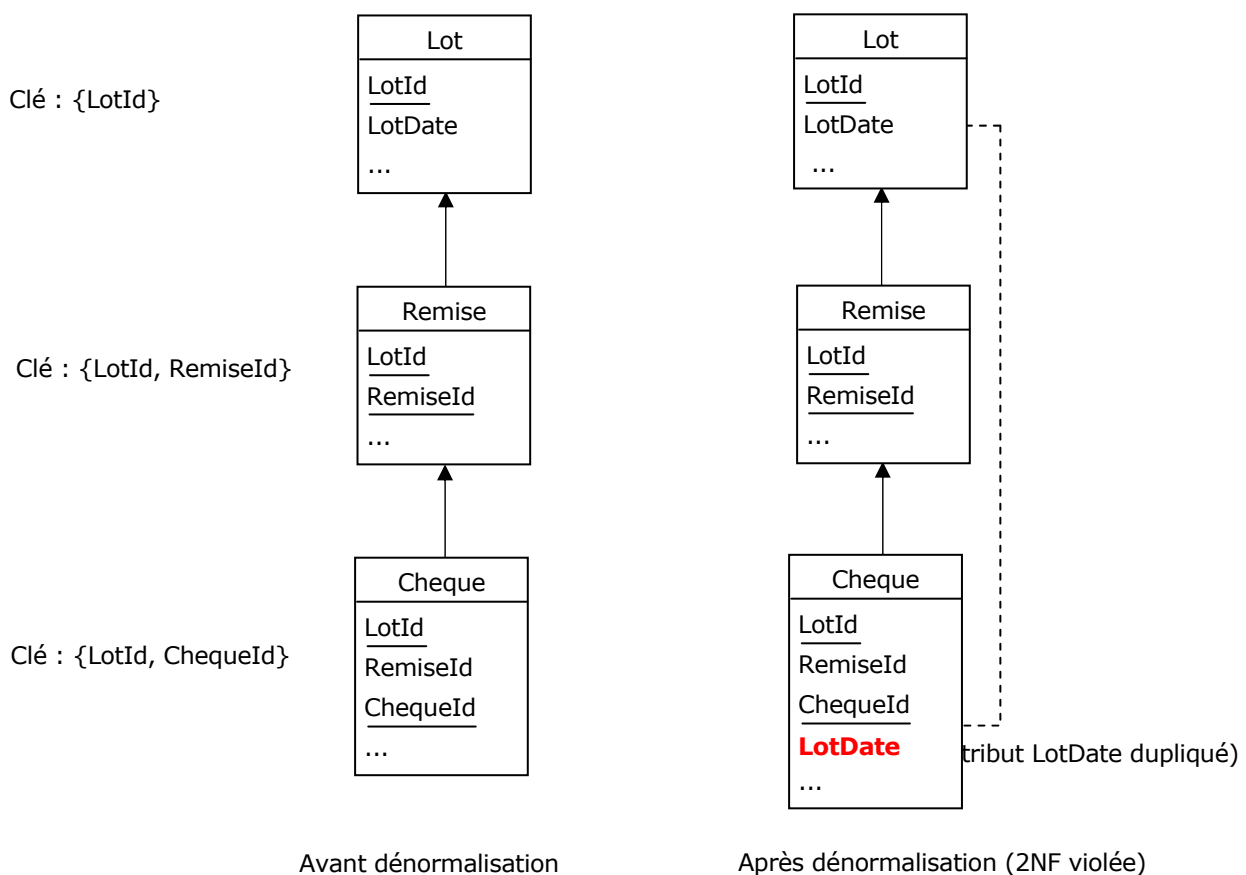


Figure 3.13 - Dénormalisation de la table des chèques

Requête de suppression des chèques :

Avant dénormalisation :

```
DELETE FROM Cheque WHERE Exists
(SELECT *
FROM Lot
WHERE Cheque.LotId = Lot.LotId
AND Lot.LotDate = d1) ;
```

Après dénormalisation :

```
DELETE FROM Cheque WHERE LotDate = d1 ;
```

Dénormaliser est certes bien tentant, mais avant que l'on ne commette le viol (en fait celui de la 2NF dans cet exemple), avant restructuration de la table Cheque, avant que ne soit aménagé le programme de suppression des chèques, etc., j'ai demandé que fut mesuré le coût du **seul** SELECT : celui-ci était de l'ordre de deux minutes. Stupeur des accusateurs ! La jointure, cette présumée coupable était lavée de tous les soupçons (son surcoût était infime, de l'ordre de quelques secondes). En fait, la table Cheque était lourdement lestée de cinq index que le SGBD (à savoir DB2) devait mettre à jour eux aussi au cours de l'opération (dont un du reste ne servait strictement à rien). La solution était donc évidente : ne pas dénormaliser, et avant d'exécuter le traitement de suppression des chèques, commencer par « dropper » les index (sauf l'index cluster qui s'avérerait peu pénalisant, par contraste avec les autres index qui étaient tous à l'origine de très nombreuses lectures/écritures de pages physiques, sans compter la journalisation), puis une fois l'opération de suppression proprement dite terminée, recréer les index. Pour une fenêtre batch autorisée de 1h30, le temps de traitement, y-compris une réorganisation de la table et la création des index, n'excéda plus une quinzaine de minutes, et l'irréparable ne fut pas commis...

Moralité : avant de dire et (faire) faire des bêtises, et se retrouver Gros-Jean comme devant, en ayant gagné une poignée de secondes sur 9 heures après avoir dénormalisé, on vérifie concrètement et on prouve. Bon d'accord, j'ai pris un exemple édifiant, mais j'en ai bien d'autres.

Mais attention, rien n'est jamais acquis...

Trois ans plus tard, le même problème se pose, mais cette fois-ci dans une banque en Normandie. Fort de mon expérience méridionale, je prépare sereinement un travail de prototypage, dans les règles de l'art. Avant de faire le ménage dans la table des chèques, allons-y pour « dropper » les index qui peuvent l'être. Ceci fait, mesurons la durée de l'opération de suppression. Surprise ! Vérité en deçà des Alpilles, erreur au delà... Il se trouve que de la table des chèques dépendent d'autres tables, et que le pourcentage de chèques à supprimer est cette fois-ci beaucoup plus important, ce qui fait que l'ensemble des pages des table spaces et index spaces (fichiers DB2) est à mettre à jour, et dans ces conditions la performance attendue n'est pas au rendez-vous. Mais si on échoue par la face Nord, reste la face Sud : plutôt que supprimer, on crée. En l'occurrence, cela consista à télécharger seulement les lignes à ne pas supprimer puis à les recharger dans la table en cause et dans ses tables « filles ». Ouf... Quoi qu'il en soit, une fois de plus, la normalisation était bien sûr totalement étrangère au problème de performance, mais elle aurait bien trouvé des accusateurs, ainsi qu'un folliculaire pour entretenir le mythe dans la presse du cœur informatique.

3.9. Une relvar respectant la BCNF peut-elle violer la 3NF ?

Une relvar en BCNF est nécessairement en 3NF, mais si l'on utilise la définition non coddienne figurant au paragraphe 3.6, alors on peut arriver à une contradiction, à savoir qu'une relvar respectant la BCNF peut violer la 3NF. Rappelons cette définition incomplète et peccamineuse :

Une relation R est en troisième forme normale si et seulement si :

1. Elle est en deuxième forme normale ;
2. Tout attribut n'appartenant pas à la **clé primaire** de R n'en dépend pas transitivement.

La contradiction peut être montrée à l'aide d'exemples très simples, tels que celui-ci :

Soit une relvar R {A, B, C} où A, B et C sont des attributs, et les dépendances fonctionnelles :

DF1 : {A} → {B}

DF2 : {A} → {C}

DF3 : {B} → {A}

DF4 : {B} → {C}

Ce sont les seules DF non triviales et leurs déterminants {A} et {B} sont clés candidates : R respecte la BCNF. Mais si l'on choisit par exemple {A} pour clé primaire, puisque {A} détermine {C} transitivement, la 3NF incomplète est violée. En effet :

$$\{A\} \rightarrow \{B\} \text{ et } \{B\} \rightarrow \{C\} \Rightarrow \{A\} \rightarrow \{C\}$$

Parce qu'incomplète, la définition de la 3NF est fausse. Mais pour varier les plaisirs et l'occasion faisant le larron, profitons d'un exercice récapitulatif pour mettre à nouveau la contradiction en évidence. Intéressons-nous pour cela aux courses hippiques (courses de plat, galop). Ainsi, le prix de l'Arc de Triomphe, édition 2009, doté de 4 000 000 d'euros a eu lieu le 4 octobre 2009 et a vu la victoire de Sea The Stars, monté par Michael Kinane. L'édition 2010, dotée à nouveau de 4 000 000 d'euros, a vu la victoire de Workforce, monté par Ryan-Lee Moore, etc.

Définissons une table des courses et une table des partants :

COURSE	<u>CourseId</u>	Prix	DateCourse	Allocation
	cs01	Arc de Triomphe	2009-10-04	4 000 000
	cs02	Arc de Triomphe	2010-10-03	4 000 000
	cs03	Prix de Bréhal	2009-12-10	17 000

Figure 3.14 - Extrait de la table des courses hippiques

PARTANT	<u>CourseId</u>	<u>ChevalId</u>	Numero	JockeyId	Place	Poids	EntraeneurId
	cs01	ch01	18	j01	1	56	En01
	cs01	ch02	1	j02	2	59	En02
	cs01	ch03	16	j03	3	56	En03
	cs02	ch04	13	j04	1	56	En04
	cs02	ch05	7	j05	2	59	En05
	cs02	ch06	20	j06	3	54	En06
	cs02	ch02	1	j07	NC	59	En02
	cs03	ch07	5	j08	1	58	En08
	cs03	ch08	11	j09	10	56	En09
	cs03	ch09	7	j10	NC	57	En09

Figure 3.15 - Extrait de la table des partants

Dans la course cs01, Le cheval ch01, porteur du numéro 18, était monté par le jockey j01, il a fini à la 1re place, il portait 56 kilos et était entraîné par l'entraîneur En01, etc. A noter que dans la course cs02, le cheval ch02 a terminé non classé, alors que l'année d'avant, dans le même Prix (Arc de Triomphe), il avait fini 2e.

Intéressons nous aux dépendances fonctionnelles de la relvar PARTANT. Dans une course donnée, un cheval ne porte qu'un numéro, il n'est monté que par un seul jockey, il termine exactement à une place, il porte exactement un poids et n'a qu'un entraîneur. On peut donc inférer la DF :

DF01 : {CourseId, ChevalId} → {Numero, JockeyId, Place, Poids, EntraeneurId}

En appliquant à cette DF l'axiome d'augmentation (cf. Annexe E.2), on infère la DF :

DF02 : {CourseId, ChevalId} → {CourseId, ChevalId, Numero, JockeyId, Place, Poids, EntraeneurId}

En conséquence quoi le déterminant {CourseId, ChevalId} de cette DF est clé candidate de la relvar.

De la même façon, dans une course donnée, un jockey ne monte qu'un cheval :

DF03 : {CourseId, JockeyId} → {ChevalId}

En appliquant à son tour à cette DF l'axiome d'augmentation, on infère la DF :

DF04 : {CourseId, JockeyId} → {CourseId, ChevalId}

En ce qui concerne le numéro porté par le cheval :

DF05 : {CourseId, Numero} → {ChevalId}

En appliquant là encore à cette DF l'axiome d'augmentation, on infère la DF :

DF06 : $\{CourseId, Numero\} \rightarrow \{CourseId, ChevalId\}$

Et en appliquant l'axiome de transitivité, on infère les DF :

DF07 : $\{CourseId, JockeyId\} \rightarrow \{CourseId, ChevalId, Numero, JockeyId, Place, Poids, EntraîneurId\}$

DF08 : $\{CourseId, Numero\} \rightarrow \{CourseId, ChevalId, Numero, JockeyId, Place, Poids, EntraîneurId\}$

Ainsi, les paires $\{CourseId, JockeyId\}$ et $\{CourseId, Numero\}$ sont aussi clés candidates. Par ailleurs, il n'existe pas d'autres DF non triviales que celles que l'on a énumérées (deux ou plusieurs chevaux peuvent partager la même place dans la même course : ex-æquo, non classé, non partant ; un entraîneur peut avoir deux ou plusieurs chevaux engagés dans la même course, etc.)

=> Le déterminant de chaque DF non triviale est clé candidate de la relvar : celle-ci respecte la BCNF.

Maintenant, quelle que soit la clé candidate que l'on retient pour être clé primaire, par exemple la paire $\{CourseId, ChevalId\}$, en l'occurrence on vérifie :

DF09 : $\{CourseId, ChevalId\} \rightarrow \{CourseId, JockeyId\}$

Mais aussi :

DF10 : $\{CourseId, JockeyId\} \rightarrow \{Place, Poids, EntraîneurId\}$

Donc par transitivité :

DF11 : $\{CourseId, ChevalId\} \rightarrow \{Place, Poids, EntraîneurId\}$

Autrement dit, alors que la relvar respecte la BCNF, selon la définition incomplète et peccamineuse de la 3NF, celle-ci est violée, en contradiction avec la règle selon laquelle toute relvar respectant la BCNF respecte nécessairement la 3NF :



La définition incomplète et peccamineuse est donc fausse et bonne pour la poubelle.

S'il fallait la suivre, par application du théorème de Heath, la relvar PARTANT donnerait lieu à un ensemble de relvars tel que celui-ci :

PARTANT01 $\{CourseId, ChevalId, Place, Poids, EntraîneurId\}$

PARTANT02 $\{CourseId, ChevalId, Numero\}$

PARTANT03 $\{CourseId, ChevalId, JockeyId\}$

Ces structures désormais « 3NF » peuvent être intéressantes, mais elles relèvent plutôt de l'optimisation de la relvar PARTANT (cf. paragraphe 1.7).

Bref, Pour conclure et marquer un temps de repos bien mérité, citons Pierre Dac (1893-1975) :

« Gloire à ceux qui ont forgé silencieusement mais efficacement le fier levain qui, demain ou après-demain au plus tard, fera germer le grain fécond du ciment victorieux, au sein duquel, enfin, sera ficelée, entre les deux mamelles de l'harmonie universelle, la prestigieuse clef de voûte qui ouvrira à deux battants la porte cochère d'un avenir meilleur sur le péristyle d'un monde nouveau ! »

Fermez le ban.

N.B. J'avoue ne pas savoir si la prestigieuse clef de voûte dont parle notre cher Dac est candidate, primaire, alternative, étrangère ou autre, libre à chacun de se faire son opinion. En tout cas, gloire à Codd, Date, Darwen, et tous ceux qui ont forgé le fier levain relationnel !

4. Quatrième forme normale

4.1. Au-delà de la BCNF

Bien que la normalisation en BCNF permette d'éliminer des wagons de redondances parasitant les relations, avec en prime la disparition dans la foulée d'un cortège de problèmes de mise à jour inhérents, il reste malgré tout des cas de redondance face auxquels la BCNF se révèle impuissante. En 1976-1977, Claude Delobel, Carlo Zaniolo et Ronald Fagin découvrirent chacun de leur côté un nouveau type de dépendance, à savoir la **dépendance multivaluée** (DMV), qui fut à la base de nouvelles techniques pour décomposer des relations réputées jusque-là « incassables » et éliminer ainsi des redondances et anomalies rebelles. En 1977 Fagin donna l'énoncé de la quatrième forme normale (4NF), nous permettant de nous assurer que les tables infectées par ces redondances auront été assainies après normalisation (cf. [Fagin 1977]). Mais avant de traiter des dépendances multivaluées et de la 4NF, faisons d'abord un détour du côté de la modélisation des données.

4.2. Observations à propos de la jointure naturelle

Intéressons-nous aux ingénieurs spécialistes en bases de données de la SARL *Les Prévoyants de l'Avenir des Bases de Données* (LPABDD, filiale de DVP), plus précisément en ce qui concerne les projets auxquels ils ont été affectés au fil du temps, outre leurs compétences concernant les SGBD.

Les règles de gestion qui nous intéressent ici sont les suivantes :

- (RG1) Un ingénieur est compétent pour au moins un SGBD.
- (RG2) Un SGBD peut être (plus ou moins bien...) maîtrisé par au moins un ingénieur.
- (RG3) Un ingénieur a été affecté à au moins un projet.
- (RG4) Un projet a pu nécessiter l'affectation d'au moins un ingénieur.
- (RG5) On doit connaître les différentes périodes pendant lesquelles un ingénieur donné est intervenu sur un projet donné (un ingénieur a pu être affecté à plus d'un projet pendant la même période). Pour simplifier, on dira qu'une période correspond à une année. Par ailleurs, on ne cherche pas à respecter les impératifs LDD (cf. paragraphe 6, « Sixième forme normale »).

Exemples :

Albert maîtrise IMS et DB2. Il a été affecté au projet Biglotron en 2003, 2004 et 2006. Il a aussi été affecté au projet Slalom en 2003 et 2005.

Bernard maîtrise BS12 et DB2. Il a été affecté au projet Biglotron en 2004 et 2005.

Charles maîtrise Oracle. Il a été affecté au projet TantkYoradla en 2007 et 2008.

Sous forme prédicative (un prédicat dyadique de la forme F_{xy} et un prédicat triadique de la forme G_{xyz}) :

- (P1) F_{xy} : L'ingénieur *IngId* maîtrise le SGBD *SgbdId*
- (P2) G_{xyz} : L'ingénieur *IngId* a été affecté au projet *ProjId* au cours de l'année *Annee*

Une représentation graphique :

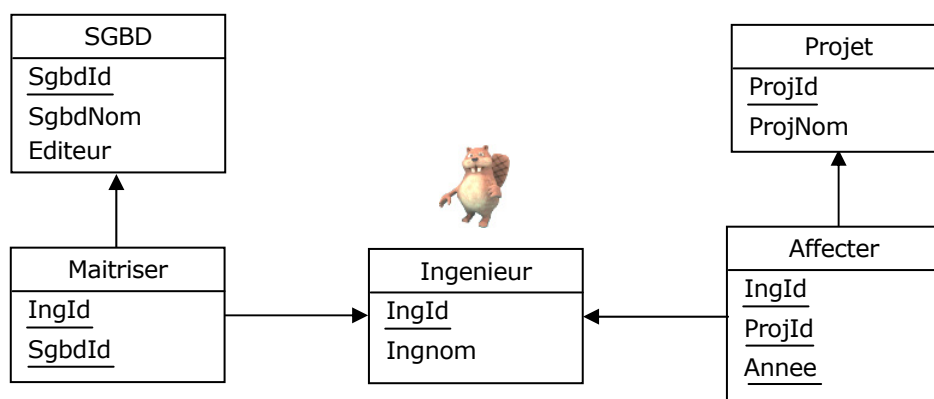


Figure 4.1 - Les ingénieurs, leurs SGBD, leurs projets

Un instantané (relations à un moment donné, ou contenu des tables au sens SQL). Les clés sont soulignées :

Ingenieur	<u>IngId</u>	IngNom	SGBD	<u>SgbdId</u>	SgbdNom	Projet	<u>ProjId</u>	ProjNom
	i1	Albert		s1	IMS		p1	Biglotron
	i2	Bernard		s2	BS12		p2	Slalom
	i3	Charles		s3	Oracle		p3	Flugdug
				s4	DB2		p4	TantkYoradla
				s5	IDMS		p5	YaduPour

Maitriser	<u>IngId</u>	<u>SgbdId</u>	Affecter	<u>IngId</u>	<u>ProjId</u>	<u>Annee</u>
	i1	s1		i1	p1	2003
	i1	s4		i1	p1	2004
	i2	s2		i1	p1	2006
	i2	s4		i1	p2	2003
	i3	s3		i1	p2	2005
				i2	p1	2004
				i2	p1	2005
				i3	p4	2007
				i3	p4	2008

Figure 4.2 - Les ingénieurs, leurs SGBD, leurs projets (instantané)

On vérifie sans difficulté que la BCNF est partout vérifiée. Supposons maintenant que l'on mette en oeuvre une relvar virtuelle (vue), appelons-la ISPV, jointure naturelle des relvars Affecter et Maitriser. La consultation de la relvar donne lieu au résultat suivant :

ISPV	<u>IngId</u>	<u>SgbdId</u>	<u>ProjId</u>	<u>Annee</u>
	i1	s1	p1	2003
	i1	s1	p1	2004
	i1	s1	p1	2006
	i1	s1	p2	2003
	i1	s1	p2	2005
	i1	s4	p1	2003
	i1	s4	p1	2004
	i1	s4	p1	2006
	i1	s4	p2	2003
	i1	s4	p2	2005
	i2	s2	p1	2004
	i2	s2	p1	2005
	i2	s4	p1	2004
	i2	s4	p1	2005
	i3	s3	p4	2007
	i3	s3	p4	2008

N.B.

En Tutorial D, la définition de la relvar est la suivante :

```
VAR ISPV VIRTUAL (Maitriser JOIN Affecter)
KEY {IngId, SgbdId, ProjId, Annee} ;
```

En SQL :

```
CREATE VIEW ISPV (IngId, SgbdId, ProjId, Annee)
AS SELECT a.IngId, a.SgbdId, b.ProjId, b.Annee
FROM Maitriser AS a INNER JOIN Affecter AS b
ON a.IngId = b.IngId ;
```

Figure 4.3 - ISPV : Jointure naturelle des tables Maitriser et Affecter

Notons en passant que ce que l'on a pour habitude d'appeler *vue* est en réalité une relvar, virtuelle certes, mais relvar quand même, d'où la présence de la clause KEY accompagnant la définition de la relvar (en Tutorial D tout du moins).

La structure et le contenu de relvar ISPV méritent quelques observations.

Observations

- **Clé de la relvar ISPV**

Le quadruplet {IngId, SgbdId, ProjId, Annee}, c'est-à-dire l'ensemble des attributs de la relvar ISPV en constitue la clé. En effet, aucun triplet d'attributs ne vérifie la règle d'unicité des clés candidates. Par exemple :

Pour le triplet {IngId, SgbdId, ProjId} et pour le tuple <i1, s1, p1>, on a 3 années distinctes : 2003, 2004, 2006.

Pour le triplet {IngId, SgbdId, Annee} et pour le tuple <i1, s1, 2003>, on a 2 projets distincts : p1, p2.

Pour le triplet {IngId, ProjId, Annee} et pour le tuple <i1, p1, 2004>, on a 2 SGBD distincts : s1, s4.

Pour le triplet {SgbdId, ProjId, Annee} et pour le tuple <s4, p1, 2004>, on a 2 ingénieurs distincts : i1, i2.

- **La relvar ISPV respecte la BCNF**

En effet, toutes les dépendances fonctionnelles qui lui sont associées sont triviales.

- **Piège de la connexion (connection trap)**

Le prédicat associé à ISPV est la conséquence d'une jointure naturelle, il est donc la conjonction des prédicats P1 et P2 (en relationnel : P1 AND P2) : L'ingénieur *IngId* maîtrise le SGBD *SgbdId* **ET** il a été affecté au projet *ProjId* au cours de l'année *Annee*. Le piège mortel serait de le confondre avec le prédicat tétradratique suivant :

(P3) H_{xyzw} : L'ingénieur *IngId* a utilisé le SGBD *SgbdId* dans le cadre du projet *ProjId* au cours de l'année *Annee*

Selon lequel on énoncerait des contre-vérités, du genre « Albert a utilisé IMS dans le cadre du projet Biglotron » (ce qu'Albert niera farouchement !) Toutefois, si le prédicat P3 devait exprimer une **règle de gestion** authentique, alors il faudrait appeler un chat un chat et mettre en oeuvre une relvar de base, telle que la relvar Utiliser ci-dessous, de prédicat P3, cette fois-ci **non dérivable**, donc sans piège :

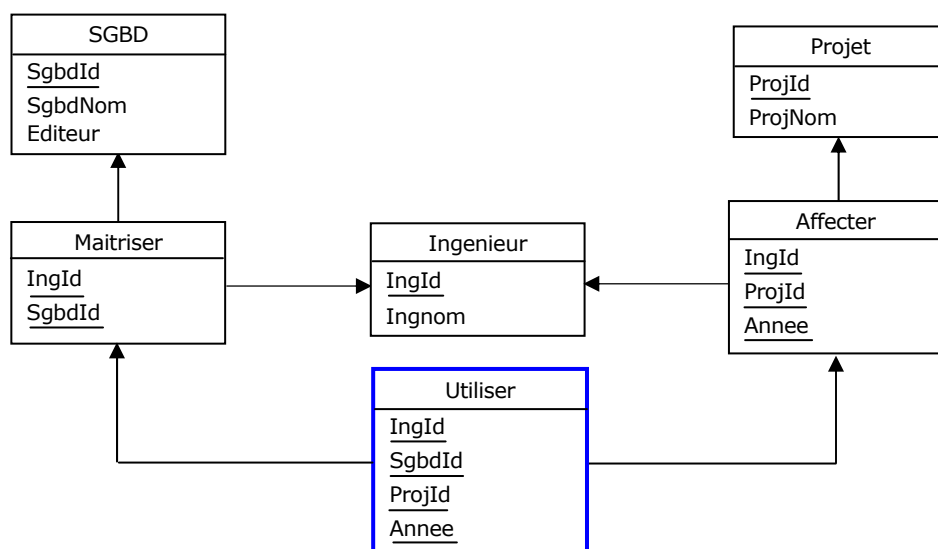


Figure 4.4 - Quels ingénieurs ont utilisé quels SGBD pour quels projets

Une fois la relvar Utiliser valorisée par qui de droit, on constaterait qu'Albert n'a effectivement pas utilisé IMS dans le cadre du projet Biglotron. Par contraste avec ISPV, la relvar Utiliser n'exprime que la réalité (hypothèse du monde clos) :

Utiliser	<u>IngId</u>	<u>SgbdId</u>	<u>ProjId</u>	<u>Annee</u>
	i1	s1	p2	2005
	i1	s4	p1	2003
	i1	s4	p1	2004
	i1	s4	p1	2006
	i1	s4	p2	2003
	i2	s2	p1	2004
	i2	s4	p1	2005
	i3	s3	p4	2007
	i3	s3	p4	2008

Figure 4.5 - Utilisation effective des SGBD dans le cadre des projets

• Redondances

Pour en revenir à la relvar ISPV, on voit qu'elle est particulièrement redondante. En effet, pour chaque ingénieur qui y figure, et pour chaque SGBD qu'il maîtrise, est systématiquement fourni l'ensemble de ses affectations aux projets, et de la même façon, pour chacune de ses affectations, la relvar fournit l'ensemble des SGBD qu'il maîtrise.

Cette gabegie est choquante eu égard au principe même de la normalisation dont un des principaux objets est quand même l'élimination de redondances non voulues. Pire, dans ISPV le mensonge est en embuscade et imaginons que, *horresco referens* ! lors de l'étape de conception de la base de données nous ayons fait d'ISPV une relvar de base, ou une table au sens SQL... Nous devons tout mettre en œuvre pour repérer et éradiquer ces redondances. A cette fin, nous devons savoir manier les outils nécessaires, à savoir les dépendances fonctionnelles bien sûr, mais aussi et surtout ceux qu'on appelle les dépendances multivaluées.

4.3. Dépendance multivaluée (DMV)

Considérons une relvar $R \{X, Y, Z\}$ dans laquelle X , Y et Z représentent des sous-ensembles d'attributs de l'en-tête H de R (avec $Z = H - X - Y$, c'est-à-dire l'ensemble des attributs appartenant à H mais ni à X ni à Y). Si pour chaque valeur prise par X , les valeurs prises par Y sont **indépendantes** des valeurs prises par Z , il existe alors ce qu'on appelle une dépendance multivaluée, notée $X \twoheadrightarrow Y$. (A signaler que X et Y ne sont pas nécessairement disjoints). Plus précisément :

Soit une relvar $R \{X, Y, Z\}$ où X , Y et Z représentent des sous-ensembles d'attributs de l'en-tête H de R (X et Y ne sont pas nécessairement disjoints). Étant donnés $\langle x, y, z \rangle$ et $\langle x, y', z' \rangle$ deux tuples appartenant à R , la dépendance multivaluée $X \twoheadrightarrow Y$ est vérifiée si et seulement si les tuples $\langle x, y', z \rangle$ et $\langle x, y, z' \rangle$ appartiennent aussi à R .

Illustrons ceci avec l'exemple de la relvar ISPV. Étant la jointure naturelle des relvars Maitriser et Affecter, indépendantes l'une de l'autre du fait des règles de gestion initiales, la relvar ISPV vérifie *ipso facto* les dépendances multivaluées (DMV) suivantes :

DMV1 : $\{IngId\} \twoheadrightarrow \{SgbdId\}$ (Ce qui se lit : $\{SgbdId\}$ est **multi-dépendant** de $\{IngId\}$, ou encore $\{IngId\}$ **multi-détermine** $\{SgbdId\}$) ;

DMV2 : $\{IngId\} \twoheadrightarrow \{ProjId, Annee\}$ ($\{ProjId, Annee\}$ est multi-dépendant de $\{IngId\}$, $\{IngId\}$ multi-détermine $\{ProjId, Annee\}$).

En effet, la jointure naturelle « Maitriser JOIN Affecter » donne lieu aux tuples (cas de l'ingénieur s1 en l'occurrence) :

$t1 = \langle i1, s1, p1, 2003 \rangle$ et $t2 = \langle i1, s4, p2, 2003 \rangle$

mais aussi aux tuples :

$t3 = \langle i1, s1, p2, 2003 \rangle$ et $t4 = \langle i1, s4, p1, 2003 \rangle$

De même, la jointure naturelle donne lieu aux tuples :

$t5 = \langle i1, s1, p1, 2004 \rangle$ et $t6 = \langle i1, s4, p1, 2006 \rangle$

mais aussi aux tuples :

$t7 = \langle i1, s1, p1, 2006 \rangle$ et $t8 = \langle i1, s4, p1, 2004 \rangle$

Etc., etc., etc.

De la sorte, alors que quel que soit l'ingénieur, les SGBD qu'il maîtrise n'ont rien à voir avec ses affectations à des projets, via ISPV on arrive à rendre mutuellement **dépendantes** les compétences et les affectations de cet ingénieur, et le piège de la connexion se referme comme on l'a vu. Supposons maintenant que le système évolue et que l'on ait à mettre œuvre la règle de gestion, selon laquelle on doit savoir *quels SGBD ont été mis en œuvre par quels ingénieurs dans le cadre de quels projets* : cette fois-ci, il n'y a plus indépendance entre les compétences et les affectations et il devient nécessaire de mettre en œuvre la relvar de base Utiliser (cf. Figure 4.4). A défaut, *horresco referens* à nouveau..., si l'on transformait ISPV elle-même en relvar de base (ou en table SQL), sa validité en serait compromise, et à moins de parfaitement maîtriser le sujet des dépendances multivaluées et de prendre les précautions qui s'imposent, qui se soucierait d'assurer la pérennité de DMV1 et DMV2 lors des opérations de mise à jour portant sur cette relvar ?

4.4. Décomposition sans perte de données. Premier théorème de Fagin

La définition de la dépendance multivaluée sous-entend que si l'on effectue la projection de la relvar ISPV sur les attributs {IngId, SgbdId} d'une part et {IngId, ProjId, Annee} d'autre part, alors cette décomposition est sans perte de données, c'est-à-dire que la jointure naturelle de ces deux projections redonne très exactement ISPV. De fait, cette relation entre dépendance multivaluée et décomposition sans perte existe bien et fait l'objet du théorème suivant, dû à Ronald Fagin (dans le cadre de cet article, nous l'appellerons le Premier théorème de Fagin, cf. [Fagin 1977]) :

La dépendance multivaluée $X \twoheadrightarrow Y$ est vérifiée par la relvar $R \{X, Y, Z\}$ si et seulement si R est égale à la jointure de ses projections $R_1 \{X, Y\}$ et $R_2 \{X, Z\}$.

(A noter qu'à partir de la jointure de $R_1 \{X, Y\}$ et $R_2 \{X, Z\}$ on infère la DMV $X \twoheadrightarrow Y$, alors qu'avec le théorème de Heath (cf. paragraphe 3.3.2), à partir de cette jointure on ne peut pas inférer la DF $X \rightarrow Y$ pour R , DF qui peut donc être perdue quand on normalise en BCNF alors qu'elle exprimerait une règle de gestion, cf. paragraphe 3.7).

On peut encore formuler ainsi le Premier théorème de Fagin ([Date 2004]) :

Soit la relvar $R \{X, Y, Z\}$ dans laquelle X, Y, Z sont des sous-ensembles d'attributs. R est égale à la jointure de ses projections sur $\{X, Y\}$ et $\{X, Z\}$ si et seulement si R vérifie les dépendances multivaluées $X \twoheadrightarrow Y$ et $X \twoheadrightarrow Z$.

Comme Y et Z jouent un rôle symétrique, on peut formuler la règle dite de **complémentation** :

La dépendance multivaluée $X \twoheadrightarrow Y$ est vérifiée par la relvar $R \{X, Y, Z\}$ si et seulement si $X \twoheadrightarrow Z$ l'est aussi.

Quant à la notation, au lieu d'écrire :

$$\{IngId\} \twoheadrightarrow \{SgbdId\}$$

$$\{IngId\} \twoheadrightarrow \{ProjId, Annee\}$$

On peut écrire de façon plus ramassée :

$$\{IngId\} \twoheadrightarrow \{SgbdId\} \mid \{ProjId, Annee\}$$

4.5. Dépendance fonctionnelle et dépendance multivaluée (règle de réplication)

Reprenons la définition de la dépendance multivaluée :

Soit une relvar $R \{X, Y, Z\}$ où X, Y et Z représentent des sous-ensembles d'attributs de l'en-tête H de R (X et Y ne sont pas nécessairement disjoints). Étant donnés $\langle x, y, z \rangle$ et $\langle x, y', z' \rangle$ deux tuples appartenant à R , la dépendance multivaluée $X \twoheadrightarrow Y$ est vérifiée si et seulement si les tuples $\langle x, y', z \rangle$ et $\langle x, y, z' \rangle$ appartiennent aussi à R .

Maintenant, si une relvar R vérifie la dépendance fonctionnelle $X \rightarrow Y$, alors pour tous les tuples $\langle x, y, z \rangle$ et $\langle x, y', z' \rangle$ de R , on a $y = y'$. Il s'ensuit que la condition d'existence des tuples $\langle x, y', z \rangle$ et $\langle x, y, z' \rangle$ est *de facto* vérifiée, en vertu de quoi la dépendance multivaluée $X \twoheadrightarrow Y$ est vérifiée elle aussi, ce qui donne lieu à la règle dite de **réplication** : Si $X \rightarrow Y$ alors $X \twoheadrightarrow Y$.

4.6. Dépendance multivaluée triviale

Le concept de dépendance multivaluée triviale (c'est-à-dire toujours vérifiée) est à définir, car elle intervient dans la définition de la 4NF.

Soit X et Y deux sous-ensembles d'attributs de l'en-tête H d'une relvar R . La dépendance multivaluée $X \twoheadrightarrow Y$ est triviale si et seulement si :

- (a) Y est un sous-ensemble de X ;
- (b) Ou encore si H est l'union de X et de Y (au sens de la théorie des ensembles). Par exemple, si H est composé des trois attributs A, B et C , et si les DMV $\{A\} \twoheadrightarrow \{B\}$ et $\{A\} \twoheadrightarrow \{C\}$ sont vérifiées, elles ne sont pas triviales contrairement à la DMV $\{A, B\} \twoheadrightarrow \{C\}$. A noter que si la DMV $\{A\} \twoheadrightarrow \{B\} \mid \{C\}$ est triviale, alors $\{B\}$ ou $\{C\}$ est l'ensemble vide $\{\}$.

4.7. Quatrième forme normale (4NF)

On peut maintenant donner l'énoncé de la quatrième forme normale (4NF), dû à Ronald Fagin ([Fagin 1977]) :

A relation schema R^* is in fourth normal form (4NF) if, whenever a nontrivial multivalued dependency $X \twoheadrightarrow Y$ holds for R^* , then so does the functional dependency $X \rightarrow A$ for every column name A of R^* .

(Une relvar d'en-tête R^* est en quatrième forme normale (4NF) si pour chaque dépendance multivaluée non triviale $X \twoheadrightarrow Y$ vérifiée par R^* , la dépendance fonctionnelle $X \rightarrow \{A\}$ est également vérifiée pour chaque attribut A de R^*).

Pour faire le parallèle avec la définition donnée de la BCNF par Zaniolo (cf. paragraphe 3.3.1), on peut encore écrire :

Soit R une relvar et X et Y des sous-ensembles d'attributs quelconques de l'en-tête H de R . R est en quatrième forme normale (4NF) si et seulement si, pour chaque dépendance multivaluée $X \twoheadrightarrow Y$ vérifiée pour R , au moins une des conditions suivantes est satisfaite :

- ✓ Cette dépendance multivaluée est triviale.
- ✓ La dépendance fonctionnelle $X \rightarrow \{A\}$ est vérifiée pour chaque attribut A de H (X est alors une surclé de R).

Illustration :

Considérons à nouveau l'instantané de la relvar ISPV (Figure 4.3), comportant comme on l'a vu les dépendances multivaluées non triviales (cf. paragraphe 4.3) :

DMV1 : $\{IngId\} \twoheadrightarrow \{SgbdId\}$
 DMV2 : $\{IngId\} \twoheadrightarrow \{ProjId, Annee\}$

Nonobstant les périls inhérents évoqués au paragraphe 4.3, imaginons que, toute honte bue, de cette relvar piège nous fassions une relvar de base (table en SQL), appelons-la ISP. Cette relvar vérifie la BCNF car les dépendances fonctionnelles sont toutes triviales. En revanche, la 4NF est violée. En effet, le multi-déterminant $\{IngId\}$ des deux DMV n'est pas surclé de la relvar puisque, comme on l'a vu précédemment avec ISPV, ISP n'a qu'une seule surclé, le quadruplet $\{IngId, SgbdId, ProjId, Annee\}$ (clé primaire au sens SQL). Mais en vertu du 1er théorème de Fagin, la relvar ISP est décomposable sans perte de données, elle n'a donc pas lieu d'exister. A cette occasion, on peut aussi se servir du théorème suivant, que l'on retrouve par exemple chez Ullman (cf. [Ullman 1982], « Theorem 7.2 ») :

- Si $\rho = (R1, R2)$ est une décomposition de R , et D un ensemble de dépendances fonctionnelles et multivaluées associé à R , alors ρ préserve le contenu de la base de données par rapport à D si et seulement si on vérifie $(R1 \cap R2) \twoheadrightarrow (R1 - R2)$ ou $(R1 \cap R2) \twoheadrightarrow (R2 - R1)$.

En posant $R1 = \{IngId, SgbdId\}$ et $R2 = \{IngId, ProjId, Annee\}$, $R1 \cap R2$ est égal à $\{IngId\}$ et $R1 - R2$ est égal à $\{SgbdId\}$. Comme la DMV $\{IngId\} \twoheadrightarrow \{SgbdId\}$ est donnée, on vérifie bien $(R1 \cap R2) \twoheadrightarrow (R1 - R2)$, donc la relvar ISP est décomposable sans perte de données, et l'on retrouve ainsi les relvars Maitriser et Affecter (cf. Figure 4.2).

Si la relvar ISP viole la 4NF mais est décomposable selon les relvars Maitriser et Affecter (voir toutefois les réserves faites au paragraphe 4.12), on observera pour leur part que ces deux dernières respectent la 4NF ; c'est vrai en ce qui concerne Maitriser $\{IngId, SgbdId\}$ car ses DMV sont toutes triviales, $\{IngId\} \twoheadrightarrow \{SgbdId\}$ et $\{SgbdId\} \twoheadrightarrow \{IngId\}$ en particulier.

A son tour, pour que la relvar Affecter ne vérifie pas la 4NF, il faudrait qu'il existe une des dépendances multivaluées non triviales suivantes :

DMV1 : $\{IngId\} \twoheadrightarrow \{ProjId\} \mid \{Annee\}$
 DMV2 : $\{ProjId\} \twoheadrightarrow \{IngId\} \mid \{Annee\}$
 DMV3 : $\{Annee\} \twoheadrightarrow \{IngId\} \mid \{ProjId\}$

Mais, conséquence des règles de gestion, ça n'est pas le cas (cf. Figure 4.2). Concernant DMV1, la présence des deux tuples :

$\langle i1, p1, 2004 \rangle$ et $\langle i1, p2, 2003 \rangle$ devrait entraîner celle du tuple $\langle i1, p2, 2004 \rangle$, lequel manque bien sûr à l'appel.

De la même façon, concernant DMV2, la présence des deux tuples :

<i1, p1, 2003> et <i2, p1, 2004> devrait entraîner la présence du tuple <i2, p1, 2003> qui lui aussi est absent.

Et, concernant DMV3, la présence des deux tuples :

<i1, p2, 2005> et <i2, p1, 2005> devrait entraîner la présence du tuple <i2, p2, 2005> qui lui aussi fait défaut.

Conclusion : la relvar Affecter vérifie la 4NF (et du même coup la BCNF, cf. paragraphe 4.14).

4.8. Un (sympathique) théorème de Date et Fagin concernant la 4NF

On doit à [Date et Fagin](#) le théorème suivant (théorème 5.2) :

Si une relvar R vérifie la BCNF et si elle est munie d'au moins une clé mono-attribut, alors elle vérifie la 4NF.

Ce théorème permet de se rendre compte qu'on peut facilement se faire piéger lors de la conception d'une base de données. Supposons en effet que le prédicat P3 : « L'ingénieur *IngId* a utilisé le SGBD *SgbdId* dans le cadre du projet *ProjId* au cours de l'année *Annee* » fasse l'objet d'une règle de gestion (cf. paragraphe 4.2), et qu'au lieu de la relvar Utiliser (cf. Figure 4.4), on mette en oeuvre la relvar piège ISP d'en-tête {*IngId*, *SgbdId*, *ProjId*, *Annee*} (cf. paragraphe 4.7), dont le prédicat est en réalité la conjonction des prédicats P1 et P2. ISP a elle aussi pour clé le quadruplet {*IngId*, *SgbdId*, *ProjId*, *Annee*}. Maintenant, certains concepteurs et DBA ont pour principe qu'en SQL toute clé primaire doit être singleton (mono-attribut) : on ajoute alors à l'en-tête d'ISP un attribut, nommons-le *IspId*, servant pour la clé primaire (symbolisée ci-dessous par un double soulignement). Le quadruplet {*IngId*, *SgbdId*, *ProjId*, *Annee*} est ravalé *de facto* au rang de clé alternative. Le contenu de la table SQL correspondante, nommons-la ISP2, est le suivant (avec, pour faire plaisir à CinePhil, auto-incrémentation de l'attribut *IspId*) :

ISP2	<u>IspId</u>	<u>IngId</u>	<u>SgbdId</u>	<u>ProjId</u>	<u>Annee</u>
	1	i1	s1	p1	2003
	2	i1	s1	p1	2004
	3	i1	s1	p1	2006
	4	i1	s1	p2	2003
	5	i1	s1	p2	2005
	6	i1	s4	p1	2003
	7	i1	s4	p1	2004
	8	i1	s4	p1	2006
	9	i1	s4	p2	2003
	10	i1	s4	p2	2005
	11	i2	s2	p1	2004
	12	i2	s2	p1	2005
	13	i2	s4	p1	2004
	14	i2	s4	p1	2005
	15	i3	s3	p4	2007
	16	i3	s3	p4	2008

Figure 4.6 - Ajout d'une clé mono-attribut (table ISP2)

Les seules dépendances fonctionnelles non triviales vérifiées par ISP2 sont les suivantes :

{*IspId*} → {*IngId*}, {*IspId*} → {*SgbdId*}, {*IspId*} → {*ProjId*}, {*IspId*} → {*Annee*},
 {*IngId*, *SgbdId*, *ProjId*, *Annee*} → {*IspId*}.

Et leurs déterminants respectifs sont les suivants : {*IspId*} et {*IngId*, *SgbdId*, *ProjId*, *Annee*}.

Ces déterminants constituent les seules clés candidates de la table, ce sont donc les seules surclés et la BCNF est vérifiée (cf. paragraphe 3.3.1). Comme par ailleurs {*IspId*} est clé mono-attribut, en vertu du théorème qui vient d'être énoncé, la 4NF est vérifiée elle aussi mais, très mauvaise nouvelle ! les dépendances multivaluées sont piégées ; en effet, si la DMV {*IngId*} →→ {*SgbdId*} vaut pour ISPV et ISP, elle ne vaut plus pour ISP2 qui n'est pas égale à la jointure naturelle de ses

projections {IngId, SgbdId} et {IngId, IspId, ProjId, Annee}. De fait, cette jointure produit des tuples n'appartenant pas à ISP2 (*spurious tuples* comme dirait Date) : par exemple, le tuple <1, i1, s4, p1, 2003> appartient seulement à la jointure.

En revanche, on notera que les DMV {IngId, IspId} →→ {SgbdId} et {IngId, IspId} →→ {ProjId, Annee} sont vérifiées. En effet, {IspId} étant clé, toute DF, donc toute DMV dont le multi-déterminant contient IspId est vérifiée (à cet effet, on peut invoquer le théorème de Heath puisqu'on peut en rester aux DF). Ainsi ISP2 peut être remplacée par ses projections {IngId, IspId, SgbdId} et {IngId, IspId, ProjId, Annee} mais pour un bien piètre résultat, car chacune d'elles contient autant de tuples qu'ISP2, conséquence de la présence systématique de la clé {IspId} : on apprend ainsi cinq fois qu'Albert maîtrise IMS et DB2. On peut dire que si ISP2 et ses projections respectent la 4NF à la lettre, elles la violent dans l'esprit.

Autrement dit, avant d'appliquer la règle « toute clé primaire sera mono-attribut », il aurait fallu normaliser la table ISP et n'appliquer cette règle qu'aux tables Maitriser et Affecter (cf. Figure 4.1) dont elle est en fait la jointure naturelle ; il y aurait eu cautères sur jambes de bois, mais en tout cas rien de bien grave puisque, par exemple, il est écrit cette fois-ci une fois et une seule qu'Albert maîtrise IMS et DB2. Les tables Maitriser et Affecter respectent authentiquement la 4NF (et aussi la 6NF, d'ailleurs), c'est-à-dire à la lettre, mais aussi dans l'esprit.

4.9. 4NF et relvars « toutes clés », une légende à détruire

Il est de coutume d'affirmer que si le travail de normalisation en BCNF a été mené à son terme, la vérification de la 4NF ne concerne plus que des relvars dont la clé est composée de l'ensemble des attributs de leur en-tête.

Il est un fait que jusqu'ici, les relvars et tables que nous avons passées à la moulinette de la 4NF étaient « toutes clés » : Maitriser, Affecter, ISPV, ISP, ISP2. Mais prenons un exemple proposé par Chris Date (« What's Normal Anyway? », Database Programming & Design, Volume 11 - Number 3, March 1998).

Considérons les règles de gestion de données suivantes, appliquées à une variante de l'exemple classique des cours dispensés par des enseignants :

- (RG1) Un enseignant enseigne au moins un sujet ;
- (RG2) Un sujet est enseigné par au moins un enseignant ;
- (RG3) Un sujet comporte au moins un thème ;
- (RG4) Un thème fait partie d'au moins un sujet ;
- (RG5) Quel que soit le sujet traité, il n'existe pas de relation particulière entre les enseignants et les thèmes ;
- (RG6) Toutefois, quand un enseignant traite d'un thème, c'est dans le cadre d'un seul sujet.

Ce que veut dire Date avec la règle RG5, c'est que les règles RG2 et RG3 donnent lieu aux dépendances multivaluées :

{Sujet} →→ {Enseignant} et {Sujet} →→ {Theme}

La règle RG6 fait quant à elle l'objet de la dépendance fonctionnelle :

{Enseignant, Theme} → {Sujet}

Un instantané de la relvar SEO correspondante (clé soulignée) :

SEO	Sujet	<u>Enseignant</u>	<u>Theme</u>
	Tutorial D	Hugh	The Askew Wall
	Tutorial D	Hugh	Les données temporelles
	Tutorial D	Chris	The Askew Wall
	Tutorial D	Chris	Les données temporelles
	Le Modèle relationnel	Ted	Le calcul relationnel
	Le Modèle relationnel	Ted	Les données temporelles
	Merise	Hugh	Le schéma directeur
	Merise	Hugh	Les modèles de données
	Merise	Hugh	Les modèles de traitements

Figure 4.7 - Un instantané de la relvar SEO

La relvar SEO vérifie la BCNF. En effet, elle ne comporte qu'une seule dépendance fonctionnelle non triviale, à savoir $\{Enseignant, Theme\} \rightarrow \{Sujet\}$ et le déterminant de cette DF est une surclé.

La relvar comporte les dépendances multivaluées non triviales $\{Sujet\} \twoheadrightarrow \{Enseignant\}$ et $\{Sujet\} \twoheadrightarrow \{Theme\}$ mais dont le multi-déterminant $\{Sujet\}$ n'est pas surclé, en conséquence de quoi la relvar ne vérifie pas la 4NF.

Il existe donc bien des relvars qui vérifient la BCNF mais pas la 4NF et qui de surcroît sont munies de clés qui ne se sont pas composées de l'ensemble des attributs de l'en-tête de la relvar.

Afin de normaliser en 4NF, on décompose la relvar SEO par projection (décomposition sans perte de données), pour obtenir les relvars SE et SO :

SE	<u>Sujet</u>	<u>Enseignant</u>
	Tutorial D	Hugh
	Tutorial D	Chris
	Le Modèle relationnel	Ted
	Merise	Hugh

SO	<u>Sujet</u>	<u>Theme</u>
	Tutorial D	The Askew Wall
	Tutorial D	Les données temporelles
	Le Modèle relationnel	Le calcul relationnel
	Le Modèle relationnel	Les données temporelles
	Merise	Le schéma directeur
	Merise	Les modèles de données
	Merise	Les modèles de traitements

Figure 4.8 - Décomposition de la relvar SEO

Malheureusement on a perdu la DF $\{Enseignant, Theme\} \rightarrow \{Sujet\}$, c'est-à-dire la règle de gestion RG6. On se retrouve donc face à l'alternative embarrassante, à savoir normaliser/ne pas normaliser, déjà traitée dans le contexte de la BCNF (cf. paragraphe 3.7, « [Un problème embarrassant de BCNF](#) »).

4.10. La 4NF et Merise

Dans les ouvrages sur Merise, il n'est pas rare de trouver des références à la normalisation. Ceci est on ne peut plus légitime, puisqu'après tout, les attributs composant l'en-tête d'une relvar ou d'une table, la liste des propriétés d'une entité-type (voire d'une association-type), la liste des attributs d'une classe, tout cela peut être vu comme la liste des variables d'un prédicat. Pour reprendre l'exemple des membres de DVP (cf. Figure 2.4) :

Table Membre	Entité-type Membre	Classe Membre																			
<table><tr><th>Membre</th></tr><tr><td><u>MbrId</u></td></tr><tr><td>Pseudonyme</td></tr><tr><td>DateInscription</td></tr><tr><td>Localisation</td></tr><tr><td>AdrCourriel</td></tr></table>	Membre	<u>MbrId</u>	Pseudonyme	DateInscription	Localisation	AdrCourriel	<table><tr><th>Membre</th></tr><tr><td><u>MbrId</u></td></tr><tr><td>Pseudonyme</td></tr><tr><td>DateInscription</td></tr><tr><td>Localisation</td></tr><tr><td>AdrCourriel</td></tr></table>	Membre	<u>MbrId</u>	Pseudonyme	DateInscription	Localisation	AdrCourriel	<table><tr><th>Membre</th></tr><tr><td>+ <u>MbrId</u></td></tr><tr><td>+ Pseudonyme</td></tr><tr><td>+ DateInscription</td></tr><tr><td>+ Localisation</td></tr><tr><td>+ AdrCourriel</td></tr><tr><td></td></tr></table>	Membre	+ <u>MbrId</u>	+ Pseudonyme	+ DateInscription	+ Localisation	+ AdrCourriel	
Membre																					
<u>MbrId</u>																					
Pseudonyme																					
DateInscription																					
Localisation																					
AdrCourriel																					
Membre																					
<u>MbrId</u>																					
Pseudonyme																					
DateInscription																					
Localisation																					
AdrCourriel																					
Membre																					
+ <u>MbrId</u>																					
+ Pseudonyme																					
+ DateInscription																					
+ Localisation																					
+ AdrCourriel																					

Figure 4.9 - Candidats à faire l'objet de prédicats

Ces représentations correspondent au prédicat pentadique « Membre chez DVP » :

Le membre identifié chez DVP par *MbrId* a pour pseudonyme *Pseudonyme*, il est inscrit depuis le *DateInscription*, il est localisé dans *Localisation* et il a pour adresse de courrier électronique *AdrCourriel*.

Étant donné que la normalisation des relvars met en jeu leurs attributs et leurs clés, il est bien naturel que l'on s'intéresse aussi à la normalisation des entités-types (et bien sûr des associations-types), en remplaçant les termes *attribut* et *clé candidate* (ou *surclé*) par ceux de *propriété* et *d'identifiant*, en mettant en évidence les dépendances fonctionnelles et multivaluées à partir des règles de gestion des données. Même principe pour les classes *mutatis mutandis*.

Concernant plus précisément la recherche méthodique des viols de 4NF, Fagin a montré la voie à suivre, mais manifestement celle-ci est escarpée. La recherche des dépendances multivaluées non triviales dont les parties gauches ne sont pas des surclés est un exercice qui pour sa part n'est pas toujours... trivial.

En ce sens, considérons à nouveau la table SQL ISP2 (cf. Figure 4.6). Le diagramme dans lequel elle s'intègre est le suivant (les attributs composant la clé alternative {IngId, SgbdId, ProjId, Annee} sont soulignés en pointillés, et ceux qui participent aux clés étrangères sont en italiques) :

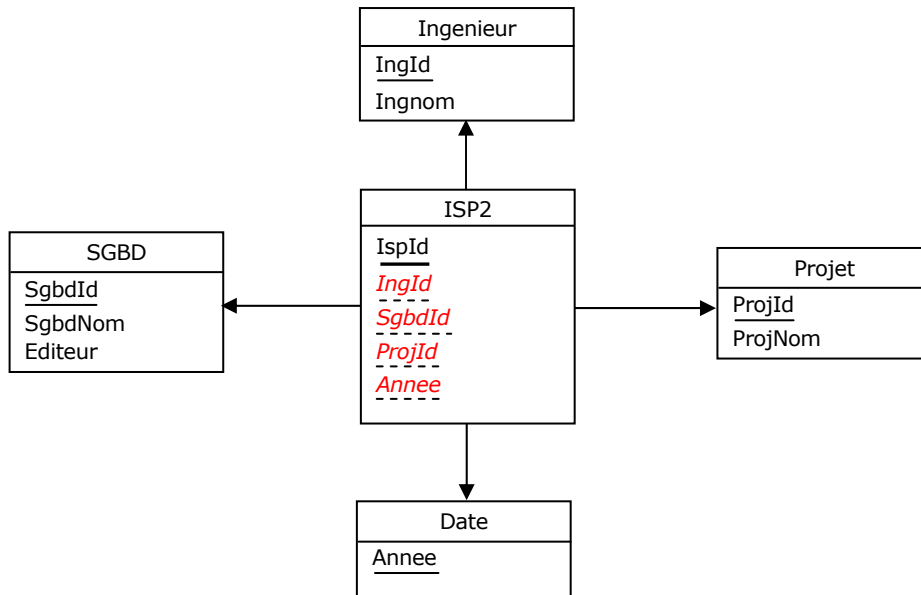


Figure 4.10 - Table ISP2 avant rétroconception

Si l'on effectue une rétroconception orientée Merise, on obtient la représentation conceptuelle suivante (dans laquelle on a renommé ISP2 en CV, comme *Curriculum Vitae*) :

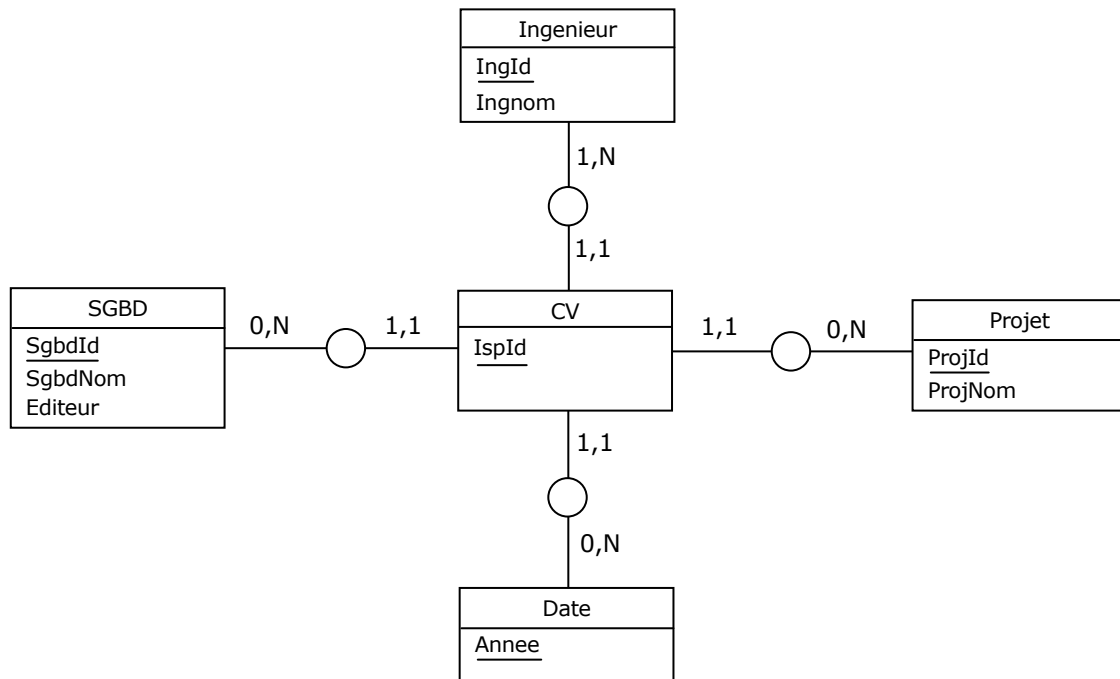


Figure 4.11 - Rétroconception de la table ISP2

A moins de se poser des questions sur l'absence de propriétés naturelles concernant l'entité-type CV et sur l'absence des associations-types telles que Maitriser et Affecter, il est probable qu'on ne cherchera malheureusement pas à débusquer un viol de 4NF pour cette entité-type peccamineuse.

Dans le même sens, considérons à nouveau la relvar ISP (cf. paragraphe 4.7). Le diagramme dans lequel elle s'intègre est le suivant :

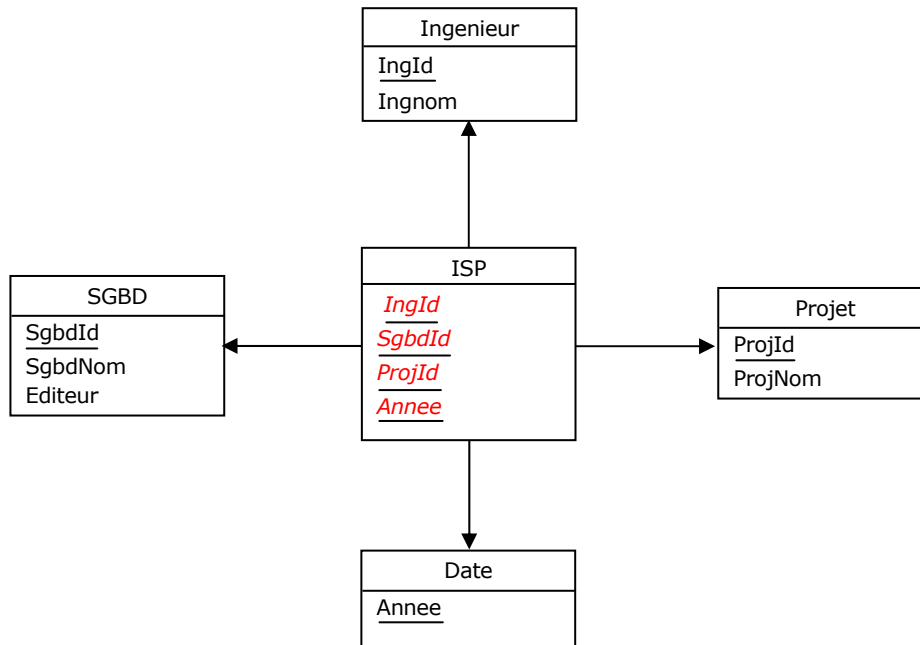


Figure 4.12 - Relvar ISP avant rétroconception

Par rétroconception on obtient la représentation conceptuelle suivante (dans laquelle on a renommé ISP en ConstituerCV) :

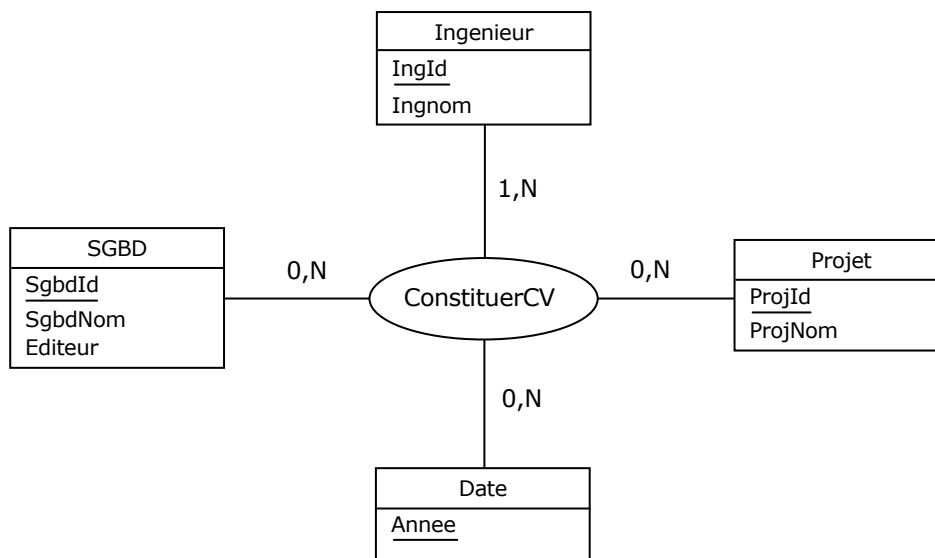


Figure 4.13 - Rétroconception de la relvar ISP

Le merisien averti, habitué à expertiser les associations-types de dimension supérieure à deux, confrontera cette représentation aux règles de gestion des données. De deux choses l'une :

- ✓ Ou bien ces règles précisent qu'effectivement on doit connaître les SGBD utilisés par les ingénieurs dans le cadre des projets, auquel cas le prédicat suivant est le bon (cf. « Piège de la connexion » au paragraphe 4.2) :

(P3) H_{xyzw} : L'ingénieur $IngId$ a utilisé le SGBD $SgbdId$ dans le cadre du projet $ProjId$ au cours de l'année $Annee$

Mais en conséquence de quoi le MCD doit résulter de la rétroconception du MLD de la Figure 4.4 :

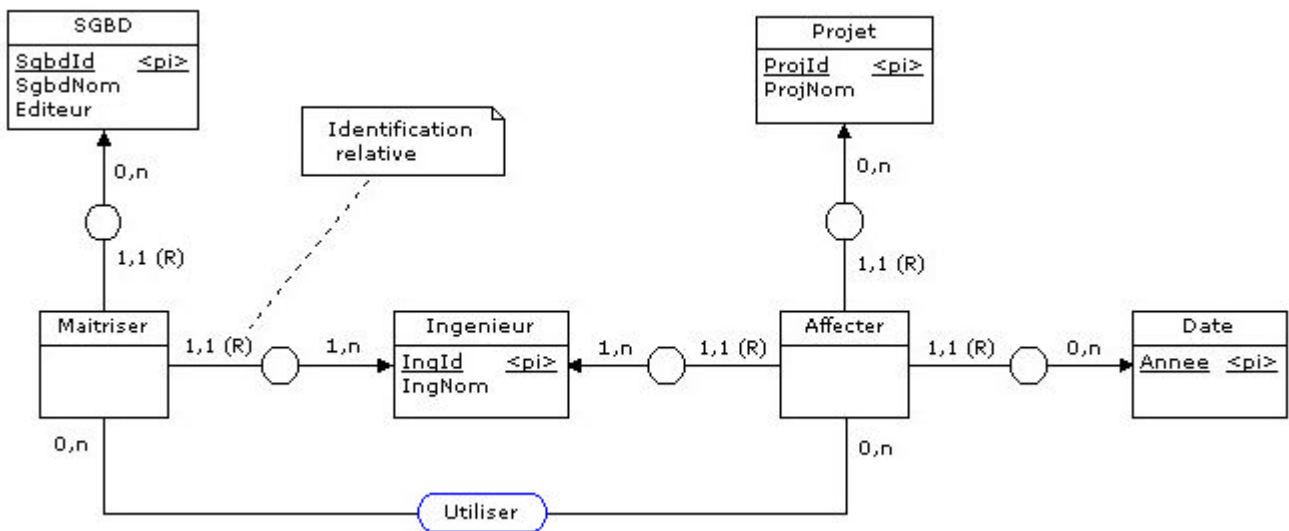


Figure 4.14 - Utilisation des SGBD par les ingénieurs dans le cadre des projets

- ✓ Ou bien ces règles ne précisent rien en ce sens et ce sont seulement celles qui ont été énoncées au paragraphe 4.2 qui prévalent (RG1 et RG3), auquel cas la représentation de la Figure 4.13 doit être remplacée par la suivante, radicalement différente et conforme aux prédicats P1 et P2 (paragraphe 4.2) :

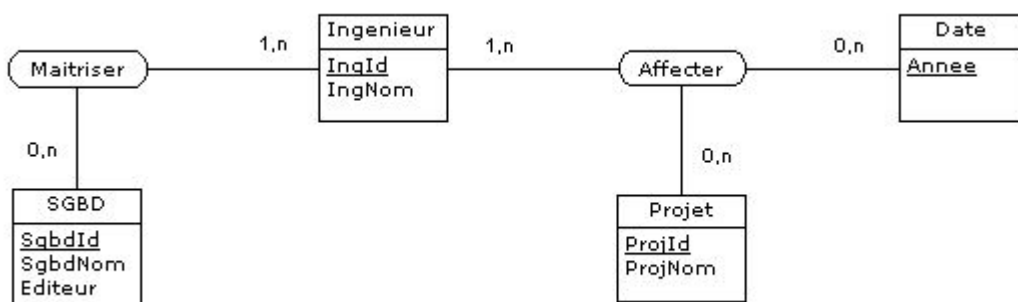


Figure 4.15 - Compétences et affectations des ingénieurs

Autrement dit, si l'on interprète correctement les règles de gestion des données et si l'on procède à ce remplacement, alors on montre que l'on sait modéliser, sans nécessairement connaître la 4NF. Néanmoins, bien maîtriser le concept de dépendance multivaluée et savoir débusquer les viols de 4NF est préférable si l'on veut être sûr de ne rien laisser passer. L'exemple du paragraphe 4.11, extrait d'un cas réel, est là pour montrer que l'on peut avoir des distractions qui pourraient coûter cher.

4.11. Merise et la chasse aux ternaires

On peut conjecturer que les viols de 4NF sont rares, mais il n'en demeure pas moins que lorsqu'ils se produisent, ils ne sautent pas aux yeux. Si on veut les débusquer, il faut, en amont, auditer les associations-types ternaires (et au-delà), qui figurent dans les diagrammes conceptuels.

Dans les années quatre-vingts, j'ai eu à expertiser le MCD (modèle conceptuel de données) d'une application bancaire ayant pour titre : « Déclaration des risques à la Banque de France ». Remontons dans le passé... Les établissements bancaires français sont tenus de faire des déclarations périodiques quant à l'état des crédits qu'ils consentent à leurs clients. Pour sa part, la Banque de France a pour mission de rassembler l'ensemble des déclarations effectuées par les établissements bancaires, et en retour, d'envoyer à chacun d'eux les récapitulatifs des risques au niveau national pour chaque client.

La banque dont j'expertise le MCD est la B.R.L. (Banque du Relationland, bien sûr...) Cette banque est organisée en guichets et filiales. L'objet de la déclaration est de fournir la liste des clients et de leurs contrats quand les sommes engagées dépassent un certain seuil, ainsi que la liste des établissements déclarants (guichets et filiales de la banque) garantissant les engagements.

Le fragment qui nous intéresse de la structure prévue dans le Système d'Information est le suivant : les contrats, les clients et les déclarants (constitués en pools pour gérer et garantir les contrats).

La partie correspondante du MCD est la suivante, d'allure tout à fait honnête, bien sûr noyée dans un diagramme d'une cinquantaine d'entités-types et d'associations-types, mais on arrive à y repérer l'association-type ternaire suivante :

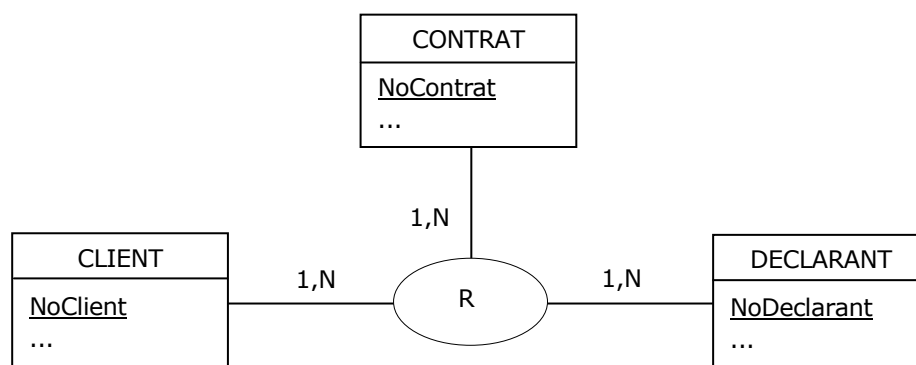


Figure 4.16 - MCD - Déclaration des risques

Une ternaire ? Suspicion ! Après entretien avec le chef de projet, les règles de gestion intéressantes se résumaient à celles-ci :

- (R1) Un client est titulaire d'au moins un contrat ;
- (R2) Un contrat a au moins un client pour titulaire ;
- (R3) Un déclarant garantit au moins un contrat ;
- (R4) Un contrat est garanti par au moins un déclarant.

Mais pas la moindre règle faisant état de relations entre les déclarants et les clients ; celles-ci existaient bien entendu, mais ne jouaient aucun rôle dans le cadre des déclarations des risques : pour chaque contrat faisant l'objet d'une déclaration, les clients co-titulaires du contrat étaient **indépendants** des déclarants garants du contrat. Autrement dit, l'association-type R (Déclarer contrat, de son vrai nom) aurait donné lieu à une table SQL violant la 4NF, du fait qu'elle aurait comporté les dépendances multivaluées non élémentaires suivantes :

{NoContrat} →→ {NoClient}

{NoContrat} →→ {NoDeclarant}

Pour éviter une telle situation délinquante et fort périlleuse pour l'application, une seule solution : en s'appuyant sur le 1er théorème de Fagin, « casser » R et produire le MCD suivant, avec $R = R1 \text{ JOIN } R2$:

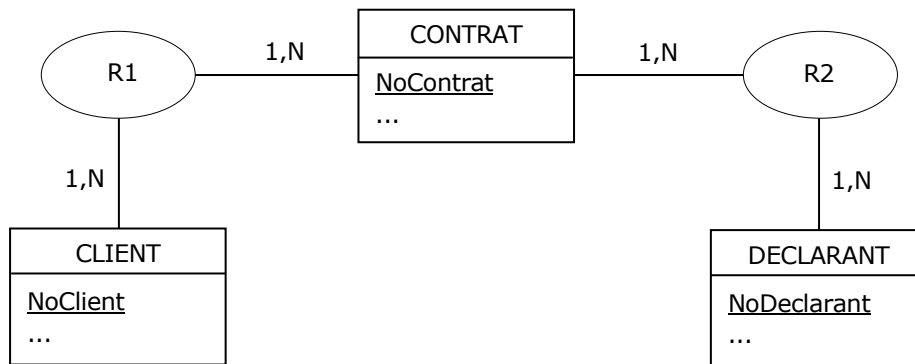


Figure 4.17 - MCD - Déclaration des risques - Respect de la 4NF

Sitôt dit, sitôt fait. L'exemple présenté se veut didactique. En réalité, contexte bancaire oblige, tout est daté, y compris les déclarations de risques. Autrement dit, le MCD initial tient compte des périodes couvertes par les déclarations :

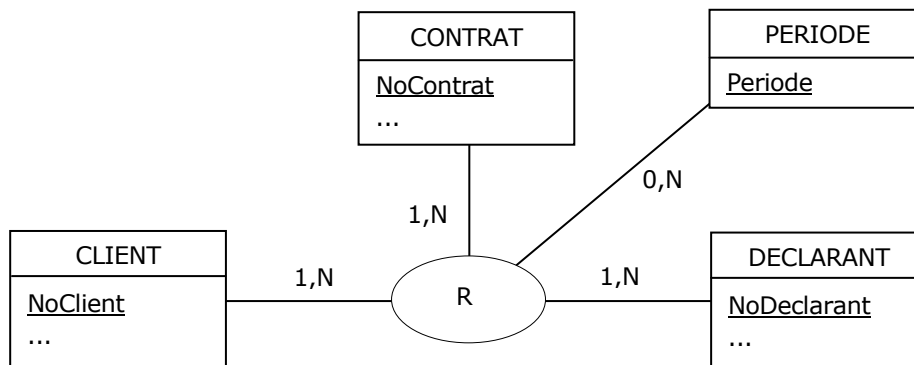


Figure 4.18 - MCD - Déclarations datées, avant normalisation

Après normalisation :

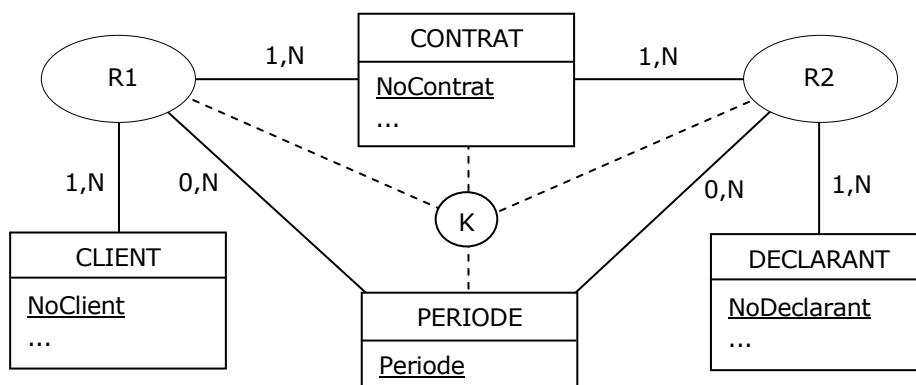


Figure 4.19 - MCD - Déclarations datées, après normalisation

⚠ K représente graphiquement la contrainte suivante : $R1 \{NoContrat, Periode\} = R2 \{NoContrat, Periode\}$, à savoir qu'au niveau relationnel, les projections de R1 et de R2 sur les attributs NoContrat et Periode doivent être égales. La contrainte K est garantie *de facto* avant normalisation, mais sans sa présence après normalisation, qu'est-ce qui empêcherait que pour un contrat donné on ait des périodes différentes côté CLIENT et côté DECLARANT ?

4.12. Observations concernant la décomposition des relvars

Les relvars Maitriser et Affecter présentées au paragraphe 4.2 respectent la 4NF, contrairement aux relvars ISPV (paragraphe 4.2) et ISP (paragraphe 4.7). De même, les relvars issues des associations-types R1 et R2 (paragraphe 4.11) sont en 4NF elles aussi, contrairement à la relvar R. Dans la mesure où l'on a fait du préventif (*better prevent than cure* comme dit un de mes amis qui vit au Relationland), tout est pour le mieux dans le meilleur des mondes. J'ai écrit au paragraphe 4.7 que la relvar ISP violait la 4NF, mais qu'elle était décomposable selon Maitriser et Affecter, *so far so good*.

Modifions maintenant légèrement (; -o^d) les règles de gestion RG1 et RG3 énoncées au paragraphe 4.2.

Règles initiales :

(RG1) Un ingénieur est compétent pour au moins un SGBD.

(RG3) Un ingénieur a été affecté à au moins un projet.

Nouvelles règles :

(RG1') Un ingénieur peut être compétent pour au moins un SGBD.

(RG3') Un ingénieur a pu être affecté à au moins un projet.

Cette fois-ci, il se peut que des ingénieurs n'aient pas de compétences en termes de SGBD (Philou et Mimile) ou qu'ils n'aient pas été affectés à des projets (Denis et Mimile).

Un instantané :

Ingenieur	<u>IngId</u>	IngNom
	i1	Albert
	i2	Bernard
	i3	Charles
	i4	Denis
	i5	Philou
	i6	Mimile

SGBD	<u>SgbdId</u>	SgbdNom
	s1	IMS
	s2	BS12
	s3	Oracle
	s4	DB2
	s5	IDMS

Projet	<u>ProjId</u>	ProjNom
	p1	Biglotron
	p2	Slalom
	p3	Flugdug
	p4	TantkYoradla
	p5	YaduPour

Maitriser	<u>IngId</u>	<u>SgbdId</u>
	i1	s1
	i1	s4
	i2	s2
	i2	s4
	i3	s3
	i4	s2
	i4	s3

Affecter	<u>IngId</u>	<u>ProjId</u>	<u>Annee</u>
	i1	p1	2003
	i1	p1	2004
	i1	p1	2006
	i1	p2	2003
	i1	p2	2005
	i2	p1	2004
	i2	p1	2005
	i3	p4	2007
	i3	p4	2008
	i5	p2	2003

Figure 4.20 - Les ingénieurs, leurs SGBD, leurs projets (nouvel instantané)

La vue ISPV, jointure naturelle des relvars Affecter et Maitriser ne change pas de valeur par comparaison avec celle qu'elle avait quand Denis, Philou et Mimile ne figuraient pas dans les effectifs.

ISPV n'a pas changé :

ISPV	<u>IngId</u>	<u>SgbdId</u>	<u>ProjId</u>	<u>Annee</u>
	i1	s1	p1	2003
	i1	s1	p1	2004
	i1	s1	p1	2006
	i1	s1	p2	2003
	i1	s1	p2	2005
	i1	s4	p1	2003
	i1	s4	p1	2004
	i1	s4	p1	2006
	i1	s4	p2	2003
	i1	s4	p2	2005
	i2	s2	p1	2004
	i2	s2	p1	2005
	i2	s4	p1	2004
	i2	s4	p1	2005
	i3	s3	p4	2007
	i3	s3	p4	2008

Rappel

En Tutorial D, la définition de la relvar est la suivante :

```
VAR ISPV VIRTUAL (Maitriser JOIN Affecter)
KEY {IngId, SgbdId, ProjId, Annee} ;
```

En SQL :

```
CREATE VIEW ISPV (IngId, SgbdId, ProjId, Annee)
AS SELECT a.IngId, a.SgbdId, b.ProjId, b.Annee
FROM Maitriser AS a INNER JOIN Affecter AS b
ON a.IngId = b.IngId ;
```

Figure 4.21 - ISPV : Jointure naturelle des tables Maitriser et Affecter

En effet, il faut se souvenir que le prédicat d'ISPV est dérivé, qu'il est la conjonction des prédicats P1 et P2 (cf. paragraphe 4.2), c'est-à-dire qu'ISPV ne permet de voir (a) que les ingénieurs qui maîtrisent au moins un SGBD **et** (b) qui ont été affectés à au moins un projet. Si l'on applique le 1er théorème de Fagin, on produira deux vues, disons ISPV1 et ISPV2 qui ne permettront de manipuler que des sous-ensembles *a priori* stricts des relvars Maitriser et Affecter (Denis et Philou en sont exclus), leur intérêt est donc nul. Supposons maintenant que nous ayons mis en oeuvre la relvar de base ISP évoquée au paragraphe 4.7, censée remplacer Maitriser et Affecter : au niveau logique, on ne disposera que du diagramme de la Figure 4.12 et au niveau conceptuel que de celui de la Figure 4.13. Mais cette fois-ci, il va falloir impérativement prendre en compte les ingénieurs qui, à l'image de Denis et Philou n'ont pas forcément de compétences en SGBD ou n'ont pas été affectés à des projets. Du coup, ISP se met à ressembler à n'importe quoi, avec l'apparition de valeurs « spéciales », du genre chaîne vide comme ci-dessous (ou pire encore, celle du bonhomme NULL) :

ISP	<u>IngId</u>	<u>SgbdId</u>	<u>ProjId</u>	<u>Annee</u>
	i1	s1	p1	2003
	i1	s1	p1	2004
	i1	s1	p1	2006
	i1	s1	p2	2003
	i1	s1	p2	2005
	i1	s4	p1	2003
	i1	s4	p1	2004
	i1	s4	p1	2006
	i1	s4	p2	2003
	i1	s4	p2	2005
	i2	s2	p1	2004
	i2	s2	p1	2005
	i2	s4	p1	2004
	i2	s4	p1	2005
	i3	s3	p4	2007
	i3	s3	p4	2008
	i4	s2		
	i4	s3		
	i5		p2	2003

Figure 4.22 - Relvar ISP, supposée remplacer à la fois Maitriser et Affecter

Mais là encore, il est évident que la projection d'ISP sur {IngId, SgbdId} d'une part et {IngId, ProjId, Annee} d'autre part ne permettra pas de retrouver les relvars Maitriser et Affecter, mais produira les relvars polluées, ISPa et ISPb :

Maitriser

<u>IngId</u>	<u>SgbdId</u>
i1	s1
i1	s4
i2	s2
i2	s4
i3	s3
i4	s2
i4	s3

ISPa

<u>IngId</u>	<u>SgbdId</u>
i1	s1
i1	s4
i2	s2
i2	s4
i3	s3
i4	s2
i4	s3
i5	

Affecter

<u>IngId</u>	<u>ProjId</u>	<u>Annee</u>
i1	p1	2003
i1	p1	2004
i1	p1	2006
i1	p2	2003
i1	p2	2005
i2	p1	2004
i2	p1	2005
i3	p4	2007
i3	p4	2008
i5	p2	2003

ISPb

<u>IngId</u>	<u>ProjId</u>	<u>Annee</u>
i1	p1	2003
i1	p1	2004
i1	p1	2006
i1	p2	2003
i1	p2	2005
i2	p1	2004
i2	p1	2005
i3	p4	2007
i3	p4	2008
i4		
i5	p2	2003

Figure 4.23 - Relvars ISPa et ISPb, supposées remplacer Maitriser et Affecter

La situation devient cauchemardesque si les partisans du bonhomme du NULL se mettent de la partie, car ISP (devenue table), ne peut même plus être dotée d'une clé primaire :

ISP	IngId	SgbdId	ProjId	Annee
	i1	s1	p1	2003
	i1	s1	p1	2004
	i1	s1	p1	2006
	i1	s1	p2	2003
	i1	s1	p2	2005
	i1	s4	p1	2003
	i1	s4	p1	2004
	i1	s4	p1	2006
	i1	s4	p2	2003
	i1	s4	p2	2005
	i2	s2	p1	2004
	i2	s2	p1	2005
	i2	s4	p1	2004
	i2	s4	p1	2005
	i3	s3	p4	2007
	i3	s3	p4	2008
	i4	s2	NULL	NULL
	i4	s3	NULL	NULL
	i5	NULL	p2	2003

Figure 4.24 - Table ISP infectée

Mais évidemment, les sectateurs du bonhomme du NULL bricoleront une rustine en définissant une colonne ad-hoc, IspId, comme dans le cas de la table ISP2 (cf. paragraphe 4.8) qui aurait alors pour clé primaire {IspId} (auto-incrémentée pour faire bonne mesure) et pour contenu (dans le moins pire des cas, c'est-à-dire si les dépendances multivaluées sont respectées) :

ISP2	<u>IspId</u>	IngId	SgbdId	ProjId	Annee
	1	i1	s1	p1	2003
	2	i1	s1	p1	2004
	3	i1	s1	p1	2006
	4	i1	s1	p2	2003
	5	i1	s1	p2	2005
	6	i1	s4	p1	2003
	7	i1	s4	p1	2004
	8	i1	s4	p1	2006
	9	i1	s4	p2	2003
	10	i1	s4	p2	2005
	11	i2	s2	p1	2004
	12	i2	s2	p1	2005
	13	i2	s4	p1	2004
	14	i2	s4	p1	2005
	15	i3	s3	p4	2007
	16	i3	s3	p4	2008
	17	i4	s2	NULL	NULL
	18	i4	s3	NULL	NULL
	19	i5	NULL	p2	2003

Figure 4.25 - Ajout d'une clé primaire ad-hoc (table ISP2)

On n'ose imaginer les conséquences d'une telle approche sur l'application « Déclaration des risques à la Banque de France » (cf. paragraphe 4.11)... Une fois de plus, mieux vaut prévenir que guérir et s'assurer au plus tôt du respect de la 4NF.

4.13. Approche ascendante vs approche descendante

Les auteurs spécialistes de Merise ne traitent pas de la 4NF à la manière dont nous l'avons fait ici. Tout au plus certains se bornent à la nommer, tandis que d'autres la connaissent très bien mais, sans y faire explicitement référence, usent de circonlocutions pour résoudre les viols de 4NF. A ce sujet, voir par exemple [Tabourier 1986] au paragraphe 6.2.5 « Problématique de la décomposition/normalisation », à propos de l'exercice fameux traitant des personnes qui conduisent des voitures et pénètrent dans des bâtiments. En l'occurrence, Tabourier ne donne aucune définition de la dépendance multivaluée et de la 4NF, il n'énonce aucun théorème, mais il connaît très bien le sujet, il manœuvre à la perfection et explique les bienfaits de la décomposition des associations-types impliquées dans cette affaire. Chapeau !

Mais à cette occasion (cf. pages 96-97 de son ouvrage), Tabourier s'en prend de façon surprenante aux spécialistes du Modèle Relationnel de Données (en l'occurrence Zaniolo, dont il cite l'article, à savoir [Zaniolo 1981]), à propos de l'élaboration de la structure des relvars :

« [Des] propositions très en vogue dans l'approche relationnelle suggèrent de mettre au jour les dépendances entre les informations élémentaires, puis d'appliquer des techniques algorithmiques ou interactives pour rassembler ces informations par "paquets" et construire ainsi la structure du modèle [...] Même en se limitant aux informations dûment répertoriées antérieurement, on dispose souvent de quelques milliers d'informations élémentaires, donc de quelques millions de couples d'informations, ce qui ne représente qu'une partie des dépendances possibles : bon courage ! »

Et plus loin (cf. page 169 du même ouvrage) :

« Le relationnel se présente, dans son esprit, comme une pure structure de données. Une obsession des chercheurs consiste d'ailleurs dans la question suivante : comment construire un ensemble minimal (en un certain sens) de schémas relationnels, rendant compte des dépendances de diverses natures ("fonctionnelles", "multivaluées", "de jointure") entre les constituants d'un vaste schéma unique ("universel") initial ? »

Halte au feu ! Le Modèle Relationnel ne se présente pas comme le prétend Tabourier. Je cite Ted Codd ([Codd 1980]) :

« Numerous authors appear to think of a data model as nothing more than a collection of data structure types. This is like trying to understand the way the human body functions by studying anatomy but omitting physiology. The operators and integrity rules are essential to any understanding of how the structures behave. In comparing data models people often ignore the operators and integrity rules altogether. When this occurs, the resulting comparisons run the risk of being meaningless. »

(Il apparaît que de nombreux auteurs pensent qu'un modèle de données n'est rien de plus qu'une collection de types de structures de données. C'est comme si on essayait de comprendre le fonctionnement du corps humain par l'étude de l'anatomie mais sans tenir compte de la physiologie. Les opérateurs et les règles d'intégrité sont essentiels pour comprendre comment se comportent les structures. En comparant les modèles de données, les gens ignorent souvent les opérateurs et les règles d'intégrité. Quand il en est ainsi, les comparaisons courent le risque d'être dénuées de sens.)

C'est de façon évidemment très formelle que pour sa part Zaniolo décrit les concepts et les algorithmes permettant de normaliser sans erreur et omission : tout au plus doit-on reconnaître qu'il faut s'accrocher pour apprécier son article de 50 pages à lire très, très, très lentement.

A titre indicatif, voici la définition que Zaniolo donne de la dépendance multivaluée :

« Si Θ et Δ sont des combinaisons d'attributs de la relation $R(\Omega)$, et $r \in R$, alors l'ensemble des Δ -valeurs associées à la valeur $r[\Theta]$ est notée $M_{\Delta}(r[\Theta])$. On peut formellement écrire :

$$M_{\Delta}(r[\Theta]) = \{r'[\Delta] \mid r' \in R \text{ et } r'[\Theta] = r[\Theta]\}$$

...»

Ou un aperçu de ses considérations sur un certain algorithme de décomposition :

« [...] Ainsi l'algorithme 5.1 a une limite supérieure évidente égale à $O(n^4 \log n)$ où n représente la cardinalité de $\bar{G}_0 \cup \bar{G}_L$. Cependant, la limite polynomiale ne s'applique qu'une fois calculés \bar{F}_0 , \bar{G}_0 et \bar{G}_L , puisque la taille de ces ensembles peut être exponentielle [...] »

A sa façon, Zaniolo nous souhaite lui aussi « Bon courage ! »

Disons que l'approche de Tabourier est descendante, tandis que celle de Zaniolo est ascendante, et la *relation universelle* à laquelle Tabourier fait allusion ne concerne pas les praticiens que nous sommes, astreints à produire des résultats selon un calendrier serré. Autrement dit, procédons d'abord comme Tabourier, commençons par produire des modèles conceptuels de données avec les entités-types et associations-types comportant les propriétés pertinentes. Mais n'en restons pas là, sortons ensuite l'artillerie lourde et, entité-type par entité-type, association-type par association-type, assurons-nous que la 4NF est respectée, grâce aux théorèmes, algorithmes et tous outils fournis par Codd, Fagin, Zaniolo et compagnie. Comme disait Poincaré à peu près en ces termes :

« L'intuition trouve, le raisonnement prouve ».

Incidentement, si Tabourier s'en prend aux spécialistes du Modèle Relationnel de Données, on peut faire observer que certains spécialistes de la méthode Merise préconisent eux aussi de commencer par fournir — à partir d'un inventaire baptisé dictionnaire des données — un graphe des dépendances fonctionnelles, une *structure d'accès théorique (sic !)* ou *couverture minimale* (un ersatz en fait de la couverture irréductible étudiée au paragraphe E.6), ainsi qu'une matrice de

ces dépendances pour en déduire les entités-types et associations-types des modèles conceptuels de données (quant à elles, les dépendances multivaluées et de jointure font l'objet d'un silence « éloquent »...) Voir par exemple [Matheron 2002]. Quoi qu'il en soit, pour reprendre les termes de Tabourier, avec des milliers d'informations élémentaires (n'oubliez pas $O(n^4 \log n)$...), alors d'accord : « Bon courage » !

Pour conclure avec Merise à propos de la normalisation :

Quand on en a bien compris l'enjeu et qu'on a eu à la pratiquer sans se relâcher, on ne peut rester indifférent à ce qui suit :

« La normalisation des tables consiste à répartir les informations dans les tables en fonction de règles. Seules les clés peuvent être redondées. Cinq étapes de normalisation sont distinguées. A chaque étape, les tables sont déclarées comme étant en première, deuxième... cinquième forme normale. Le but est d'arriver à la dernière étape pour obtenir des tables normalisées. *Cette normalisation est obligatoire uniquement si les tables ont été directement construites sans méthode.*

Ces règles peuvent être rapprochées des règles sur les informations d'individus ou de relation (une seule valeur d'information par individu ou relation par exemple). Quand le passage s'effectue du MCD MOD (MLD) au MPD, les tables sont obligatoirement normalisées. *Merise évite d'avoir à normaliser les tables.* »

Aux innocents les mains pleines... L'auteur de ces lignes — qui n'est d'ailleurs pas le seul à tenir ce genre de propos — est Michel Diviné qui a eu l'élégance de mettre à la disposition de tous son ouvrage « Parlez-vous Merise » (cf. [Diviné 1989]). Diviné est très bon dans sa partie, c'est-à-dire Merise, mais quand il en arrive au stade relationnel, ses considérations et conclusions relèvent de l'incantation et sont bonnes pour le pilon (voyez le chapitre qu'il a consacré au MPD). Qu'il nous pardonne ce jugement plutôt sévère, mais Merise n'évite pas d'avoir à normaliser, pas plus que la peur n'évite le danger, même s'il faut reconnaître que violer la 4NF n'est sans doute pas courant (conjecturons-le) quand on sait bâtir un MCD Merise ou un diagramme de classes UML et que l'on sait normaliser en BCNF.

4.14. Implication de la BCNF par la 4NF

A titre de récréation et pour réfuter les conclusions hallucinantes du clerc qui met en cause Delobel et Adiba (cf. paragraphe 2.8), nous reprenons ici la démonstration de l'implication de la BCNF par la 4NF, telle qu'elle figure dans [Delobel 1982]. D. & A. remplacent le terme *BCNF* par celui de *3FNBCK*, en l'honneur de William Kent qui a aiguillé les recherches de Boyce et Codd, ses collègues chez IBM. De même ils écrivent « 4FN », plus conforme en français que « 4NF ». L'expression « Schéma R » (R souligné) est à prendre ici comme synonyme de « relvar ».

Voici la démonstration donnée dans [Delobel 1982], dans laquelle $X \rightarrow Y$ représente une DF non triviale :

« On peut noter que la 4FN implique la 3FNBCK. En effet, supposons qu'un schéma R soit en 4FN et pas en 3FNBCK. D'après la définition de la 3FNBCK, il existe une dépendance $X \rightarrow Y$ où X ne contient pas une clé. Si $X \rightarrow Y$ est vérifiée dans R, la dépendance multivaluée $X \twoheadrightarrow Y$ l'est aussi. Or puisque R est en 4FN, X doit contenir une clé, d'où la contradiction. »

De fait, par référence à la définition de la BCNF (cf. paragraphe 3.3.1), si X et Y sont des sous-ensembles d'attributs de R, alors R est en BCNF si et seulement si pour chaque DF non triviale $X \rightarrow Y$ qui doit être vérifiée, le déterminant X de cette DF est une surclé de R. Si donc R n'est pas en BCNF, *a contrario* il existe une DF non triviale $X \rightarrow Y$ dont le déterminant X n'est pas surclé. Mais d'après la règle de réplcation (cf. paragraphe 4.5), si $X \rightarrow Y$ alors $X \twoheadrightarrow Y$: le multi-déterminant X de cette DMV n'étant pas surclé, R n'est donc pas en 4NF, en contradiction avec l'hypothèse de départ.

On trouve dans [Korth 1986] exactement la démonstration de Delobel et Adiba. On trouvera d'autres démonstrations, notamment dans [Fagin 1977] : Theorem 2, ou dans [Maier 1983] : Chapter 7, Lemma 7.2.

5. Cinquième forme normale

5.1. Introduction

Résumons ce que nous savons jusqu'ici de la normalisation. D'un point de vue technique, la normalisation en 2NF, 3NF, BCNF consiste à décomposer par projection une relvar non normalisée, en appliquant le théorème de Heath (cf. paragraphe 3.3.2) — ou une de ses variantes (cf. paragraphe E.7.1) —, théorème qui met en jeu les **dépendances fonctionnelles** :

Soit la relvar $R \{A, B, C\}$ dans laquelle A , B et C sont des sous-ensembles d'attributs. Si R satisfait à la dépendance fonctionnelle $A \rightarrow B$, alors R est égale à la jointure de ses projections sur $\{A, B\}$ et $\{A, C\}$.

Même principe quand il s'agit de la normalisation en 4NF qui consiste à décomposer par projection une relvar non normalisée, en appliquant le premier théorème de Fagin (cf. paragraphe 4.4), lequel met en jeu les **dépendances multivaluées** (en notant le « *si et seulement si* » caractérisant la préservation des dépendances multivaluées) :

Soit la relvar $R \{A, B, C\}$ dans laquelle A , B et C sont des sous-ensembles d'attributs. R est égale à la jointure de ses projections sur $\{A, B\}$ et $\{A, C\}$ si et seulement si R vérifie les dépendances multivaluées $A \twoheadrightarrow B$ et $A \twoheadrightarrow C$.

L'application de ces théorèmes consiste à décomposer par projection une relvar R non normalisée en **deux** relvars R_1 et R_2 dont la jointure naturelle est égale à R (décomposition sans perte de données), étant entendu que la normalité de R_1 et R_2 est à vérifier à son tour. Au bout du compte, chaque relvar de la base de données doit respecter la 4NF, au moins en théorie, car certaines situations (rares il est vrai) peuvent être embarrassantes (cf. paragraphes 3.7 et 4.9).

Mais on peut aller plus loin. Dans son tout premier article ([Codd 1969], pages 7 et suivantes), Codd nota déjà que certaines relvars ne pouvaient pas être décomposées sans perte de données en deux projections, mais pouvaient l'être par décomposition en **trois** projections (ou plus) **formant un cycle**. Comme l'écrit Chris Date (« The Birth of the Relational Model - Part 2 »

http://www.aisintl.com/case/library/Date_Birth%20of%20the%20Relational%20Model-2.html) :

« Remarkably, Codd also gives an example that shows he was aware back in 1969 of the fact that some relations can't be nonloss-decomposed into two projections but can be nonloss-decomposed into three! This example was apparently overlooked by most of the paper's original readers; at any rate, it seemed to come as a surprise to the research community when that same fact was rediscovered several years later... »

Quoi qu'il en soit, huit ans plus tard, Aho, Beeri et Ullman étudièrent de très près cette affaire, suivis par Jorma Rissanen et Jean-Marie Nicolas. Des travaux de ces chercheurs résulta le concept de dépendance de jointure (DJ ou JD pour *join dependency*), sorte de troisième étage de la fusée, bien calé sur les DF et les DMV. Les propriétés des DJ furent minutieusement étudiées et motivèrent à son tour Fagin qui nous livra son très bel article de 1979 « *Normal forms and relational database operators* » (cf. [Fagin 1979]). C'est à lui que revint l'honneur de découvrir la 5NF (qu'il préféra appeler Project/Join Normal Form (PJ/NF), « 5NF » manquant un peu de sel, d'autant plus qu'il montra qu'avec la décomposition des relvars par le mécanisme de **projection/jointure**, on ne pouvait aller plus loin). Dans ce qui suit, quand nous mentionnons Fagin sans précision particulière, nous faisons implicitement référence à cet article.

Le sujet de la normalisation par projection/jointure était clos (à ceci près qu'une vingtaine d'années plus tard Date proposa une 6e forme normale, car certaines relvars en 5NF peuvent encore être décomposées avec grand profit, quand des données de type intervalle — essentiellement **temporelles** — sont parties prenantes).

En tout état de cause, Aho et les autres montrèrent que certaines relvars comportent des redondances face auxquelles les dépendances fonctionnelles et les dépendances multivaluées ne nous sont d'aucun secours, mais qui ne résistent pas à une attaque par les dépendances de jointure, la 5NF étant là pour nous assurer d'une base de données bien normalisée.

5.2. Dépendance de jointure

Définition :

Soit une relvar $R \{A, B, \dots, M\}$ où A, B, \dots, M représentent des sous-ensembles d'attributs de l'en-tête de R . La dépendance de jointure (DJ) notée :

$\ast\{A, B, \dots, M\}$

est vérifiée par R si et seulement si chaque valeur de R est égale à la jointure de ses projections sur A, B, \dots, M .

En particulier, si son en-tête ne comporte que deux sous-ensembles d'attributs, disons A et B , la relvar $R \{A, B\}$ vérifie la DJ $\ast\{A, B\}$ si et seulement si elle est égale à la jointure de ses projections sur A et B .

Faisant référence à son 1er théorème (cf. paragraphe 4.4), Fagin précise que toute dépendance multivaluée peut être représentée par une dépendance de jointure. Soit la relvar $R \{A, B, C\}$ où A, B, C sont des sous-ensembles d'attributs de R , l'union de A et de B étant notée AB et celle de A et de C notée AC :

La dépendance multivaluée $A \twoheadrightarrow B$ est vérifiée par R si et seulement si la dépendance de jointure $\ast\{AB, AC\}$ l'est aussi.

Ce que Date résume ainsi de façon synthétique : $A \twoheadrightarrow B \mid C \equiv \ast\{AB, AC\}$

Ainsi, la dépendance multivaluée peut être vue comme un cas particulier de la dépendance de jointure et en contrepartie, il y a des DJ qui ne sont pas des DMV et qu'on cherchera à mettre en évidence, en partant du principe qu'elles comportent alors plus de deux sous-ensembles d'attributs.

5.3. Relvars formant un cycle

Considérons le diagramme merisien ci-dessous, selon lequel les ingénieurs spécialistes en bases de données de la SARL *Les Prévoyants de l'Avenir des Bases de Données* (LPABDD, filiale de DVP), sont affectés chez les clients de cette entreprise, afin d'administrer les bases de données de ces derniers, les piloter dans l'utilisation des SGBD, etc.

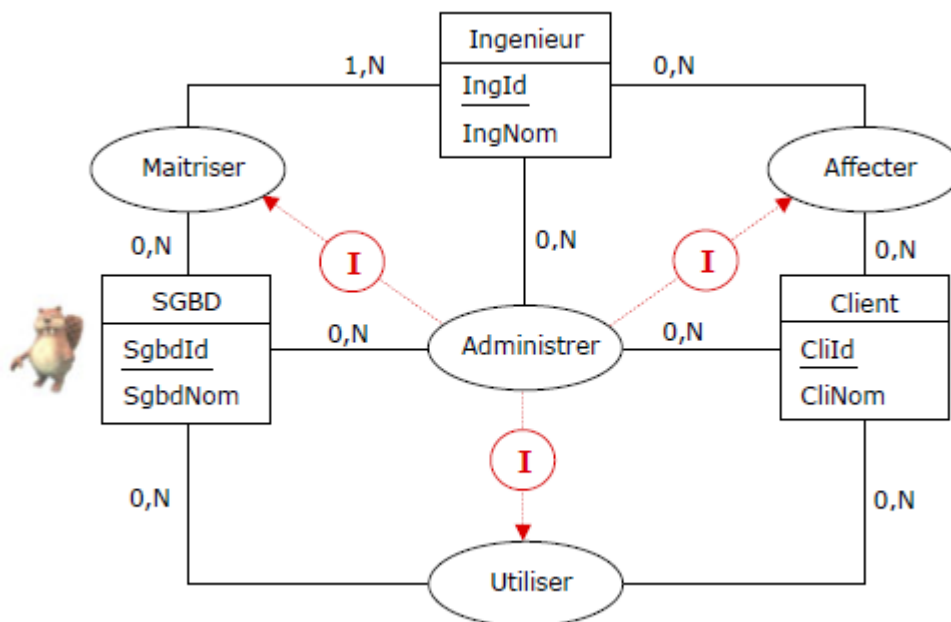


Figure 5.1 - MCD - Mise en œuvre de l'association-type Administrer

Les ingénieurs maîtrisent donc des SGBD, ils sont affectés chez des clients qui utilisent ces SGBD. Les contraintes d'inclusion (symbolisées par la lettre I cerclée de rouge) sont là pour signifier qu'un ingénieur ne peut être administrateur d'un SGBD qu'à la condition de maîtriser celui-ci, qu'il ne peut être administrateur chez un client qu'à la condition d'y être affecté et qu'il ne peut être administrateur chez un client que de SGBD utilisés par celui-ci.

Le MLD (Modèle Logique des Données) dérivé est le suivant :

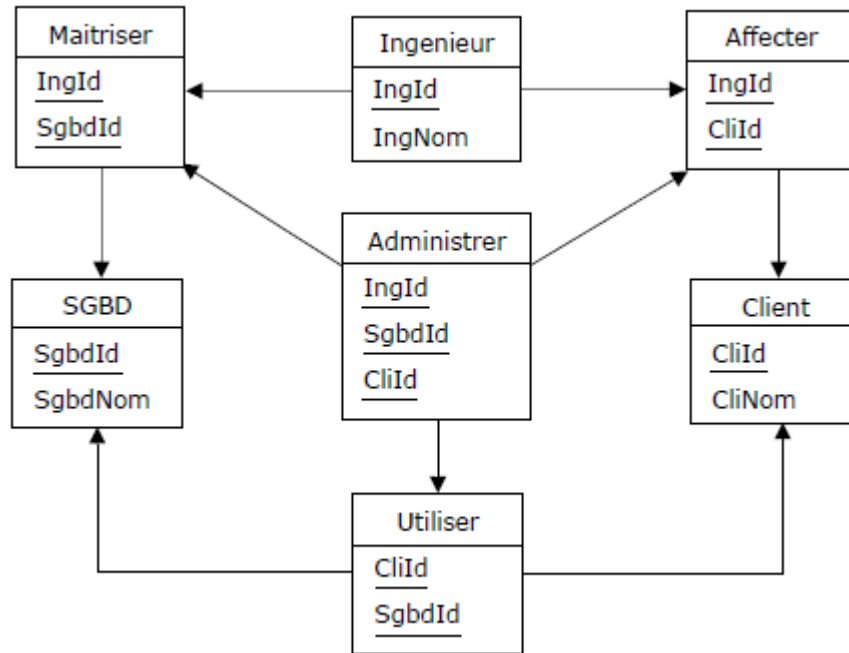


Figure 5.2 - MLD - Mise en œuvre de l'association-type Administrer

Un instantané :

Maitriser	<u>IngId</u>	<u>SgbdId</u>
	Philou	DB2
	Philou	IMS
	Philou	Adabas
	Philou	MS SQL
	Mimile	DB2

Affecter	<u>IngId</u>	<u>CliId</u>
	Philou	DVP
	Philou	HAL
	Mimile	DVP

Utiliser	<u>CliId</u>	<u>SgbdId</u>
	DVP	DB2
	DVP	IMS
	HAL	Adabas
	HAL	DB2

Administrer	<u>IngId</u>	<u>SgbdId</u>	<u>CliId</u>
	Philou	DB2	DVP
	Philou	DB2	HAL
	Philou	IMS	DVP
	Mimile	DB2	DVP

Figure 5.3 - Ingénieurs, SGBD, Clients : Administrer

On notera que le client HAL utilise Adabas, et qu'aucun ingénieur de la filiale de DVP n'administre ce SGBD : chez LPABDD les commerciaux devront s'activer afin de combler cette lacune...

Considérons maintenant la double jointure naturelle (en Tutorial D) des relvars Maitriser, Utiliser et Affecter :

JOIN {Maitriser, Utiliser, Affecter} ;

Ou l'équivalent SQL :

SELECT x.IngId, y.SgbdId, z.CliId
FROM Maitriser **AS** x **NATURAL JOIN** Utiliser **AS** y **NATURAL JOIN** Affecter **AS** z ;

Cette double jointure est **cyclique** et produit la relation suivante, appelons-la MUA, dont la relation Administrer est ici un sous-ensemble strict (car le tuple <Philou, Adabas, HAL> n'appartient pas à cette relation, nous y reviendrons) :

MUA	<u>IngId</u>	<u>SgbdId</u>	<u>CliId</u>
	Philou	DB2	DVP
	Philou	DB2	HAL
	Philou	Adabas	HAL
	Philou	IMS	DVP
	Mimile	DB2	DVP

Figure 5.4 - Cycle entre Maitriser, Utiliser et Affecter

A cette relation (valeur) on ne peut pas associer de DMV non triviale. En effet :

La DMV $\{IngId\} \rightarrow \{SgbdId\} \mid \{CliId\}$ n'est pas vérifiée pour MUA puisqu'elle présuppose l'existence des tuples <Philou, IMS, HAL> et <Philou, Adabas, DVP>, lesquels sont absents de MUA.

La DMV $\{SgbdId\} \rightarrow \{IngId\} \mid \{CliId\}$ n'est pas non plus vérifiée puisqu'elle présuppose l'existence du tuple <Mimile, DB2, HAL>, lequel est absent de MUA.

La DMV $\{CliId\} \rightarrow \{IngId\} \mid \{SgbdId\}$ n'est pas non plus vérifiée puisqu'elle présuppose l'existence du tuple <Mimile, IMS, DVP>, lequel est absent de MUA.

Mais si l'on projette MUA sur les paires IS $\{IngId, SgbdId\}$, SC $\{SgbdId, CliId\}$ et CI $\{CliId, IngId\}$:

IS	<u>IngId</u>	<u>SgbdId</u>
	Philou	DB2
	Philou	IMS
	Philou	Adabas
	Mimile	DB2

SC	<u>SgbdId</u>	<u>CliId</u>
	DB2	DVP
	IMS	DVP
	Adabas	HAL
	DB2	HAL

CI	<u>CliId</u>	<u>IngId</u>
	DVP	Philou
	HAL	Philou
	DVP	Mimile

Puis, si l'on effectue la jointure naturelle par exemple de IS et SC, on produit la relation ISC :

ISC	<u>IngId</u>	<u>SgbdId</u>	<u>CliId</u>
	Philou	DB2	DVP
	Philou	DB2	HAL
	Philou	IMS	DVP
	Philou	Adabas	HAL
	Mimile	DB2	DVP
	Mimile	DB2	HAL

Du fait de la présence du tuple <Mimile, DB2, HAL> dans ISC, celle-ci est un surensemble strict de MUA, mais si l'on effectue la jointure naturelle de ISC et CI, alors on évacue le tuple étranger et l'on retrouve la relation MUA (comme prévu par Codd en 1969) : $MUA = JOIN \{IS, SC, CI\}$.

Dans ces conditions, la relation MUA satisfait à la DJ suivante, qui ne peut être ramenée à une combinaison de DMV non triviales (on a montré que MUA en était dépourvue) :

$\ast\{\{IngId, SgbdId\}, \{SgbdId, CliId\}, \{CliId, IngId\}\}$

DJ qui exprime en fait la contrainte suivante :

Si $\langle Philou, DB2 \rangle \in IS$ et $\langle DB2, DVP \rangle \in SC$ et $\langle DVP, Philou \rangle \in CI$ alors $\langle Philou, DB2, DVP \rangle \in MUA$

(A ce sujet, on pourra utilement se reporter à [Date 2004], paragraphe 13.3 « Join Dependencies and Fifth Normal Form » ou à [Date 2009], Appendice B « Database Design Theory », paragraphe « More on 5NF »).

Supposons maintenant que les choses évoluent chez LPABDD et que la maîtrise d'ouvrage décide de faire de cette contrainte une règle de gestion. Dans le dossier de conception, on la réécrit sous une forme plus digeste pour l'utilisateur :

Si un ingénieur maîtrise un SGBD utilisé par un client chez qui il affecté, alors cet ingénieur administre ce SGBD chez ce client.

Mais la relation Administrer de la Figure 5.3 n'est alors plus valide. En effet Philou maîtrise Adabas et il est affecté chez HAL qui utilise Adabas : il y manque le tuple **<Philou, Adabas, HAL>** et pour respecter la nouvelle règle de gestion on devra l'ajouter (pour retrouver ainsi MUA).

Suite à cet ajout, la relation Administrer ci-dessous est conforme à la nouvelle règle de gestion :

Administrer	<u>IngId</u>	<u>SgbdId</u>	<u>CliId</u>
	Philou	DB2	DVP
	Philou	DB2	HAL
	Philou	Adabas	HAL
	Philou	IMS	DVP
	Mimile	DB2	DVP

Figure 5.5 - Relation Administrer complétée

Mais désormais chaque valeur (relation) prise par la relvar Administrer fait double emploi avec la valeur prise dans le même temps par la jointure naturelle MUA des relvars Maitriser, Utiliser et Affecter (en notant qu'Administrer vérifie donc elle aussi la DJ $*\{\{IngId, SgbdId\}, \{IngId, CliId\}, \{SgbdId, CliId\}\}$) : il est inutile de matérialiser Administrer et au besoin on se contentera d'en faire une **vue** (pour ne pas avoir à modifier les programmes qui manipulent la relvar, conformément à la 9e des [12 règles de Codd](#)). En tant que relvar de base, Administrer peut disparaître du MLD (et doit même disparaître, cf. paragraphe 5.4), tandis qu'à son tour le diagramme conceptuel (Figure 5.1) devient le suivant :

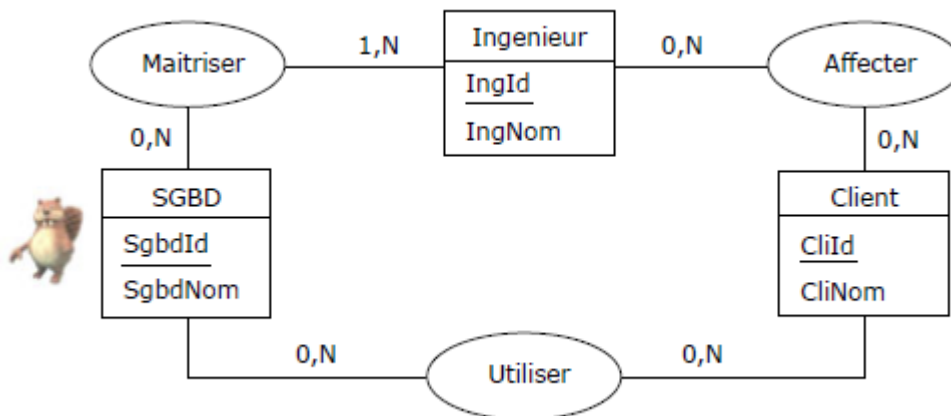


Figure 5.6 - Élimination de l'association-type Administrer

! A noter que, pour sa part, la relation IS $\{IngId, SgbdId\}$ n'est pas égale à la relation Maitriser $\{IngId, SgbdId\}$, celle-ci comportant un tuple **<Philou, MS SQL>** n'appartenant pas à IS qui n'est donc qu'un sous-ensemble, une restriction de Maitriser (conformément à la contrainte d'inclusion de la Figure 5.1). Par conséquent, qu'elle soit de base ou dérivée (vue), la relvar Administrer n'est pas décomposable en Maitriser, Affecter, Utiliser, tandis qu'elle en est la jointure naturelle, mais ceci n'a aucune incidence sur les règles de gestion précédemment énoncées.

Nota bene. Pour l'anecdote, en remplaçant le trio Ingenieur, SGBD, Client par le suivant : Fournisseur, Produit, Projet, on retrouve l'exemple donné par Codd en 1969, dans lequel les fournisseurs pourvoient en produits, lesquels sont utilisés pour des projets (voir aussi [Date 2004] au paragraphe 13.3 « Join Dependencies and Fifth Normal Form »).

On peut aussi considérer que ce trio correspond à celui de Nicolas ([Nicolas 1978]) et repris par Fagin, les acteurs étant cette fois-ci des VRP multiscartes proposant à la clientèle des produits fabriqués par les entreprises qu'ils représentent.

A l'attention des merisiens : On peut encore considérer qu'il s'agit d'une autre transposition, celle de l'exemple proposé dans [Tabourier 1986] : des personnes pénètrent dans des bâtiments, conduisent des voitures, lesquelles sont garées dans les bâtiments. Yves Tabourier a réussi à traiter des dépendances de jointure (et de la 5NF), sans en faire mention, tout comme pour les dépendances multivaluées et la 4NF (cf. paragraphe 4.10). *Well done! Mr. Tabourier...*

5.4. Dépendance de jointure triviale

Soit une relvar $R \{A, B, \dots, M\}$ où A, B, \dots, M représentent des sous-ensembles d'attributs de l'en-tête H de R . La dépendance de jointure $*\{A, B, \dots, M\}$ vérifiée par R est triviale si et seulement si au moins un des sous-ensembles A, B, \dots, M , a pour éléments l'ensemble des attributs de H .

Par exemple, la relvar MUA (Figure 5.4) vérifie la DJ $*\{\{IngId, SgbdId\}, \{IngId, CliId\}, \{SgbdId, CliId\}\}$ comme on l'a vu, mais celle-ci n'est pas triviale car ses éléments $\{IngId, SgbdId\}$, $\{IngId, CliId\}$ et $\{SgbdId, CliId\}$ sont tous des sous-ensembles stricts de l'en-tête de MUA. Même chose pour la relvar Administrer « complétée » (Figure 5.5).



Le concept de dépendance de jointure triviale intervient dans la définition de la 5NF et dans celle de la 6NF.

5.5. Définition de la 5NF (PJ/NF)

Une relvar R (1NF) est en 5NF (ou PJ/NF, *Project/Join Normal Form*) si et seulement si chaque dépendance de jointure non triviale à laquelle elle satisfait est une conséquence logique de ses clés.
(« Conséquence logique » signifiant en l'occurrence : pour chaque dépendance de jointure $*\{R_1, \dots, R_m\}$, chaque R_i ($1 \leq i \leq m$) est une clé de R).

La définition donnée par Fagin :

A 1NF relation schema R^* with attributes X is in PJ/NF if $\Delta \models \sigma$ for each JD σ in R^* , where Δ is the set of key dependencies of R^* . Thus, every JD is the result of keys.

N.B. R^* représente l'en-tête de la relvar R . « $\Delta \models \sigma$ » se lit ainsi : Δ implique logiquement σ , ou encore σ est une conséquence logique de Δ . Par ailleurs une dépendance de clé est une dépendance fonctionnelle $K \rightarrow X$ vérifiée par R , telle que X représente l'ensemble des attributs de R : K est donc une clé de R .

Pour en revenir à la relvar Administrer dans sa version « complétée » (cf. Figure 5.5) :

- (a) Elle fait l'objet de la DJ non triviale $*\{\{IngId, SgbdId\}, \{SgbdId, CliId\}, \{CliId, IngId\}\}$;
- (b) Elle a pour unique clé candidate $K = \{IngId, SgbdId, CliId\}$, car aucun des sous-ensembles stricts $\{IngId, SgbdId\}$, $\{IngId, CliId\}$ et $\{SgbdId, CliId\}$ ne vérifie la propriété d'unicité des clés.

Comme aucun des sous-ensembles $\{IngId, SgbdId\}$, $\{IngId, CliId\}$ et $\{SgbdId, CliId\}$ de la DJ ne contient K , la relvar ne respecte pas la 5NF. Normaliser consiste alors à la remplacer par ses projections $\{IngId, SgbdId\}$, $\{IngId, CliId\}$ et $\{SgbdId, CliId\}$. Comme on dispose déjà des relvars Maitriser, Affecter et Utiliser qui en sont des sur-ensembles, on se contentera d'éliminer purement et simplement la relvar Administrer, comme dans le cas de la Figure 5.6.

Si malgré tout on devait conserver cette relvar, toujours eu égard à la 9e des 12 règles de Codd, ça devrait être seulement sous forme de vue. Par exemple, en Tutorial D :

VAR Administrer **VIRTUAL** (**JOIN** $\{Maitriser, Utiliser, Affecter\}$) **KEY** $\{IngId, SgbdId, CliId\}$;

Si le chef n'aime pas les vues (ça arrive) et qu'il faille donc mettre en œuvre Administrer sous forme d'une relvar de base, alors il deviendrait indispensable de mettre en œuvre une contrainte pour s'assurer de la pérennité de la DJ non triviale $*\{\{IngId, SgbdId\}, \{SgbdId, CliId\}, \{CliId, IngId\}\}$. En Tutorial D :

CONSTRAINT Administrer_Contrainte_01

Administrer = **JOIN** $\{Maitriser, Utiliser, Affecter\}$;

En SQL, il faudrait en passer par une assertion (ou des triggers si le SGBD ne connaît pas les assertions).

5.6. Exemples de relvars respectant ou non la 5NF

On vient de voir que la relvar Administrer « complétée » ne respecte pas la 5NF (alors qu'elle respecte la 4NF), conséquence de la mise en application de la règle de gestion selon laquelle « Si un ingénieur maîtrise un SGBD utilisé par un client chez qui il affecté, alors cet ingénieur administre ce SGBD chez ce client ». En revanche, tant que cette règle n'existait pas (cf. Figure 5.1 et suivantes), la relvar Administrer respectait la 5NF. En effet, la jointure de ses projections avait pour résultat une relation comportant un corps étranger (cf. Figure 5.4), à savoir le tuple **<Philou, Adabas, HAL>** ne figurant pas dans la relation Administrer de la Figure 5.3 : la DJ $*\{\{IngId, SgbdId\}, \{SgbdId, CliId\}, \{CliId, IngId\}\}$ n'existait pas.

Pour leur part, les relvars Maitriser, Affecter et Utiliser dont on a des instantanés dans la Figure 5.3 respectent la 5NF. Prenons le cas de la relvar Maitriser : la seule DJ qu'elle comporte $*\{\{IngId, SgbdId\}\}$ est triviale puisque l'en-tête de la relvar est la paire $\{IngId, SgbdId\}$ (qui représente aussi la seule clé de la relvar). Ce qui vaut pour Maitriser vaut aussi pour Affecter et Utiliser, *mutatis mutandis*.

Autre exemple : considérons la relvar FPV étudiée notamment aux paragraphes 3.2.2 et 3.3.3. Elle satisfait à la DJ $*\{FV, FPQ\}$ puisqu'elle est la jointure naturelle des relvars FV et FPQ (cf. paragraphe 3.3.3). Elle a pour seule clé la paire $\{Four_No, Piece_No\}$, paire qui n'appartient pas à FV, en conséquence de quoi FPV ne respecte pas la 5NF (on savait du reste qu'elle ne respecte même pas la 2NF, cf. paragraphe 3.4.2).

En revanche, la relvar FV $\{Four_No, Ville\}$ ayant pour clé $\{Four_No\}$ n'est pas décomposable sans perte, et a pour seule DJ $*\{\{Four_No, Ville\}\}$ laquelle est triviale (tout en notant que la clé $\{Four_No\}$ est évidemment incluse dans $\{Four_No, Ville\}$). La relvar FV respecte la 5NF.

De même (toujours au paragraphe 3.3.3), la relvar FPQ $\{Four_No, Piece_No, Quantite\}$ de clé $\{Four_No, Piece_No\}$ n'est pas décomposable sans perte. Elle a pour seule DJ $*\{\{Four_No, Piece_No, Quantite\}\}$ laquelle est triviale : la relvar respecte donc la 5NF.

Prenons un autre exemple, celui d'une relvar en 2NF mais pas en 3NF. Soit la relvar suivante :

Salarie $\{\underline{SalId}, EntId, EntNom\}$ (clé soulignée)

L'attribut SalId représente l'identifiant (au sens Merise) d'un salarié, EntId représente l'identifiant de l'entreprise de ce salarié et EntNom le nom de cette entreprise.

Les DF sont les suivantes :

DF1 $\{SalId\} \rightarrow \{EntId, EntNom\}$

DF2 $\{EntId\} \rightarrow \{EntNom\}$

Par application du théorème de Heath et de la règle de Rissanen de préservation des dépendances (cf. paragraphe 3.3.2), la relvar Salarie peut être décomposée sans perte selon les projections :

$\{\underline{SalId}, EntId\}$ (clé soulignée)

$\{\underline{EntId}, EntNom\}$ (clé soulignée)

Étant décomposable sans perte, la relvar Salarie satisfait à la DJ non triviale $*\{\{SalId, EntId\}, \{EntId, EntNom\}\}$. Elle a pour unique clé $\{SalId\}$, et comme le sous-ensemble $\{EntId, EntNom\}$ n'est pas clé de la relvar, celle-ci ne respecte pas la 5NF.

Encore un exemple, celui d'une relvar respectant la BCNF mais pas la 4NF. Soit la relvar ISPV de la Figure 4.3 et décortiquée notamment dans les paragraphes 4.3 et 4.7 :

ISPV $\{IngId, SgbdId, ProjId, Annee\}$

ISPV a pour seule clé $\{IngId, SgbdId, ProjId, Annee\}$.

Elle comporte les DMV $\{IngId\} \rightarrow \{SgbdId\}$ et $\{IngId\} \rightarrow \{ProjId, Annee\}$.

ISPV satisfait donc à la DJ $*\{\{IngId, SgbdId\}, \{IngId, ProjId, Annee\}\}$, or, ni $\{IngId, SgbdId\}$ ni $\{IngId, ProjId, Annee\}$ ne sont clés de ISPV : cette relvar ne respecte pas la 5NF (comme on l'a rappelé, elle ne respectait déjà pas la 4NF).

5.7. Parallèle entre la BCNF, la 4NF et la 5NF

Fagin fournit un parallèle très intéressant et élégant entre la BCNF, la 4NF et la 5NF. Date l'a résumé ainsi ([Date 2004]) :

- Une relvar R est en BCNF si et seulement si chaque DF à laquelle elle satisfait est impliquée par les clés candidates de R.
- Une relvar R est en 4NF si et seulement si chaque DMV à laquelle elle satisfait est impliquée par les clés candidates de R.
- Une relvar R est en 5NF si et seulement si chaque DJ à laquelle elle satisfait est impliquée par les clés candidates de R.

5.8. Un autre théorème de Date et Fagin

Si une relvar est en 3NF et si chacune de ses clés candidates ne comporte qu'un seul attribut, alors cette relvar est en 5NF.

Ce théorème de [Date et Fagin](#) (théorème 4.1) est utile, il permet en effet de restreindre la liste des relvars à vérifier, puisque celles dont les clés sont seulement mono-attributs et qui sont en 3NF sont *de facto* en 5NF...

Encore un théorème en passant (cf. [Date 2009] page 296) :

Si une relvar R est en BCNF et comporte au moins un attribut non-clé (c'est-à-dire un attribut qui ne participe à aucune clé de R), alors cette relvar est en 5NF.

Ce théorème est intéressant lui aussi, car il concerne des situations fréquemment rencontrées. Attention quand même à ce qu'un attribut non-clé soit la conséquence d'une règle de gestion authentique. A défaut, si on dotait par exemple la relvar Administrer d'un attribut technique (attribut d'audit, dans le genre de la date d'insertion du tuple, etc.), une relation telle que celle de la Figure 5.5 ne serait plus décomposable. Cette remarque rejoint celle qui figure en fin du paragraphe 4.8, concernant les clés primaires ne comportant systématiquement qu'un seul attribut.

5.9. Pour conclure avec la normalisation en 5NF

Si l'on se plonge directement dans les écrits de Fagin, Ullman, Delobel, Maier, Zaniolo et autres chercheurs, la normalisation apparaît comme un sujet très formalisé, mathématisé, faisant l'objet de nombreux théorèmes pas toujours faciles à redémontrer, et l'on peut avoir tendance à se désintéresser du sujet. Heureusement, Chris Date présente les choses de façon claire, didactique, précise et strictement conforme à la théorie fournie par ces chercheurs (cf. [Date 2004]). En tout état de cause, nous disposons des outils permettant de contrôler le respect de la 5NF. La méthode qui consiste à chercher les DF, DMV et DJ, ainsi que les clés, puisque celles-ci doivent impliquer ces dépendances, peut être qualifiée d'ascendante, conduisant à décomposer par projection des relvars non normalisées en relvars qui le deviennent. La démarche descendante, à laquelle tient tellement Yves Tabourier (cf. paragraphe 4.13) est évidemment celle qu'il faut privilégier car elle permet d'évacuer à la pelle les dépendances multivaluées et les dépendances de jointure, mais il n'empêche qu'il faut prouver que chaque relvar de base (entité-type ou association-type en Merise) est normalisée, d'où une part de démarche ascendante inévitable.

Si celui qui a la compétence au niveau conceptuel mais ne l'a pas au niveau relationnel, c'est-à-dire le concepteur au sens classique du terme, tient à ce que ses MCD et diagrammes de classes soient irréprochables quant à la normalisation, le mieux — une fois son travail de modélisation en bonne voie d'achèvement — est qu'il s'associe avec celui qui jongle avec les DF, DMV et DJ, afin de procéder à une relecture à deux des règles de gestion des données, pour :

- Débusquer les anomalies potentielles restantes (redondances, anomalies en mise à jour) ;
- Améliorer encore les modèles conceptuels afin que ceux-ci puissent évoluer plus tard sans problème ;
- Simplifier la mise en oeuvre des contraintes d'intégrité.

Comme nous l'avons déjà fait observer, quelque part la normalisation a un côté yoyo...

Une fois le travail de normalisation achevé, on peut être amené à projeter une relvar respectant la 5NF selon deux relvars ou plus (verticalement et/ou horizontalement), mais on entre là dans des considérations relevant pour l'essentiel :

- De la gestion de données intervallaires, disons temporelles et qui intéressent la 6NF (cf. paragraphe 6) ;
- De l'optimisation (cf. paragraphe 1.7) : par exemple, certains attributs de cette relvar sont seulement consultés par une population X d'utilisateurs, tandis que d'autres attributs sont mis à jour par une population différente Y d'utilisateurs, et les contentions qui en résultent sont pénalisantes pour les premiers, ce qui amène en l'occurrence à encore décomposer des structures qui sont déjà en 5NF (dans le sens de la 6NF soit dit en passant).

La normalisation des relvars n'est évidemment pas la panacée et peut être source de dilemmes, quand elle peut faire perdre des règles de gestion, comme on l'a vu avec la BCNF (cf. paragraphe 3.7), ou avec la 4NF (cf. paragraphe 4.9). Heureusement, ce genre de situation est rare et — d'expérience — provoque en général une remise en cause des règles de gestion impliquées.

Bref, normalisons, normalisons, normalisons et les applications s'appuieront sur des bases de données saines.

Et attention à ce que l'on peut glaner sur la Toile ou dans des ouvrages écrits par des « experts » (*sic*), gens peut-être convaincus et pleins de bonne volonté, mais ayant une connaissance insuffisante du sujet, n'ayant pas étudié les bons auteurs et n'ayant manifestement pas véritablement transpiré sur des projets d'une certaine importance, nécessitant une revue rigoureuse et pertinente. Voir ci-dessous les considérations de l'« expert » de service déjà épinglé (cf. paragraphe 2.8)...

5.10. Un petit exercice

A titre d'exercice, je vous invite à recenser et commenter les erreurs contenues dans le texte suivant, ayant pour titre :

« Les Quatrième et Cinquième Formes Normales ne sont pas incluses dans la Troisième Forme Normale »

« Tous les auteurs affirment que ces deux dernières Formes Normales sont incluses dans la Troisième, prolongeant de façon mécanique la série d'inclusion entre les Troisième, Deuxième et Première Formes Normales. Nous allons montrer que cette affirmation est fausse, car elle gêne énormément la compréhension des Quatrième et Cinquième Formes Normales.

S'il est exact que la Troisième Forme Normale est incluse dans la Deuxième et elle-même dans la Première, le simple bon sens permet de voir que la Quatrième n'est pas incluse dans la Troisième.

En effet, les trois premières Formes Normales sont fondées sur les Dépendances Fonctionnelles alors que la Quatrième (comme la Cinquième) s'appuie sur les Dépendances Multivaluées. Or, la Dépendance Fonctionnelle est un cas particulier de la Dépendances Multivaluée. On peut donc considérer qu'elle est incluse dans cette dernière, mais en aucun cas l'inverse. Donc la Quatrième Forme Normale ne peut être incluse dans la Troisième ; tout au plus pourrait-on admettre l'inverse : que la Troisième soit incluse dans la Quatrième. »

... (Au passage, une [critique à l'endroit des chercheurs](#) (cf. paragraphe 2.8), puis reprise des explications.)

« En fait un Tableau en Quatrième Forme Normale n'est pas en Troisième Forme Normale, ni même en Deuxième, tout simplement parce qu'il n'est pas en Première Forme Normale. En effet, la condition essentielle pour qu'un Tableau soit en Première Forme Normale est l'existence d'une Clef.

Une Clef est une Colonne ou un groupe de Colonnes identifiant toutes les autres de façon unique, c'est-à-dire qu'elle est à l'origine des Dépendances Fonctionnelles. Or, dans un tableau en Quatrième Forme Normale, il n'existe aucune Dépendance Fonctionnelle, seulement une Dépendance Multivaluée.

Il n'y a pas de Clef dans un Tableau en Quatrième Forme Normale, et celui-ci n'est donc pas en Première Forme Normale.

Le processus de Normalisation que nous venons de détailler a donc pour but de se prémunir contre la redondance d'informations et contre les difficultés de mise à jour. Lorsqu'il est conduit à son terme, ne subsistent que deux types de Tableaux :

- Des Tableaux en Forme Normale de Boyce-Codd, c'est-à-dire en Troisième Forme Normale avec, éventuellement, Normalisation de Boyce-Codd ;
- Des Tableaux Binaires exprimant chacun une Dépendance Multivaluée Élémentaire. »

Fermez le ban.

Vous pouvez aussi relever le peu qui soit pertinent, mais si vous avez échoué dans le démontage de cet énorme sophisme, prenez la peine de repasser par la case Départ.

N.B. Il est fait mention de la dépendance multivaluée élémentaire : pour l'auteur « la Dépendance Multivaluée est Élémentaire seulement si l'Identifiant est constitué d'une seule Colonne ». Quant à l'Identifiant, il précise qu'il s'agit d'une colonne (ou d'un ensemble de colonnes), dont une autre colonne est dépendante : il s'agit donc du déterminant d'une dépendance multivaluée (ou fonctionnelle), rien à voir l'identifiant au sens Merise du terme.

Bonne chasse.

6. Sixième forme normale

6.1. Introduction

A partir de 1979 et pendant près d'un quart de siècle, nous avons vécu avec la 5NF qui marquait la limite du processus de normalisation par projection/jointure. Et puis la 6NF est venue repousser cette limite. Nous verrons les bienfaits que l'on peut en retirer.

En attendant, reprenons la structure de la relvar F des fournisseurs (paragraphe 2.6) :

F	<u>Four_No</u>	Four_Nom	Statut	Ville
	S1	Salsa	20	Lille
	S2	Jean	10	Paris
	S3	Bernard	30	Paris
	S4	Catherine	20	Lille
	S5	Alain	30	Arles

Figure 6.1 - Relvar F : valeur actuelle de la relvar

Par référence au théorème de Date et Fagin (cf. paragraphe 5.8), cette relvar respecte la 5NF, puisqu'elle a pour unique clé candidate le singleton {Four_No} et parce qu'elle respecte par ailleurs la BCNF. Mais rien n'interdit de pousser au maximum la décomposition de la relvar, sans perte, selon les trois projections suivantes :

F_Nom	<u>Four_No</u>	Four_Nom
	S1	Salsa
	S2	Jean
	S3	Bernard
	S4	Catherine
	S5	Alain

F_Statut	<u>Four_No</u>	Statut
	S1	20
	S2	10
	S3	30
	S4	20
	S5	30

F_Ville	<u>Four_No</u>	Ville
	S1	Lille
	S2	Paris
	S3	Paris
	S4	Lille
	S5	Arles

Figure 6.2 - F = JOIN {F_Nom, F_Statut, F_Ville}

En première approche, une telle décomposition n'offre aucun intérêt et relève même du « Modèle binaire », lequel a été rejeté par Ted Codd [Codd 1990]. Pourtant, si l'on fait référence à l'énoncé de la 6NF, on verra que les trois relvars F_Nom, F_Statut et F_Ville sont conformes à celle-ci, tandis que la relvar F ne l'est pas. Mais Chris Date nous prévient qu'il ne faut pas tenir compte du respect à tout crin de la 6NF, laquelle concerne fondamentalement les relvars dotées d'attributs de type **intervalle** (dont les intervalles de **dates**), donc les relvars à caractère **historique**, qui feront l'objet de toute notre attention. Quant à elle, la relvar F a un caractère intemporel et la décomposer est en l'occurrence inutile.

Notons en passant que la relvar F (qui respecte la 5NF, rappelons-le) vérifie un certain nombre de dépendances de jointure (DJ), dont celle qui correspond à sa décomposition en F_Nom, F_Statut, F_Ville :

$$DJ1 = * \{ \{ \text{Four_No}, \text{Four_Nom} \}, \{ \text{Four_No}, \text{Statut} \}, \{ \text{Four_No}, \text{Ville} \} \}$$

Dépendance de jointure qui n'est pas **triviale**, contrairement à celles-ci (cf. paragraphe 5.4) :

$$DJ2 = * \{ \{ \text{Four_No}, \text{Four_Nom}, \text{Statut}, \text{Ville} \} \},$$

$$DJ3 = * \{ \{ \text{Four_No}, \text{Four_Nom}, \text{Statut}, \text{Ville} \}, \{ \text{Four_No}, \text{Ville} \} \},$$

Et autres de la même farine.

Commençons par définir la 6NF pour découvrir son rôle et son intérêt dans le contexte des données **temporelles**, étudiées de façon complète et très approfondie dans [Date 2003].

6.2. Définition de la sixième forme normale (6NF)

Une relvar est en sixième forme normale (6NF) si et seulement si, quelle que soit la dépendance de jointure à laquelle elle satisfait, cette dépendance est triviale.

Une variante intéressante :

Une relvar est en 6NF si et seulement si elle est en 5NF, elle est de degré n , et n'a aucune clé de degré inférieur à $n - 1$.

La relvar F n'est pas en 6NF puisque, comme on l'a vu, elle vérifie la DJ non triviale :

DJ1 = $*\{\{Four_No, Four_Nom\}, \{Four_No, Statut\}, \{Four_No, Ville\}\}$

Tandis que les relvars F_Nom, F_Statut et F_Ville sont en 6NF, car elles ne vérifient que des DJ triviales, dont voici quelques exemples :

Relvar F_Nom : $*\{\{Four_No, Four_Nom\}\}$
 Relvar F_Nom : $*\{\{Four_No, Four_Nom\}, \{Four_No\}\}$
 Relvar F_Statut : $*\{\{Four_No, Statut\}\}$
 Relvar F_Ville : $*\{\{Four_No, Ville\}\}$

Mais, répétons-le, parce qu'elle respecte la 5NF et ne comporte pas de données de type intervallaire, du point de vue de la normalisation il est inutile de décomposer la relvar F. En revanche, le type intervallaire est fréquent dans les bases données du monde de la finance, de l'assurance, de la retraite, de la distribution, de la sécurité sociale, etc., là où la datation est omniprésente, quand l'historisation des données — entre autres — donne bien des soucis, certes à cause des problèmes de volumétrie, mais aussi du fait de la programmation passablement compliquée qui peut en découler : c'est dans ce contexte que la normalisation en 6NF devient utile, comme on pourra s'en rendre compte dans les paragraphes qui suivent. Nous débordons sensiblement du champ de la normalisation pour pénétrer dans celui de la modélisation des données temporelles, mais nous pensons que la visite vaut largement le détour.

6.3. Relvars à caractère temporel, purement historiques

6.3.1. Un exemple

Prenons le cas des fournisseurs avec lesquels nous avons définitivement cessé de travailler (et qui ne figurent donc pas dans la relvar F ci-dessus). Supposons qu'au moyen d'une relvar ad-hoc nous ayons conservé la trace des changements successifs dont ils ont fait l'objet :


F_Cessation	Four_No	Four_Nom	Statut	Ville	Depuis	Jusqu_au
	C1	Albert & Fils	10	Lille	2005-02-01	2005-02-28
	C1	Albert & Fils	12	Lille	2005-03-01	2005-03-27
	C1	Albert & Fils	12	Paris	2005-03-28	2005-05-01
	C1	Albert & Fils	14	Paris	2005-06-12	2005-07-14
	C1	Albert & Fils	14	Lyon	2005-07-15	2005-09-30
	C1	Albert & Cie	14	Lyon	2005-10-01	2005-12-02
	C1	Albert & Cie	14	Nantes	2005-12-03	2006-04-16
	C1	Albert & Cie	14	Caen	2006-04-17	2006-05-01
	C1	Albert & Cie	25	Lille	2006-05-02	2006-07-14
	C1	Albert	25	Lille	2006-07-15	2007-12-01
	C2	Bernard	15	Amiens	2005-02-01	2007-02-15
	C3	Mimile	10	Paris	2005-07-01	2005-08-30

Figure 6.3 - Relvar F_Cessation : fournisseurs avec lesquels nous ne travaillons plus

Ce que raconte cette relvar : par exemple, que le fournisseur Albert & Fils a commencé à travailler avec nous le 1er février 2005, il était alors localisé à Lille et avait un statut valant 10. Le 1er mars de la même année, son statut est passé à 12. Le 28 mars suivant, il a déménagé à Paris. Du 2 mai au 11 juin il a cessé de travailler avec nous. Le 12 juin, il a repris son activité et son statut est passé à 14. Le 15 juillet, il a déménagé à Lyon. Le 1er octobre, le nom de sa raison sociale a changé, pour devenir Albert et Cie. Le 3 décembre on le retrouve à Nantes et le 17 avril de l'année suivante à Caen (il a la bougeotte). Le 2 mai suivant, on le retrouve à Lille et son statut est passé à 25. Le 15 juillet, sa raison sociale a encore changé de nom, pour devenir Albert. A compter du 2 décembre 2007, nous avons cessé de travailler avec ce fournisseur.

La relvar F_Cessation peut prendre bien d'autres formes, mais commençons avec la structure que nous avons décrite ci-dessus. Tout d'abord, cette relvar est en 5NF. En effet :

- Elle a pour clés candidates les paires {Four_No, Depuis} et {Four_No, Jusqu_au}
 - Chaque projection participant à chaque dépendance de jointure contient les attributs composant ces clés :
- DJ1 : *{{Four_No, Depuis, Four_Nom}, {Four_No, Depuis, Statut}, {Four_No, Depuis, Ville}, {Four_No, Jusqu_au}}
- DJ2 : *{{Four_No, Jusqu_au, Four_Nom}, {Four_No, Jusqu_au, Statut}, {Four_No, Jusqu_au, Ville}, {Four_No, Depuis}}
- DJ3 : *{{Four_No, Depuis, Four_Nom, Statut}, {Four_No, Depuis, Ville}, {Four_No, Jusqu_au}}
- DJ4 : *{{Four_No, Depuis, Four_Nom}, {Four_No, Depuis, Statut Ville}, {Four_No, Jusqu_au}}
- DJ5 : *{{Four_No, Four_Nom, Statut, Ville, Depuis, Jusqu_au}}
- Etc.

 (On peut encore facilement montrer que la relvar respecte la BCNF, puis mettre à profit le 2e théorème mentionné au paragraphe 5.8 pour montrer que la 5NF est respectée elle aussi).

Mais la relvar n'est pas en 6NF, parce que parmi ces dépendances de jointure, seule DJ5 est triviale. Par exemple, DJ1 ne l'est pas, puisque la projection {Four_No, Depuis, Four_Nom} est un des éléments de cette DJ, sans que l'attribut Ville en fasse partie. Et constat pas inintéressant, on se rend compte que la relvar est fortement **redondante** : on apprend ainsi plus que de raison que, à une époque, le fournisseur C1 s'est appelé Albert & Fils, et nous devons chercher à éliminer ce genre de redondance qui pollue bien des relvars à caractère historique.

Comme dans le cas de la relvar F, pleins de bonne volonté, nous pourrions décomposer F_Cessation en quatre relvars, appelons-les F_N, F_S, F_V et F_D, respectant chacune la 6NF :

F_N	<u>Four_No</u>	Four_Nom	<u>Depuis</u>
	C1	Albert & Fils	2005-02-01
	C1	Albert & Fils	2005-03-01
	C1	Albert & Fils	2005-03-28
	C1	Albert & Fils	2005-06-12
	C1	Albert & Fils	2005-07-15
	C1	Albert & Cie	2005-10-01
	C1	Albert & Cie	2005-12-03
	C1	Albert & Cie	2006-04-17
	C1	Albert & Cie	2006-05-02
	C1	Albert	2006-07-15
	C2	Bernard	2005-02-01
	C3	Mimile	2005-07-01

F_S	<u>Four_No</u>	Statut	<u>Depuis</u>
	C1	10	2005-02-01
	C1	12	2005-03-01
	C1	12	2005-03-28
	C1	14	2005-06-12
	C1	14	2005-07-15
	C1	14	2005-10-01
	C1	14	2005-12-03
	C1	14	2006-04-17
	C1	25	2006-05-02
	C1	25	2006-07-15
	C2	15	2005-02-01
	C3	10	2005-08-30

Figure 6.4 - Relvar F_Cessation : Décomposition de la relvar

F_V	<u>Four_No</u>	Ville	<u>Depuis</u>	F_D	<u>Four_No</u>	<u>Depuis</u>	<u>Jusqu_au</u>
	C1	Lille	2005-02-01		C1	2005-02-01	2005-02-28
	C1	Lille	2005-03-01		C1	2005-03-01	2005-03-27
	C1	Paris	2005-03-28		C1	2005-03-28	2005-05-01
	C1	Paris	2005-06-12		C1	2005-06-12	2005-07-14
	C1	Lyon	2005-07-15		C1	2005-07-15	2005-09-30
	C1	Lyon	2005-10-01		C1	2005-10-01	2005-12-02
	C1	Nantes	2005-12-03		C1	2005-12-03	2006-04-16
	C1	Caen	2006-04-17		C1	2006-04-17	2006-05-01
	C1	Lille	2006-05-02		C1	2006-05-02	2006-07-14
	C1	Lille	2006-07-15		C1	2006-07-15	2007-12-01
	C2	Amiens	2005-02-01		C2	2002-03-31	2007-02-15
	C3	Paris	2005-07-01		C3	2004-07-01	2005-08-30

Figure 6.5 - Relvar F_Cessation : Décomposition de la relvar (suite)

Mais la redondance est toujours là, et qui plus est, on a maintenant quatre relvars sur les bras au lieu d'une : piètre résultat, même si ces nouvelles relvars sont en 6NF. Néanmoins, avant de conclure un peu hâtivement que la 6NF ne sert finalement à rien et qu'il était parfaitement inutile de l'inventer, poussons plus avant, en étudiant de plus près le caractère intervallaire de la relvar F_Cessation. Intéressons-nous à l'esprit de la 6NF plutôt qu'en rester à la lettre.

6.3.2. Typage des intervalles, opérateurs

La relvar F_Cessation n'est donc pas en 6NF. Pour normaliser efficacement, c'est-à-dire éliminer les **redondances**, il est préférable, pour commencer, d'en passer par une représentation intervallaire des dates de début et de fin, et bénéficier ainsi des opérateurs dédiés. Remplaçons donc la paire d'attributs {Depuis, Jusqu_au} par un attribut unique, que nous appellerons Durant :

F_Cessation	<u>Four_No</u>	Four_Nom	Statut	Ville	<u>Durant</u>
	C1	Albert & Fils	10	Lille	[2005-02-01:2005-02-28]
	C1	Albert & Fils	12	Lille	[2005-03-01:2005-03-27]
	C1	Albert & Fils	12	Paris	[2005-03-28:2005-05-01]
	C1	Albert & Fils	14	Paris	[2005-06-12:2005-07-14]
	C1	Albert & Fils	14	Lyon	[2005-07-15:2005-09-30]
	C1	Albert & Cie	14	Lyon	[2005-10-01:2005-12-02]
	C1	Albert & Cie	14	Nantes	[2005-12-03:2006-04-16]
	C1	Albert & Cie	14	Caen	[2006-04-17:2006-05-01]
	C1	Albert & Cie	25	Lille	[2006-05-02:2006-07-14]
	C1	Albert	25	Lille	[2006-07-15:2007-12-01]
	C2	Bernard	15	Amiens	[2005-02-01:2007-02-15]
	C3	Mimile	10	Paris	[2005-07-01:2005-08-30]

Figure 6.6 - Relvar F_Cessation : Représentation intervallaire

La déclaration de la relvar F_Cessation pourrait être la suivante (en Tutorial D) :

```

VAR F_Cessation
  BASE RELATION {Four_No CHAR,
                  Four_Nom CHAR,
                  Statut INTEGER,
                  Ville CHAR,
                  Durant INTERVAL_DATE}
  KEY {Four_No, Durant} ;

```

Où l'attribut *Durant* est du type `INTERVAL_DATE` dont l'étude n'est pas *a priori* l'objet de cet article, mais vu son importance, il faut en dire un mot. Ce type (et plus généralement tout type `INTERVAL_TRUC`) doit permettre de comparer, manipuler des intervalles. Ainsi, doit-on déjà disposer d'un opérateur de sélection :

`INTERVAL_DATE ([deb:fin])`

deb et *fin* étant des expressions de type `DATE`.

Dans notre exemple, durant l'intervalle $i = [2005-02-01:2005-02-28]$, le fournisseur C1 s'est appelé Albert & Fils, il avait le statut 10 et était localisé à Lille.

Le type `INTERVAL_TRUC` (en particulier le type `INTERVAL_DATE`) présente, les caractéristiques suivantes :

- `INTERVAL` est un générateur de type, au même titre que le générateur de type `RELATION` qu'on a utilisé pour déclarer la relvar `F_Cessation` (cf. paragraphe A.2 en annexe). Le type `INTERVAL_TRUC` est obtenu par invocation du générateur de type `INTERVAL` (cf. [Date 2004], chapitre 23 « Temporal Databases », ou encore [Date 2004], chapitre 5 « Point Types and Interval Types »).

Un intervalle (ou valeur d'intervalle) i du type `INTERVAL_TRUC` est une valeur scalaire (« atomique » pour faire court) pour laquelle ont été définis deux opérateurs monadiques, `BEGIN` et `END`, et un opérateur dyadique, \in , tels que :

1. `BEGIN (i)` et `END (i)` renvoient une valeur de type `TRUC` (donc une date dans le cas du type `DATE`).
2. `BEGIN (i) ≤ END (i)`.
3. Si p est une valeur du type `TRUC`, alors $p \in i$ est vrai si et seulement si `BEGIN (i) ≤ p` et $p \leq \text{END (i)}$ sont vrais en même temps.

Par exemple (en Tutorial D), pour savoir quel était le statut du fournisseur C1 au 15 septembre 2005 :

`(F_Cessation WHERE Four_No = 'C1' AND DATE ('2005-09-15') ∈ Durant) {Statut}`

- `TRUC` (et en particulier `DATE`) est le **type de point** utilisé en relation avec le type d'intervalle `INTERVAL_TRUC` : un intervalle est une succession de points (valeurs discrètes) et dans le cas qui nous intéresse, ces points se succèdent dans le temps, c'est-à-dire que ce sont des dates (disons du calendrier). Dans d'autres situations, les points peuvent être des températures, des tranches d'imposition, des paliers de remise, etc.

Dans le cas général, le type de point `TRUC` (et donc le type de point `DATE` en particulier) possède certaines propriétés :

- Il est **totalelement ordonné** : si $v1$ et $v2$ sont deux valeurs du même type de point `TRUC` et si l'expression « $v2 > v1$ » est vraie, alors l'expression « $v1 > v2$ » est fausse. Plus généralement, on doit disposer des opérateurs de comparaison habituels, « = », « > », etc. Cela vaut évidemment dans le cas particulier du type de point `DATE` : le système doit nous permettre de répondre par exemple à la question « Quels fournisseurs ont commencé à travailler avec nous avant le fournisseur C3 ? »
- Il est doté d'opérateurs permettant de retrouver les plus petites et les plus grandes valeurs pour ce type de point : `FIRST_TRUC`, `LAST_TRUC`, le prédécesseur (`PRIOR_TRUC`) et le successeur (`NEXT_TRUC`) d'un point. Dans le cas du type de point `DATE` : `FIRST_DATE`, `LAST_DATE` (en particulier `LAST_DATE ()` qui correspond à la « fin des temps »), `PRIOR_DATE`, `NEXT_DATE`.

Mentionnons encore la panoplie des opérateurs de comparaison connus collectivement comme étant les opérateurs de Allen (*Allen's operators*), permettant de savoir si deux intervalles sont contigus (opérateur `MEETS`), se recouvrent (opérateur `OVERLAPS`), etc.

6.3.3. Opérateurs **PACK** et **UNPACK**



Complétons notre boîte à outils avec les opérateurs `PACK` et `UNPACK`.

Ces opérateurs sont à la 6NF ce que le poussé et le tiré sont au soufflet d'un accordéon, et constituent une des clés (clefs en l'occurrence...) d'une authentique (et efficace) normalisation en 6NF.

A noter que `PACK` et `UNPACK` ne sont pas des opérateurs primitifs, ce ne sont que des combinaisons d'opérateurs relationnels existants (à ce sujet, se reporter à [Date 2003], chapitre 8 « The `PACK` and `UNPACK` Operators »).

PACK

L'opérateur PACK doit nous permettre de regrouper des intervalles d'une relvar R. Sa syntaxe est la suivante :

PACK R ON A

Où R est une expression relationnelle et A un attribut de type intervallaire, appartenant à l'en-tête de R.

Soit deux intervalles $i1$ et $i2$. Leur regroupement est possible dans les conditions suivantes :

- Soit on vérifie : $BEGIN(i1) \leq END(i2)$ et $BEGIN(i2) \leq END(i1)$, ce qui se produit quand $i1$ et $i2$ se recouvrent totalement ou partiellement (l'opérateur de Allen OVERLAPS renvoie alors la valeur *vrai*).

Prenons un exemple, tiré de [Date 2004], dans lequel la relvar R comporte deux attributs S# (codes des fournisseurs) et DURING (Périodes d'activité). L'attribut DURING est du type INTERVAL_DATE et les dates sont symbolisées par $d01$, $d02$, etc. Réalisons l'opération PACK suivante :

PACK R ON DURING

R	S#	DURING	Au résultat :	S#	DURING
	S1	[d04:d10]		S1	[d04:d10]
	S1	[d05:d10]		S2	[d02:d04]
	S1	[d09:d10]		S2	[d08:d10]
	S1	[d06:d10]		S3	[d08:d10]
	S2	[d02:d04]		S4	[d04:d10]
	S2	[d08:d10]			
	S2	[d03:d03]			
	S2	[d09:d10]			
	S3	[d08:d10]			
	S4	[d06:d09]			
	S4	[d04:d08]			
	S4	[d05:d10]			

Figure 6.7 - Regroupement des intervalles par PACK, dans l'esprit de l'opérateur OVERLAPS

- Soit on vérifie : $END(i1+1) = BEGIN(i2)$ ou $END(i2+1) = BEGIN(i1)$, c'est-à-dire que $i1$ et $i2$ sont contigus (l'opérateur de Allen MEETS renvoie alors la valeur *vrai*). Ainsi, pour la paire {Four_No, Four_Nom} de la relvar F_Cessation, dans le cas du couple <'C1', 'Albert & Fils'>, on vérifie les égalités :

$END([2005-02-01:2005-02-28] + 1) = BEGIN([2005-03-01:2005-03-27])$;
 $END([2005-03-01:2005-03-27] + 1) = BEGIN([2005-03-28:2005-05-01])$;
 $END([2005-06-12:2005-07-14] + 1) = BEGIN([2005-07-15:2005-09-30])$.

En conséquence de quoi, on a tout à gagner à regrouper les intervalles pendant lesquels le nom de la raison sociale de C1 fut 'Albert & Fils' (Figure 6.6), pour condenser au maximum l'information :

[2005-02-01:2005-05-01] et [2005-06-12:2005-09-30]

De la même façon, le nom de la raison sociale de C1 fut 'Albert & Cie' pendant l'intervalle :

[2005-10-01:2006-07-14]

Puis 'Albert' pendant l'intervalle :

[2006-07-15:2007-12-01]

Si donc on écrit (ce que l'on appelle parfois une *projection temporelle*) :

PACK F_Cessation {Four_No, Four_Nom, Durant} **ON** Durant

Concernant le fournisseur C1, l'image du résultat est alors la suivante (appelons FND la relvar correspondante) :

FND	<u>Four_No</u>	Four_Nom	<u>Durant</u>
	C1	Albert & Fils	[2005-02-01:2005-05-01]
	C1	Albert & Fils	[2005-06-12:2005-09-30]
	C1	Albert & Cie	[2005-10-01:2006-07-14]
	C1	Albert	[2006-07-15:2007-12-01]

Figure 6.8 - PACK F_Cessation {Four_No, Four_Nom, Durant} **ON** Durant

On observera que la redondance a singulièrement diminué concernant l'attribut Four_Nom, on peut même dire qu'elle a disparu. Si l'on fait subir la même cure d'amaigrissement à l'attribut Statut :

PACK F_Cessation {Four_No, Statut, Durant} **ON** Durant

Toujours pour le fournisseur C1, l'image du résultat est la suivante (appelons FSD la relvar correspondante) :

FSD	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]
	C1	12	[2005-03-01:2005-05-01]
	C1	14	[2005-06-12:2006-05-01]
	C1	25	[2006-05-02:2007-12-01]

Figure 6.9 - PACK F_Cessation {Four_No, Statut, Durant} **ON** Durant

Dans le cas de l'attribut Ville :

PACK F_Cessation {Four_No, Ville, Durant} **ON** Durant

Toujours pour le fournisseur C1, l'image du résultat est la suivante (appelons FVD la relvar correspondante) :

FVD	<u>Four_No</u>	Ville	<u>Durant</u>
	C1	Lille	[2005-02-01:2005-03-27]
	C1	Paris	[2005-03-28:2005-05-01]
	C1	Paris	[2005-06-12:2005-07-14]
	C1	Lyon	[2005-07-15:2005-12-02]
	C1	Nantes	[2005-12-03:2006-04-16]
	C1	Caen	[2006-04-17:2006-05-01]
	C1	Lille	[2006-05-02:2007-12-01]

Figure 6.10 - PACK F_Cessation {Four_No, Ville, Durant} **ON** Durant

On peut subodorer l'influence qu'aura l'opérateur PACK dans le travail de normalisation, mais à lui seul il ne suffit pas, il devra le plus souvent binômer avec son inverse, UNPACK (au moins dans le cadre théorique).

Remarque. Si la question suivante était posée : « Quand le fournisseur C1 a-t-il travaillé pour nous ? », on pourrait *a priori* interroger n'importe laquelle des trois relvars FND, FSD, FVD. Mais, pour éviter tout choix arbitraire et surtout pour en plus

tenir compte des données non historisées (cf. paragraphe 6.4), il sera préférable de mettre en oeuvre une relvar supplémentaire, appelons-la FDD, résultat de l'opération :

PACK F_Cessation {Four_No, Durant} **ON** Durant

Concernant le fournisseur C1, l'image du résultat est la suivante :

FDD	<u>Four_No</u>	<u>Durant</u>
	C1	[2005-02-01:2005-05-01]
	C1	[2005-06-12:2007-12-01]

Figure 6.11 - PACK F_Cessation {Four_No, Durant} ON Durant

UNPACK

L'opérateur UNPACK a pour fonction de nous permettre de dégroupier un intervalle en intervalles élémentaires, non décomposables. Un intervalle *i* est élémentaire si et seulement si BEGIN (*i*) = END (*i*).

Pour parvenir à dégroupier les intervalles, on utilise cet opérateur dont la syntaxe est la suivante :

UNPACK R **ON** A

Où R est une expression relationnelle et A un attribut de type intervallaire, appartenant à l'en-tête de R.

Pour reprendre la relvar F_Cessation (Figure 6.6), écrivons :

UNPACK F_Cessation {Four_No, Four_Nom, Durant} **ON** Durant

Le résultat produit est le suivant :

<u>Four_No</u>	Four_Nom	<u>Durant</u>
C1	Albert & Fils	[2005-02-01:2005-02-01]
C1	Albert & Fils	[2005-02-02:2005-02-02]
C1	Albert & Fils	[2005-02-03:2005-02-03]
...
C1	Albert & Fils	[2005-05-01:2005-05-01]
C1	Albert & Fils	[2005-06-12:2005-06-12]
...
C1	Albert & Fils	[2005-09-29:2005-09-29]
C1	Albert & Fils	[2005-09-30:2005-09-30]
C1	Albert & Cie	[2005-10-01:2005-10-01]
C1	Albert & Cie	[2005-10-02:2005-10-02]
C1	Albert & Cie	[2005-10-03:2005-10-03]
...
C1	Albert & Cie	[2006-07-13:2006-07-13]
C1	Albert & Cie	[2006-07-14:2006-07-14]
C1	Albert	[2006-07-15:2006-07-15]
C1	Albert	[2006-07-16:2006-07-16]
C1	Albert	[2006-07-17:2006-07-17]
...
C1	Albert	[2007-11-30:2007-11-30]
C1	Albert	[2007-12-01:2007-12-01]
...

Figure 6.12 - UNPACK F_Cessation {Four_No, Four_Nom, Durant} ON {Durant}

A cette occasion, on apprend 201 fois que le fournisseur C1 s'est d'abord appelé Albert & Fils, puis 287 fois qu'il s'est appelé Albert & Cie et enfin 505 fois Albert. Ce qui fait qu'à partir d'une (valeur de) relation F_Cessation requérant une dizaine de tuples pour le fournisseur C1 (cf. Figure 6.6), par UNPACK on produit près d'un millier de tuples, tandis que par PACK le résultat n'est que de quatre tuples (Figure 6.8).

Mais alors, quel intérêt présente l'opérateur UNPACK si son utilisation conduit à une telle inflation, à une consommation *a priori* énorme de temps et d'espace ? Disons déjà que d'un point de vue théorique, contexte dans lequel les considérations basement matérielles n'interviennent pas, le principe général est de pouvoir utiliser les opérateurs relationnels (RESTRICT, PROJECT, JOIN, UNION, etc.) selon le mode habituel, intemporel, ce qui nécessite de procéder en trois temps :

- 1) Transformer par UNPACK un intervalle quelconque en intervalles élémentaires,
- 2) Effectuer l'opération relationnelle voulue, dans laquelle sont impliqués ces intervalles élémentaires,
- 3) Regrouper par PACK les intervalles élémentaires contigus.

Au passage, je cite et traduis [Date 2003], paragraphe A.8 :

« L'opérateur UNPACK est un composant conceptuel qui joue un rôle crucial dans notre approche de la gestion des données intervallaires en général, et des données temporelles en particulier. UNPACK a été très critiqué dans la littérature, au motif qu'il ne peut en résulter qu'une piètre performance. [...] Mais ça n'est qu'un composant conceptuel. Du moment qu'on peut s'en passer, nous ne tenons pas à ce que les opérations de décompression aient physiquement lieu. »

Il est entendu que, dans ce genre d'opération, le but n'est pas de gâcher inutilement des ressources en espace et en temps qui ne peuvent qu'être considérables, pour finir à tout coup par une annulation de la tâche. En ce sens, [Date 2003] montre quelle doit être l'approche d'un optimisateur dans l'exploitation de UNPACK (Cf. son appendice A : « Implementation Considerations »). La matérialisation du résultat n'est évidemment à effectuer qu'à la demande expresse de l'utilisateur.

6.3.4. Opérateurs U_PROJECT et U_JOIN

Revenons à la 6NF : elle se situe dans le prolongement de celles qui l'ont précédée, de la deuxième à la cinquième, c'est-à-dire qu'à l'aide du marteau PROJECT on « casse » une relation *r* non normalisée en deux (ou plusieurs) relations, que l'on sait « recoller » par JOIN pour retrouver très exactement la relation *r* (principe de la décomposition sans perte de données, *nonloss decomposition*).

Pour le moment, nous savons que la relvar F_Cessation est en 5NF, c'est-à-dire égale à la jointure de ses projections :

```
F_Cessation = F_Cessation {Four_No, Four_Nom, Durant}
              JOIN
              F_Cessation {Four_No, Statut, Durant}
              JOIN
              F_Cessation {Four_No, Ville, Durant} ;
```

Nous savons aussi que ces projections sont en 6NF, car non décomposables par projection et ne comportent donc que des DJ triviales :

```
DJ1 : *{{Four_No, Four_Nom, Durant}}
DJ2 : *{{Four_No, Statut, Durant}}
DJ3 : *{{Four_No, Ville, Durant}}
```

Mais nous savons également que la relvar F_Cessation n'est pas en 6NF, car elle comporte la DJ non triviale :

```
*{{Four_No, Four_Nom, Durant}, {Four_No, Statut, Durant}, {Four_No, Ville, Durant}}
```

Ce dont nous avons besoin, c'est d'un mécanisme qui permette, par projection, de décomposer F_Cessation en relvars « compressées », à savoir FND, FSD et FVD, FDD (cf. Figure 6.8 et suivantes) et, toujours selon le principe de la décomposition sans perte, d'avoir l'assurance que par leur jointure on retrouve très exactement F_Cessation.

Cela passe en fait par la **généralisation** des opérateurs de l'algèbre relationnelle (rien que ça !), ceux que nous connaissions jusqu'ici (RESTRICT, PROJECT, JOIN, UNION, etc.) n'étant plus que des cas particuliers des opérateurs généraux (RENAME étant le seul opérateur non concerné par la généralisation).

Étant donné un opérateur relationnel traditionnel symbolisé par OPERATEUR, par généralisation celui-ci donne lieu à l'opérateur U_OPERATEUR, lequel au plan théorique fonctionne comme on l'a vu ci-dessus :

- 1) Décomposition par UNPACK d'un intervalle quelconque en intervalles élémentaires,
- 2) Exécution de l'opération relationnelle traditionnelle voulue, à l'aide de l'opérateur OPERATEUR, et dans laquelle sont impliqués ces intervalles élémentaires,
- 3) Regroupement par PACK des intervalles élémentaires contigus.

Si l'on passe du plan théorique au plan pratique, on observera que pour des raisons évidentes de performance, un moteur relationnel se doit de court-circuiter l'opérateur UNPACK tant que faire se peut (cf. par exemple le cas de l'opérateur U_PROJECT).

Quoi qu'il en soit, dans le cadre de la normalisation par projection/jointure, nous nous limiterons aux deux opérateurs U_PROJECT et U_JOIN (U pour USING, ou UNPACKING, au choix).

U_PROJECT

La « U_projection » est notée :

USING (ACL) ◀ R {BCL} ▶

Expression dans laquelle R désigne une expression relationnelle et BCL la liste de noms d'attributs de l'en-tête de R qui sont utilisés pour effectuer la projection traditionnelle R {BCL} (cf. Annexe B). ACL représente une liste de noms d'attributs (*Attribute Comma List*) de l'en-tête de R, qui sont de type intervallaire et font partie de la liste BCL.

Cette expression est équivalente à :

PACK ((**UNPACK** R **ON** (ACL)) {BCL}) **ON** (ACL)

Expression qui peut être réduite à celle-ci, dans laquelle on court-circuite l'opérateur UNPACK (et dans ce cas on évite la manipulation des intervalles élémentaires pour eux-mêmes) :

PACK (R {BCL}) **ON** (ACL)

Ainsi, l'exemple que l'on a vu ci-dessus :

PACK F_Cessation {Four_No, Four_Nom, Durant} **ON** Durant

est la U_Projection dont le résultat FND est représenté dans la Figure 6.8 :

USING Durant ◀ F_Cessation {Four_No, Four_Nom, Durant} ▶

Notes :

1. Si la liste ACL ne comporte qu'un seul nom d'attribut, on peut supprimer les parenthèses, comme on vient de le faire avec l'attribut Durant.
2. Si la liste ACL est vide, on retrouve la projection traditionnelle.

U_JOIN

La « U_jointure » est notée :

USING (ACL) ◀ R1 **JOIN** R2 ▶

Expression relationnelle dans laquelle R1 **JOIN** R2 représente la jointure (naturelle) traditionnelle (cf. Annexe B) et ACL représente une liste de noms d'attributs de l'en-tête de R1 et de celui de R2, attributs qui sont de type intervallaire.

Cette expression est un raccourci pour celle-ci :

PACK ((**UNPACK** R1 **ON** (ACL))
JOIN
(**UNPACK** R2 **ON** (ACL)))
ON (ACL)

On retrouve une fois de plus le grand principe du fonctionnement du tandem PACK/UNPACK (sachant qu'en réalité l'étape UNPACK peut, là encore, être court-circuitée) :

- 1) Décomposition par UNPACK d'un intervalle quelconque en intervalles élémentaires,
- 2) Exécution de l'opération relationnelle traditionnelle voulue, dans laquelle sont impliqués ces intervalles élémentaires,
- 3) Regroupement par PACK des intervalles élémentaires contigus.

Ainsi, la U_jointure des U_projections FND et FSD (cf. Figure 6.8 et Figure 6.9) donnera lieu au résultat suivant concernant le fournisseur C1 (appelons-le FNSD) :

FNSD	<u>Four_No</u>	Four_Nom	Statut	<u>Durant</u>
	C1	Albert & Fils	10	[2005-02-01:2005-02-28]
	C1	Albert & Fils	12	[2005-03-01:2005-05-01]
	C1	Albert & Fils	14	[2005-06-12:2005-09-30]
	C1	Albert & Cie	14	[2005-10-01:2006-05-01]
	C1	Albert & Cie	25	[2006-05-02:2006-07-14]
	C1	Albert	25	[2006-07-15:2007-12-01]

Figure 6.13 - U_jointure des U_projections FND et FSD

A son tour, la U_jointure de FNSD et de FVD redonnera la relvar initiale F_Cessation.

Grâce aux opérateurs U_PROJECT et U_JOIN, on a ainsi pu réaliser une U_projection / U_jointure sans perte de données, faisant que l'on peut remplacer la relvar F_Cessation par le trio FND, FSD, FVD. Avant d'en reparler en relation avec la 6NF, reste à évoquer les opérateurs de comparaison relationnelle généralisés et la dépendance de jointure généralisée.

Note : La U_jointure est associative, et encore une fois l'optimiseur peut jouer là-dessus, ainsi que sur la concision de l'expression générale **USING** (ACL) ◀ R1 JOIN R2 ▶, qui autorise une grande liberté dans la façon de construire les algorithmes pour implémenter l'opérateur.

6.3.5. Opérateurs de comparaison relationnelle généralisés

En algèbre relationnelle, deux relations R1 et R2 peuvent être comparées à l'aide des opérateurs ensemblistes de comparaison relationnelle : « = » (R1 est égale à R2), « ≠ » (R1 n'est pas égale à R2), « ⊆ » (R1 est incluse dans R2), « ⊂ » (R1 est strictement incluse dans R2), « ⊇ » (R1 inclut R2), « ⊃ » (R1 inclut strictement R2). Le résultat de la comparaison est une valeur de vérité : *vrai*, *faux*.

Par exemple, si l'on se reporte à la Figure 2.6, pour savoir si la projection des fournisseurs sur l'attribut Ville est égale à la projection des pièces sur ce même attribut, l'expression relationnelle suivante doit être vraie :

F {Ville} = P {Ville} /* Si *vrai*, alors réponse affirmative */

Ou encore, pour savoir si certains fournisseurs n'ont pas livré de pièces :

F {Four_No} ⊃ FP {Four_No} /* Si *vrai*, alors réponse affirmative */

Ce qui vaut pour les opérateurs JOIN, PROJECT, etc. à savoir, comme on l'a vu, qu'ils sont généralisables en U_JOIN, U_PROJECT, etc., vaut également pour les opérateurs de comparaison relationnelle. Ainsi, la contrepartie de l'opérateur de comparaison « = » est l'opérateur « U_ = ». Plus précisément, l'expression :

USING (ACL) ◀ R1 = R2 ▶

est une abréviation pour :

(UNPACK R1 ON (ACL)) = (UNPACK R2 ON (ACL))

Où chaque attribut mentionné dans ACL est de type intervalle et figure à la fois dans R1 et R2. A noter que la participation de l'opérateur PACK est inutile, puisque le résultat n'est pas une relation mais seulement une valeur de vérité, *vrai/faux*.

On dit que les relations R1 et R2 sont **équivalentes** par rapport à ACL. Pour reprendre un exemple de [Date 2003], étant données les relations :

R1	<table><tr><th>A</th></tr><tr><td>[d01:d03]</td></tr><tr><td>[d02:d05]</td></tr><tr><td>[d04:d04]</td></tr></table>	A	[d01:d03]	[d02:d05]	[d04:d04]	R2	<table><tr><th>A</th></tr><tr><td>[d01:d02]</td></tr><tr><td>[d03:d05]</td></tr></table>	A	[d01:d02]	[d03:d05]
A										
[d01:d03]										
[d02:d05]										
[d04:d04]										
A										
[d01:d02]										
[d03:d05]										

« R1 = R2 » est *faux*, mais « **USING** A ◀ R1 = R2 ▶ » est *vrai*.

6.3.6. Dépendance de jointure généralisée

A son tour, la dépendance de jointure fait l'objet d'une généralisation.

Soit R une relvar, A, B, ..., Z des sous-ensembles d'attributs de l'en-tête de R, ACL une liste d'attributs de type intervalle de l'en-tête de R, et soit R' la U_jointure des U_projections de R sur A, B, ..., Z.

R vérifie la **dépendance de jointure** (généralisée)

USING (ACL) *{A, B, ..., Z}

si et seulement si

USING (ACL) ◀ R = R' ▶

est *vrai*. Considérons à nouveau la relvar F_Cessation de la Figure 6.6 :

F_Cessation	<u>Four_No</u>	Four_Nom	Statut	Ville	<u>Durant</u>
	C1	Albert & Fils	10	Lille	[2005-02-01:2005-02-28]
	C1	Albert & Fils	12	Lille	[2005-03-01:2005-03-27]
	C1	Albert & Fils	12	Paris	[2005-03-28:2005-05-01]
	C1	Albert & Fils	14	Paris	[2005-06-12:2005-07-14]
	C1	Albert & Fils	14	Lyon	[2005-07-15:2005-09-30]
	C1	Albert & Cie	14	Lyon	[2005-10-01:2005-12-02]
	C1	Albert & Cie	14	Nantes	[2005-12-03:2006-04-16]
	C1	Albert & Cie	14	Caen	[2006-04-17:2006-05-01]
	C1	Albert & Cie	25	Lille	[2006-05-02:2006-07-14]
	C1	Albert	25	Lille	[2006-07-15:2007-12-01]
	C2	Bernard	15	Amiens	[2005-02-01:2007-02-15]
	C3	Mimile	10	Paris	[2005-07-01:2005-08-30]

Figure 6.14- Relvar F_Cessation : Représentation intervallaire (bis)

Cette relation a pour U_projections les relvars FND, FSD et FVD (cf. Figure 6.8 et suivantes). On a vu au paragraphe 6.3.4 que la U_jointure de ces U_projections permet de retrouver F_Cessation (U_projection et U_jointure sans perte de données).

Ainsi, R représentant F_Cessation et R' la U_jointure de ses U_projections, l'expression

USING Durant ◀ R = R' ▶

est vraie. Autrement dit, F_Cessation vérifie la dépendance de jointure généralisée

USING Durant *{FND, FSD, FVD}

Et cette dépendance de jointure est non triviale. C'est ici l'occasion de reparler de la 6NF.

6.3.7. Retour sur la 6NF, place à la concision

Remarque préalable

Reprenons la définition de la 6NF :

Une relvar est en sixième forme normale (6NF) si et seulement si, quelle que soit la dépendance de jointure à laquelle elle satisfait, cette dépendance est triviale.

La relvar F_Cessation (cf. Figure 6.14) n'est pas en 6NF, puisqu'on a vu qu'elle vérifie la DJ non triviale :

USING Durant *{FND, FSD, FVD}

Il y a deux méthodes pour normaliser en 6NF une telle relvar comportant des attributs intervallaires, la bonne et la mauvaise.

- La bonne méthode consiste donc à décomposer F_Cessation à coups de U_projections et ainsi éliminer les redondances non triviales, pour produire des relvars à l'image de FND, FSD et FVD (cf. Figure 6.8 et suivantes) ;
- La mauvaise méthode consiste à décomposer F_Cessation à coups de projections traditionnelles, sans tenir compte de la dimension temporelle des données, faisant que l'on conserve les redondances (cf. Figure 6.3 et suivantes). En outre, la relvar F_D (cf. Figure 6.5) et reprise ci-dessous (cf. Figure 6.15), viole la 2e des 9 règles impératives (**Nine Requirements**) — que j'appellerai *Impératifs LDD* par référence à leurs auteurs —, énoncés dans [Date 2003], dont la programmation sous forme de contraintes ne pose pas de problème particulier, et auxquels doit se conformer une base de données temporelle. Ce 2e impératif a pour objet de nous contraindre à grouper les intervalles consécutifs, éliminer l'**atomisation** en intervalles inutilement trop fins et s'énonce ainsi :

2e impératif LDD



Si la base de données montre que tel fournisseur est sous contrat à j et j+1, alors il y a exactement un tuple exprimant ce fait.

Or, dans la relation F_D (recopiée ci-dessous), un premier tuple montre que le fournisseur C1 a travaillé pour nous le 28/02/2005 et un 2e tuple montre que ce même fournisseur a aussi travaillé pour nous le lendemain : alors qu'avec F_D on a pu respecter la 6NF à la lettre, mais certainement pas dans l'esprit, on enfreint les impératifs LDD, qui sont là pour nous mettre en garde. L'objet de l'article étant limité — en principe — à l'étude de la normalisation, nous n'approfondissons pas ici celle des 9 impératifs, mais on pourra se reporter avec grand profit, soit à [Date 2003] où ils sont étudiés en détail, soit à [Date 2004] qui en fournit une synthèse, ou encore au [support de cours](#) fourni par Hugh Darwen.

En tout cas, pour être en conformité avec le 2e impératif LDD, F_D doit subir une cure d'amaigrissement, être réduite à F_D' comme le montre l'image ci-dessous, où l'on voit qu'une surabondance de mauvais aloi fait place à la concision :

F_D	<u>Four_No</u>	<u>Depuis</u>	<u>Jusqu_au</u>
	C1	2005-02-01	2005-02-28
	C1	2005-03-01	2005-03-27
	C1	2005-03-28	2005-05-01
	C1	2005-06-12	2005-07-14
	C1	2005-07-15	2005-09-30
	C1	2005-10-01	2005-12-02
	C1	2005-12-03	2006-04-16
	C1	2006-04-17	2006-05-01
	C1	2006-05-02	2006-07-14
	C1	2006-07-15	2007-12-01
	C2	2005-02-01	2007-02-15
	C3	2005-07-01	2005-08-30

F_D'	<u>Four_No</u>	<u>Depuis</u>	<u>Jusqu_au</u>
	C1	2005-02-01	2005-05-01
	C1	2005-06-12	2007-12-01
	C2	2005-02-01	2007-02-15
	C3	2005-07-01	2005-08-30

Figure 6.15 - Relvars F_D et F_D', place à la concision

Si l'on remplaçait F_D par la table historisant les données relatives aux dix millions de clients d'une banque, ou mieux, celles d'un organisme à la dimension de la Sécurité sociale, on peut imaginer l'intérêt qu'il y aurait à respecter la 6NF et les impératifs. Plus généralement, cela vaut pour certaines bases de données réputées « énormes », comportant des tables de dizaines de milliards de lignes, truffées de dates.

👉 Le contenu de la relvar Affecter de la Figure 4.2 donne matière à réfléchir quant au respect du 2e impératif LDD...

Nous nous intéresserons un peu plus loin aux autres impératifs LDD.

6.3.8. Déclaration des relvars

Si on normalise la relvar F_Cessation en 6NF, c'est-à-dire si on la remplace par ses U_projections FND, FSD, FVD et par FDD (cf. Figure 6.8 et suivantes), celles-ci feront l'objet d'une déclaration dans laquelle on pourra faire figurer deux nouvelles contraintes, à savoir PACKED ON et WHEN/THEN.

Avec la contrainte PACKED ON, on demande au système que les valeurs prises par une relvar (valeurs qui sont des relations) soient toujours sous forme compressée. Par exemple :

```
VAR FSD
    BASE RELATION {Four_No CHAR,
                    Statut  INTEGER,
                    Durant  INTERVAL_DATE}
    PACKED ON Durant
    KEY {Four_No, Durant} ;
```

Ceci permet de se prémunir contre les **redondances** et d'assurer la **concision** (cf. paragraphe 6.3.7).

Avec la contrainte WHEN/THEN, on demande au système de garantir l'unicité des clés candidates dans la version décompressée d'une relvar :

```
VAR FSD
    BASE RELATION {Four_No CHAR,
                    Statut  INTEGER,
                    Durant  INTERVAL_DATE}
    PACKED ON Durant
    WHEN UNPACKED ON Durant THEN KEY {Four_No, Durant}
    KEY {Four_No, Durant} ;
```

Ceci permet de garantir la règle : « A un jour donné, un fournisseur n'a qu'un statut », donc de se prémunir contre des **contradictions** telles que celle-ci-dessous (conduisant à défaut à développer des contraintes faisant intervenir les opérateurs MEETS et OVERLAPS), auxquelles sont exposées les relvars FND, FSD et FVD (mais pas FDD) :

<u>Four_No</u>	<u>Statut</u>	<u>Durant</u>
C1	10	[2005-02-01:2005-02-28]
C1	12	[2005-02-15:2005-05-01]
...

Figure 6.16 - Contradiction, car un fournisseur ne peut pas avoir deux statuts le même jour

N.B. Toujours dans le but d'assurer la concision, la déclaration précédente peut être remplacée par la suivante :

```
VAR FSD
    BASE RELATION {Four_No CHAR,
                    Statut  INTEGER,
                    Durant  INTERVAL_DATE}
    USING Durant KEY {Four_No, Durant} ;
```

La paire {Four_No, Durant} constitue alors ce que l'on appelle une « **U_Key** ».

6.4. Données actives et données historisées

Premières tentatives de modélisation

Ce qui suit concerne évidemment la 6NF, mais il s'agit de considérations ayant fondamentalement trait aux données temporelles, et il n'est pas inutile d'en parler dans le contexte de la modélisation. Nous avons traité de la 6NF à partir de l'exemple d'une relvar F_Cessation (cf. Figure 6.14) comportant un attribut de type intervallaire (attribut Durant). Prenons aussi en compte les informations actives, « en cours ». Remontons le temps et situons-nous au 1er janvier 2005 : à cette date est née une relvar de base semblable à la relvar F (cf. Figure 6.1) et répondant au doux nom de F_Depuis. Le 3 février de la même année elle héberge ses deux premiers tuples :

F_Depuis	<u>Four_No</u>	Four_Nom	Statut	Ville
	C1	Albert & Fils	10	Lille
	C2	Bernard	20	Amiens

Figure 6.17 - Relvar F_Depuis, données actives

Comme on historise les données, associons à F_Depuis une relvar, appelons-la F_Histo, pour laquelle on reprend la structure de la relvar F_Cessation (cf. Figure 6.14) et dont la valeur initiale est la suivante (cardinal 0) :

F_Histo	<u>Four_No</u>	Four_Nom	Statut	Ville	<u>Durant</u>

Figure 6.18 - Relvar F_Histo, valeur initiale

Le 1er mars, le fournisseur C1 change de statut et l'on en historise la valeur initiale 10, mais on serait bien en peine *a priori* de déterminer la valeur prise par l'attribut Durant, car quelle est la date de début correspondante ? S'agit-il du 1er janvier (date de naissance de F_Histo), du 3 février, date à laquelle la relvar F_Depuis a changé de valeur, ou d'une quelconque autre date, antérieure au 1er mars ? L'usage veut que l'on enrichisse la structure de la relvar F_Depuis en la dotant d'un attribut correspondant à ce que les informaticiens ont coutume d'appeler « la date de début » (fixée par l'utilisateur au 1er février pour les fournisseurs C1 et C2, date à laquelle l'entreprise a passé contrat avec eux). Appelons *Depuis* cet attribut. La valeur de la relvar restructurée F_Depuis est celle-ci :

F_Depuis	<u>Four_No</u>	Four_Nom	Statut	Ville	Depuis
	C1	Albert & Fils	10	Lille	2005-02-01
	C2	Bernard	15	Amiens	2005-02-01

Figure 6.19 - Relvar F_Depuis : prise en compte du temps

Ainsi, on est à même de renseigner correctement F_Histo quand le statut du fournisseur C1 passe de la valeur 10 à la valeur 12, avec effet au 1er mars. Les relvars deviennent :

F_Depuis	<u>Four_No</u>	Four_Nom	Statut	Ville	Depuis
	C1	Albert & Fils	12	Lille	2005-03-01
	C2	Bernard	15	Amiens	2005-02-01

F_Histo	<u>Four_No</u>	Four_Nom	Statut	Ville	<u>Durant</u>
	C1	Albert & Fils	10	Lille	[2005-02-01:2005-02-28]

Figure 6.20 - Historisation au 1er mars

Mais déjà un problème se pose : on enfreint le 2e impératif LDD (cf. paragraphe 6.3.7), puisqu'on apprend que le fournisseur C1 est sous contrat le 28 février 2005 (relvar F_Histo), et qu'il en est de même le lendemain, 1er mars (relvar F_Depuis). Qui plus est, on enfreint le 5e impératif LDD dont l'énoncé est le suivant :

5e impératif LDD



Si la base de données montre que tel fournisseur a le même nom à j et j+1, alors il y a exactement un tuple exprimant ce fait (même principe concernant son statut et son lieu de résidence).

On est en infraction parce que le nom de la raison sociale (Albert & Fils) du fournisseur C1 n'a pas changé entre le 28 février (cf. relvar F_Histo) et le lendemain, 1er mars (cf. relvar F_Depuis). De la même façon, on constate qu'à ces deux dates il réside toujours au même endroit (Lille).

Pour se mettre en règle, on est conduit à n'historiser **que les seules** données ayant été modifiées (en l'occurrence le statut du fournisseur C1). A cet effet (voir ci-dessous) on décompose la relvar F_Histo par projection selon les relvars FNH, FSH, FVH (et FDH tant qu'à faire), à l'image des relvars FND, FSD, FVD (et FDD) représentées Figure 6.8 et suivantes. Ainsi, puisque lui seul a changé, on historise le statut du fournisseur C1 (dans la relvar FSH), et c'est tout.

Dans ces conditions, la relvar F_Histo passe au fil du rasoir d'Ockham et disparaît. Les valeurs des relvars obtenues par projection sont les suivantes :

FNH - Au 1er mars 2005, aucun nom de raison sociale n'a été changé :

FNH	<u>Four_No</u>	Four_Nom	<u>Durant</u>

Figure 6.21 - Historique des noms des raisons sociales (cardinal 0)

FSH - Au 1er mars 2005, le fournisseur C1 a changé de statut :

FSH	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]

Figure 6.22 - Historique des statuts

FVH - Au 1er mars 2005, aucun fournisseur n'a changé de lieu de résidence :

FVH	<u>Four_No</u>	Ville	<u>Durant</u>

Figure 6.23 - Historique des lieux de résidence (cardinal 0)

FDH - Au 1er mars 2005, il n'y a eu aucune interruption de contrat :


FDH	<u>Four_No</u>	<u>Durant</u>

Figure 6.24 - Historique des contrats (cardinal 0)

(Notons en passant que les contrats peuvent connaître des interruptions, lesquelles doivent être historisées. Il est des cas de figure dans lesquels ces interruptions sont monnaie courante, se reporter par exemple au « suivi » par les caisses de retraite des périodes de carrière des salariés dans les entreprises).

Cette fois-ci, le 2e et le 5e impératifs LDD sont respectés (le 2e met en jeu les relvars F_Depuis et FDH, tandis que le 5e met en jeu les relvars F_Depuis et FSH pour les statuts). Mais nous ne sommes pas au bout de nos peines, il y a en effet infraction par rapport au 6e impératif LDD qui suit (c'est le parcours du combattant...)


6e impératif LDD

 Si la base de données montre que tel fournisseur a tel nom au jour j, alors elle doit aussi montrer que ce fournisseur est sous contrat ce même jour (même principe concernant son statut et son lieu de résidence).

Beau défi, et comme dirait Raoul Volfoni : « C'est du brutal... » Au 1er février le fournisseur C1 avait le statut 10 (relvar FSH, cf. Figure 6.22), mais ni la relvar F_Depuis ni la relvar FDH ne nous permettent de savoir de manière formelle que ce jour-là ce fournisseur était sous contrat, alors que c'est leur rôle et pas celui de la relvar FSH qui a pour unique objet de nous renseigner de façon explicite sur les statuts passés et rien d'autre.

Concernant le nom de la raison sociale du fournisseur C1 (Albert & Fils), les relvars FNH et FDH étant vides, ne reste que la relvar F_Depuis pour nous renseigner. Mais en l'état, tout ce que l'on peut conjecturer est que le fournisseur C1 s'appellerait Albert & Fils (et serait sous contrat) depuis peut-être le 1er mars 2005 ; malheureusement cela est faux car en réalité la bonne date est celle du 1er février.

Aux grands maux, les grands remèdes, en un mot comme en cent, il n'y a pas trente-six solutions pour se tirer d'affaire :

 On adjoint à chaque attribut de la relvar F_Depuis un attribut de type date dès lors que les données correspondantes sont sujettes à historisation. De cette façon, on saura exactement depuis quand un fournisseur est sous contrat, de quand date son nom actuel, son statut, sa localisation.

Bref, quitte à ne respecter que la 5NF (ce qui en l'occurrence est suffisant), la structure de la relvar F_Depuis évolue et sa valeur au 1er mars 2005 est la suivante :

F_Depuis	Four_No	Four_No_Deb	Four_Nom	Four_Nom_Deb	Statut	Statut_Deb	Ville	Ville_Deb
C1		2005-02-01	Albert & Fils	2005-02-01	12	2005-03-01	Lille	2005-02-01
C2		2005-02-01	Bernard	2005-02-01	15	2005-02-01	Amiens	2005-02-01


Figure 6.25 - Relvar F_Depuis : valeur à ce jour, par attribut de la relvar

On sait désormais que c'est depuis le 1er février 2005 (attribut Four_No_Deb) que le fournisseur C1 est sous contrat, sans interruption jusqu'à ce jour car FDH est vide. On peut vérifier que cette fois-ci on respecte le 6e impératif LDD. En effet :

- ✓ Selon la relvar F_Depuis, à dater du 1er février 2005 (attribut Four_Nom_Deb), le nom de la raison sociale du fournisseur C1 est Albert & Fils, et depuis cette date ce fournisseur est sous contrat (attribut Four_No_Deb).
- ✓ Du 1er février 2005 au 28 février 2005, le statut du fournisseur C1 valait 10 (relvar FSH) et pendant cette période il était sous contrat (c'est ce que dit la relvar F_Depuis, attribut Four_No_Deb). Depuis le 1er mars 2005 son statut vaut 12 et à cette date, il est sous contrat (relvar F_Depuis, attribut Four_No_Deb).
- ✓ Selon la relvar F_Depuis, à dater du 1er février 2005 (attribut Ville_Deb), le fournisseur C1 réside à Lille, et à cette date il est sous contrat (attribut Four_No_Deb).

Du fait de ces structures, le travail du développeur sera incontestablement simplifié et sécurisé dans la manipulation des données temporelles. En ce sens, il n'est pas inutile d'avoir aussi à l'esprit les 1er, 3e et 4e impératifs LDD (courage !)

1er impératif LDD

 Si la base de données montre que tel fournisseur est sous contrat au jour j, alors il y a exactement un tuple exprimant ce fait.

Cette règle permet de s'assurer qu'il n'y a pas de redondance en ce sens. Seules sont concernées les relvars F_Depuis (attribut Four_No_Deb) et FDH. En l'occurrence, le 1er impératif LDD est bien respecté, puisque pour le fournisseur C1 l'attribut Four_No_Deb de la relvar F_Depuis vaut '2005-02-01' et que C1 est absent de FDH.

3e impératif LDD



Si la base de données montre que tel fournisseur est sous contrat au jour j, alors elle doit aussi montrer que ce fournisseur a un nom ce même jour (ainsi qu'un statut et un lieu de résidence).

Cet impératif (qui est le symétrique du 6e) permet de s'assurer qu'il n'y a pas d'information absente (nom de la raison sociale, statut, lieu de résidence) quand un fournisseur est sous contrat, quel que soit j.

Par exemple, le fournisseur C1 est sous contrat le 1er février 2005 (Figure 6.25 : relvar F_Depuis, attribut Four_No_Deb). On apprend une seule fois que ce même jour sa raison sociale est Albert & Fils (relvar F_Depuis), que son statut vaut 10 (relvar FSH) et qu'il réside à Paris (relvar F_Depuis). De même, au 1er mars 2005 le fournisseur C1 est toujours sous contrat, et si son statut vaut désormais 12 (relvar F_Depuis), sa raison sociale est encore Albert & Fils et il réside toujours à Paris (relvar F_Depuis).

4e impératif LDD



Si la base de données montre que tel fournisseur a un nom au jour j, alors il y a exactement un tuple exprimant ce fait (même principe concernant son statut et son lieu de résidence).

Par exemple, selon la relvar F_Depuis, le fournisseur C1 n'a bien qu'un nom au jour j (grâce à la clé {Four_No} et du fait que la relvar FNH ne contient rien concernant ce fournisseur). De même, on peut vérifier que C1 n'a qu'un statut et ne réside qu'à un endroit au jour j.

Cette règle a à voir avec la redondance.

Poursuite du processus d'historisation

On n'a effectué jusqu'ici qu'une seule modification : le 1er mars 2005, le fournisseur C1 a changé de statut. Supposons maintenant qu'il a déménagé à Paris le 28 mars. L'état de la base de données devient alors le suivant :

F_Depuis	Four_No	Four_No_Deb	Four_Nom	Four_Nom_Deb	Statut	Statut_Deb	Ville	Ville_Deb
	C1	2005-02-01	Albert & Fils	2005-02-01	12	2005-03-01	Paris	2005-03-28
	C2	2005-02-01	Bernard	2005-02-01	15	2005-02-01	Amiens	2005-02-01

FNH	Four_No	Four_Nom	Durant

FSH	Four_No	Statut	Durant
	C1	10	[2005-02-01:2005-02-28]

FVH	Four_No	Ville	Durant
	C1	Lille	[2005-02-01:2005-03-27]

FDH	Four_No	Durant

Figure 6.26 - État de la base données au 28 mars 2005

Le fait que le fournisseur C1 a déménagé s'est traduit par l'historisation dans la relvar FVH de son passé lillois, tandis que la relvar F_Depuis a été mise à jour pour tenir compte de sa nouvelle localisation (*ach, Paris!*)

On peut vérifier que les impératifs LDD sont respectés.

Continuons à suivre le parcours mouvementé du fournisseur C1. A dater du 2 mai 2005 il n'est plus sous contrat, mais on a décidé de conserver l'historique de ses données. L'état de la base de données est alors le suivant :

F_Depuis	<u>Four_No</u>	Four_No_Deb	Four_Nom	Four_Nom_Deb	Statut	Statut_Deb	Ville	Ville_Deb
	C2	2005-02-01	Bernard	2005-02-01	15	2005-02-01	Amiens	2005-02-01

FNH	<u>Four_No</u>	Four_Nom	<u>Durant</u>
	C1	Albert & Fils	[2005-02-01:2005-05-01]

FSH	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]
	C1	12	[2005-03-01:2005-05-01]

FVH	<u>Four_No</u>	Ville	<u>Durant</u>
	C1	Lille	[2005-02-01:2005-03-27]
	C1	Paris	[2005-03-28:2005-05-01]

FDH	<u>Four_No</u>	<u>Durant</u>
	C1	[2005-02-01:2005-05-01]

Figure 6.27 - État de la base données au 2 mai 2005

Le 12 juin 2005, le fournisseur C1 est à nouveau sous contrat, avec la même raison sociale, il habite toujours Paris, mais son statut vaut 14. L'état de la base de données devient alors le suivant :

F_Depuis	<u>Four_No</u>	Four_No_Deb	Four_Nom	Four_Nom_Deb	Statut	Statut_Deb	Ville	Ville_Deb
	C1 C2	2005-06-12 2005-02-01	Albert & Fils Bernard	2005-06-12 2005-02-01	14 15	2005-06-12 2005-02-01	Paris Amiens	2005-06-12 2005-02-01

FNH	<u>Four_No</u>	Four_Nom	<u>Durant</u>
	C1	Albert & Fils	[2005-02-01:2005-05-01]

FSH	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]
	C1	12	[2005-03-01:2005-05-01]

FVH	<u>Four_No</u>	Ville	<u>Durant</u>
	C1	Lille	[2005-02-01:2005-03-27]
	C1	Paris	[2005-03-28:2005-05-01]

FDH	<u>Four_No</u>	<u>Durant</u>
	C1	[2005-02-01:2005-05-01]

Figure 6.28 - État de la base données au 12 juin 2005

On peut vérifier que les impératifs LDD sont respectés.

Poursuivons. Nouvel événement : le 15 juillet 2005, le fournisseur C1 déménage à Lyon (Ciao Paris...) L'état de la base de données devient le suivant :

F_Depuis	<u>Four_No</u>	Four_No_Deb	Four_Nom	Four_Nom_Deb	Statut	Statut_Deb	Ville	Ville_Deb
	C1	2005-06-12	Albert & Fils	2005-06-12	14	2005-06-12	Lyon	2005-07-15
	C2	2005-02-01	Bernard	2005-02-01	15	2005-02-01	Amiens	2005-02-01

FNH	<u>Four_No</u>	Four_Nom	<u>Durant</u>
	C1	Albert & Fils	[2005-02-01:2005-05-01]

FSH	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]
	C1	12	[2005-03-01:2005-05-01]

FVH	<u>Four_No</u>	Ville	<u>Durant</u>
	C1	Lille	[2005-02-01:2005-03-27]
	C1	Paris	[2005-03-28:2005-05-01]
	C1	Paris	[2005-06-12:2005-07-14]

FDH	<u>Four_No</u>	<u>Durant</u>
	C1	[2005-02-01:2005-05-01]

Figure 6.29 - État de la base données au 15 juillet 2005

Supposons qu'au fil du temps, le fournisseur C1 fasse l'objet des changements suivants :

- Le 1er octobre 2005, changement du nom de sa raison sociale, qui devient 'Albert & Cie',
- Le 3 décembre 2005, déménagement à Nantes,
- Le 17 avril 2006, déménagement à Caen,
- Le 2 mai 2006, déménagement à Lille et changement de son statut qui passe à 25,
- Le 15 juillet 2006, changement du nom de sa raison sociale, qui devient 'Albert'.

A partir de là, calme plat.

Au 15 juillet 2006, l'état de la base de données est le suivant, exhaustif, concis et non redondant, dans le respect des impératifs LDD :

F_Depuis	<u>Four_No</u>	Four_No_Deb	Four_Nom	Four_Nom_Deb	Statut	Statut_Deb	Ville	Ville_Deb
	C1	2005-06-12	Albert	2006-07-15	25	2006-05-02	Lille	2006-05-02
	C2	2005-02-01	Bernard	2005-02-01	15	2005-02-01	Amiens	2005-02-01

FNH	<u>Four_No</u>	Four_Nom	<u>Durant</u>
	C1	Albert & Fils	[2005-02-01:2005-05-01]
	C1	Albert & Fils	[2005-06-12:2005-09-30]
	C1	Albert & Cie	[2005-10-01:2006-07-14]

FSH	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]
	C1	12	[2005-03-01:2005-05-01]
	C1	14	[2005-06-12:2006-05-01]

FVH	<u>Four_No</u>	Ville	<u>Durant</u>
	C1	Lille	[2005-02-01:2005-03-27]
	C1	Paris	[2005-03-28:2005-05-01]
	C1	Paris	[2005-06-12:2005-07-14]
	C1	Lyon	[2005-07-15:2005-12-02]
	C1	Nantes	[2005-12-03:2006-04-16]
	C1	Caen	[2006-04-17:2006-05-01]

FDH	<u>Four_No</u>	<u>Durant</u>
	C1	[2005-02-01:2005-05-01]

Figure 6.30 - État de la base données au 15 juillet 2006

Voilà une affaire qui roule. L'étude de la 6NF nous a fait voyager dans le temps, mais de façon raisonnée.

6.5. Points particuliers

6.5.1. A propos de l'intégrité référentielle et des contraintes d'intégrité en général

Si on examine la Figure 6.27, on se rend compte que la relvar F_Depuis ne peut pas être référencée par les relvars FNH, FSH, FVH et FDH, et l'on peut compter sur les gendarmes de l'intégrité référentielle (IR) pour nous rappeler sèchement à l'ordre et exiger que cette lacune soit comblée, même si l'IR n'est pas la panacée. Toutefois, si pour les satisfaire, on conservait le fournisseur C1 dans la relation F_Depuis, à quoi ressemblerait le tuple correspondant ? Nombre de ceux qui pratiquent SQL marqueraient à NULL tous les attributs autres que Four_No, ou concocteraient des valeurs spéciales, folkloriques, du genre '9999-12-31' pour les dates, ou encore marqueraient ce tuple comme « logiquement supprimé », mais nous ne sommes pas là pour bricoler. En fait, tout ce qu'on peut exiger de notre part est la parfaite cohérence des

données constitutives du fournisseur C1. En l'espèce, cela concerne les relvars FNH, FSH, FVH et FDH, et cette cohérence est effective parce que nous respectons les impératifs LDD 3 et 6 : nous sommes irréprochables.

Ainsi, quand un fournisseur fait l'objet d'une interruption de contrat, à l'instar du fournisseur C1 il peut être supprimé sans risque de la relvar F_Depuis. On trouvera dans [Date 2003] au chapitre 12 « Integrity Constraints II: General Constraints » des pages fort intéressantes, détaillant la programmation de l'ensemble des contraintes permettant de garantir les impératifs LDD, donc l'intégrité des données dans le contexte de leur historisation. Cela évitera aux développeurs de perdre inutilement du temps à réinventer l'eau chaude et à concevoir la programmation de ces contraintes (programmation qui du reste peut être sous-traitée au SGBD, sous le capot) : nous les renvoyons aux développements proposés dans le chapitre 12 mentionné.

6.5.2. Choix du mode d'historisation

Certains chefs de projets préconisent que, lors de l'interruption de son contrat, les données du fournisseur C1 soient transférées (disons archivées, par exemple pour des raisons légales, à des fins de preuves) dans une relvar ad-hoc, du genre F_Cessation (cf. paragraphe 6.3) et donc que l'on n'en conserve aucune trace, tant dans la relvar F_Depuis que dans les relvars FNH, FSH, FVH et FDH. Cet archivage peut être effectué dans une base de données dédiée, dans la mesure où l'on ne s'intéresse plus à ce genre de données dans le cadre normal de la Production, mais seulement de façon exceptionnelle. Si l'ex-fournisseur C1 faisait à nouveau l'objet d'un contrat, il serait à considérer comme un tout nouveau fournisseur, avec un numéro de fournisseur Four_No dont la valeur différerait de 'C1'. Cette approche est parfaitement légitime et offre des avantages, ne serait-ce qu'au plan de la simplicité, mais, rappelons-le, il est des situations dans lesquelles les interruptions temporelles sont normales, sans pour autant que l'on supprime toutes les données et, comme on l'a déjà signalé, c'est par exemple ainsi que l'on peut procéder dans les caisses de retraite, dans le cadre du suivi des périodes de carrière des salariés des entreprises, des artisans, professions libérales, etc., puisqu'il peut y avoir des interruptions dans les cotisations (changement de caisse, chômage, etc.), suivies de reprises de celles-ci. En tout état de cause, nous considérons ici qu'une interruption de contrat peut être considérée comme temporaire (d'où l'état de la base de données au 12 juin 2005 à l'occasion du retour du fournisseur C1, cf. Figure 6.27 & Figure 6.28) : on conserve le passé dans les relvars FNH, FSH, FVH et FDH et l'on en débarrasse la relvar F_Depuis (d'où une perte de surpoids bénéfique pour la performance des traitements de masse).

6.5.3. Relvars « associatives »

Si l'on se reporte à la Figure 2.6 (paragraphe 2.6), on est amené à se poser la question suivante : On a traité jusqu'ici de l'historisation des données propres aux fournisseurs (relvar F), mais qu'en est-il des données qui en dépendent, par exemple celles qui sont contenues dans la relvar *associative* FP (pièces livrées) ? Nous ne chercherons pas dans cet article à poursuivre le travail esquissé jusqu'ici, le lecteur intéressé approfondira lui-même le sujet en étudiant plus avant [Date 2003], en se frottant aux impératifs LDD 7, 8 et 9 établis justement pour garantir l'historisation correcte des données concernant les relvars telles que FP (historisation des relations (associations) au sens Merise).

6.5.4. Trois relvars ou une relvar unique ?

Peut-on remplacer les trois relvars FNH, FSH et FVH par une relvar unique F_Histo qui en serait leur U_jointure ?

F_Histo	<u>Four_No</u>	Four_Nom	Statut	Ville	<u>Durant</u>
---------	----------------	----------	--------	-------	---------------

Figure 6.31 - Relvar F_Histo hypothétique

Une telle relvar serait à même de séduire les développeurs qui le plus souvent voient d'un bon œil l'accès à une seule relvar et sont peu favorables à la prétendue « complexité des jointures » (la rengaine). De même, la Production informatique serait partante, parce qu'elle préfère n'avoir qu'un minimum de fichiers de sauvegarde et autres à gérer,

etc. Mais, la mauvaise nouvelle est que la relvar F_Histo n'est malheureusement pas égale à la U_jointure de FNH, FSH et FVH (cf. Figure 6.30).

En effet, sans même produire le résultat final, considérons déjà la relvar FNSH, U_jointure de FNH et FSH :

FNSH	<u>Four_No</u>	Four_Nom	Statut	<u>Durant</u>
	C1	Albert & Fils	10	[2005-02-01:2005-02-28]
	C1	Albert & Fils	12	[2005-03-01:2005-05-01]
	C1	Albert & Fils	14	[2005-06-12:2005-09-30]
	C1	Albert & Cie	14	[2005-10-01:2006-05-01]

Figure 6.32 - U_jointure de FNH et FSH

Par U_projection, on obtient les relvars FNH2 et FSH2, et l'on voit que la valeur de FNH2 n'est pas égale à celle de FNH :

FNH2	<u>Four_No</u>	Four_Nom	<u>Durant</u>
	C1	Albert & Fils	[2005-02-01:2005-05-01]
	C1	Albert & Fils	[2005-06-12:2005-09-30]
	C1	Albert & Cie	[2005-10-01: 2006-05-01]

FSH2	<u>Four_No</u>	Statut	<u>Durant</u>
	C1	10	[2005-02-01:2005-02-28]
	C1	12	[2005-03-01:2005-05-01]
	C1	14	[2005-06-12:2006-05-01]

Figure 6.33 - U_projection de FNSH

De fait, au vu de FNH2, on a perdu l'information selon laquelle le nom de la raison sociale du fournisseur C1 était Albert & Cie pendant la période allant du 2006-05-02 au 2006-07-14. Ceci s'explique par le fait que ce fournisseur a obtenu le statut 25 le 2006-05-02, statut toujours en vigueur (cf. la relvar F_Depuis) : cette valeur du statut et donc la période en cause ([2006-05-02:2006-07-14]) ne peuvent pas figurer dans la relation FSH, sinon on enfreindrait le 4e impératif LDD.

6.6. Pour conclure avec la normalisation en 6NF

L'étude de la 6NF conduit inévitablement à s'intéresser de très près aux données temporelles, aux redondances qu'elles occasionnent, à leur concision (pas d'atomisation inutile, donc de tuples superflus) et à leur cohérence, ce qui nous fait très largement déborder du strict cadre de cette ultime forme normale qu'est la 6NF (selon la normalisation par projection/jointure), mais qui autrement ne présenterait guère d'intérêt qu'au plan académique. Il reste énormément à dire, ne serait-ce qu'en ce qui concerne l'expression des contraintes d'intégrité et le respect des impératifs LDD, ou encore la rédaction des requêtes de manipulation des données : si l'on veut en savoir plus, le mieux est de se lancer dans une étude approfondie de l'ouvrage de référence [Date 2003]. On y découvrira que la grande foule des contraintes et contrôles concernant spécifiquement l'historisation des données sont automatisables et peuvent donc être rendus transparents, le système pouvant les prendre à sa charge. Je traduis et résume ce qui est écrit page 242 de l'ouvrage :

« Nous conjecturons que le système est à même d'inférer ces contraintes, évitant donc à l'utilisateur d'avoir à les définir de manière explicite. »

Le moyen d'y parvenir est d'élever le niveau d'abstraction (*to raise the level of abstraction*, page 239), ce qui a toujours été un souci constant chez Codd, Date & Darwen et n'est rendu possible que grâce à un soin extrême apporté à la structure du langage, ce qui est le cas de Tutorial D. Toujours dans [Date 2003], on découvrira l'ensemble des impératifs

LDD, le principe de généralisation des concepts (*U_keys*, *foreign U_keys* par exemple) ; on y trouvera de nombreux exemples de requêtes de consultation et de mise à jour des bases de données, l'aide apportée par les relvars virtuelles (vues). Outre la *U_jointure* et la *U_projection* dont nous avons donné un aperçu, on y trouvera encore la généralisation des opérateurs relationnels, et bien d'autres sujets en relation avec les données temporelles et plus généralement intervallaires. La théorie relationnelle n'a pas fini d'évoluer, et pour s'en convaincre il suffit d'explorer cette fois-ci [Date 2010], loin du train-train quotidien.

Quoi qu'il en soit, pour avoir crapahuté dans les données temporelles depuis le milieu des années soixante, tant dans les domaines de la conception, de la programmation, de l'administration des bases de données, de leur audit et de leur sauvetage, dans tous types d'entreprises, je me dis que si l'on avait disposé dès le départ des travaux de Date, Darwen et Lorentzos, il y aurait eu infiniment moins d'âneries de proférées, de temps inutilement perdu lors du développement des applications et, il va sans dire, d'erreurs qui auraient pu être évitées, par wagons entiers.

Vive le Modèle Relationnel de Données !

Annexes

A. Tuples, relations, relvars (définitions)

A.1. Tuple (n-uplet), attribut

Étant donné une collection de types T_i ($i = 1, 2, \dots, n$), non nécessairement tous distincts, une **valeur de tuple** (tuple pour abréger) sur ces types — disons t — est un ensemble de triplets ordonnés de la forme $\langle A_i, T_i, v_i \rangle$, dans lesquels A_i est un **nom d'attribut**, T_i est un **nom de type** et v_i est une **valeur** du type T_i , et :

- La valeur n est le **degré** de t .
- Le triplet ordonné $\langle A_i, T_i, v_i \rangle$ est un **composant** de t .
- Le couple ordonné $\langle A_i, T_i \rangle$ est un **attribut** de t , identifié de façon unique par le nom d'attribut A_i (les noms d'attributs A_i et A_j sont les mêmes seulement si $i = j$). La valeur v_i est la **valeur d'attribut** de l'attribut A_i de t . Le type T_i est le **type d'attribut** correspondant.
- L'ensemble $\{H\} = \{A_1 T_1, A_2 T_2, \dots, A_n T_n\}$ des attributs constitue l'**en-tête** de t .
- Le **type de tuple** de t est déterminé par l'en-tête de t , et l'en-tête et le type de tuple possèdent les mêmes attributs (et donc les mêmes noms d'attributs et mêmes types) et le même degré que ceux de t . Le **nom du type de tuple** — défini au moyen du **générateur de type** TUPLE — est précisément :

TUPLE $\{A_1 T_1, A_2 T_2, \dots, A_n T_n\}$

A.2. Relation, relvar

Une **valeur de relation** (relation pour abréger) — appelons r cette valeur — est constituée d'un en-tête et d'un corps tels que :

- L'**en-tête** $\{H\}$ de r est identique à celui d'un tuple, tel que défini ci-dessus. La relation r a les mêmes attributs (et donc les mêmes noms d'attributs et les mêmes types de référence) et le même degré que cet en-tête.
- Le **corps** de r est un ensemble de tuples ayant tous cet en-tête ; le cardinal de cet ensemble est le **cardinal** de r .
- Le **type de relation** de r est déterminé par l'en-tête de r et possède les mêmes attributs (et donc les mêmes noms d'attributs et mêmes types) et le même degré que cet en-tête. Le **nom du type de relation** — défini au moyen du **générateur de type** RELATION — est précisément :

RELATION $\{A_1 T_1, A_2 T_2, \dots, A_n T_n\}$

La **relvar** (abréviation de variable relationnelle) est une variable prenant pour valeurs des relations dont le type est mentionné lors de la définition de la relvar. Exemple :

```
VAR Membre BASE RELATION
{
    MbrId            INTEGER
    , Pseudonyme     CHAR
    , DateInscription DATE
    , Localisation    CHAR
    , AdrCourriel     CHAR
}
KEY {MbrId}
KEY {Pseudonyme}
KEY {AdrCourriel} ;
```

Figure A-1 - Relvar Membre

En SQL, la déclaration de la structure de la table Membre, homologue de la relvar Membre, pourrait être la suivante :

```
CREATE TABLE Membre
(
    MbrId          INTEGER          NOT NULL
  , Pseudonyme    VARCHAR(16)      NOT NULL
  , DateInscription DATE           NOT NULL
  , Localisation  VARCHAR(48)      NOT NULL
  , AdrCourriel   VARCHAR(48)      NOT NULL
  , CONSTRAINT MbrPk PRIMARY KEY (MbrId)
  , CONSTRAINT CK2 UNIQUE (Pseudonyme)
  , CONSTRAINT CK3 UNIQUE (AdrCourriel)) ;
```

Figure A-2 - Table Membre

Signalons que la relvar dont nous parlons est plus précisément une relvar de base (*base* (ou *real*) *relvar*), par contraste avec la relvar dérivée (*derived* (ou *virtual*) *relvar*), autrement dit la vue.



En tout état de cause, le Modèle Relationnel ne comporte qu'un seul type de variable, à savoir la relvar.

A.3. Note à propos de Tutorial D

Le Modèle Relationnel de Données fut défini avec une grande rigueur (ce qui ne veut pas dire rigidité !) par Codd à partir de 1969. Cette rigueur permet au Modèle d'évoluer (évolution, oui, révolution, non !) grâce notamment au soin pris par Codd pour éviter la redondance, donc la complication. Les années passant, Date et Darwen assistèrent à des dérives et à des critiques non fondées du Modèle, notamment concernant de prétendues lacunes, ce qui les incita à remettre les pendules à l'heure. Ceci fut fait lors de la publication en 1998 de leur ouvrage « Foundation for Object/Relational Databases, The Third Manifesto ». A cette occasion, D & D procédèrent à une remise à plat complète du Modèle, un peu à la manière de Russell et Whitehead avec les *Principia Mathematica*. Ainsi, les concepts de tuple, de relation et relvar, sans oublier celui de type ont été formulés avec le plus grand soin dans [Date 2006], ce qui les a conduits à définir un langage approprié, **Tutorial D**.

D & D ont conçu Tutorial D en respectant les « *Principles of good language design* » qui leurs sont chers :

« **Tutorial D** est un langage de programmation complet du point de vue du calcul, intégrant toutes les fonctionnalités des bases de données. Nous n'avons pas voulu qu'il soit perçu comme doté de la "puissance industrielle" ; Il s'agit plutôt d'un langage "jouet" dont l'objet principal est de servir de support pour l'enseignement. En conséquence, ont été volontairement omises de nombreuses fonctionnalités qu'exigerait un langage véritablement industrialisé. (L'extension du langage pour la prise en compte de ces fonctionnalités serait un projet qui en vaudrait la peine, le transformant ainsi en ce qu'on pourrait appeler **Industrial D.**) ... ».

(Cf. [Date 2006], page 93).

Par « *langage de programmation complet du point de vue du calcul* », on doit comprendre que des applications entières peuvent être ainsi développées, il ne s'agit pas seulement d'un « sous-langage » de données hébergé par quelque langage hôte propre à fournir les possibilités de calcul nécessaires.

Tutorial D est un langage « jouet » dans la mesure où rien n'est pris en compte en ce qui concerne par exemple les sessions et les connexions, les communications avec le monde extérieur (gestion des entrées/sorties, etc.), ou la gestion des exceptions et des codes-retour.

Un langage vraiment relationnel, disons de la famille **D**, peut très bien intégrer des fonctionnalités indépendantes du Modèle Relationnel, dans la mesure où elles n'en pervertissent pas l'esprit. Par exemple, **D** pourrait à l'instar de SQL proposer un générateur de type ARRAY ou MULTISSET (cf. [Date 2006], chapitre 10 / « *RM Very strong suggestions* », page 234), mais en aucune façon un concept en contradiction avec l'esprit du Modèle Relationnel tel que celui de pointeur (exemple : type REF de SQL). En effet, toute information, quelle qu'elle soit, doit être représentée dans la base de données **uniquement sous forme de valeurs** prises par les attributs, au sein de tuples dans les relations (**Information Principle** de Codd).

Pour les SGBD basés sur Tutorial D, voyez <http://www.thethirdmanifesto.com/>

Si les liens ci-dessous sont toujours actifs au moment où vous lisez ces lignes, vous pourrez en tirer profit. Chris Date raconte les débuts du Modèle Relationnel de Données :

Thirty Years of Relational: The First Three Normal Forms (By C.J. Date)

http://web.archive.org/web/20050307045933/www.intelligententerprise.com/db_area/archives/1999/993003/online2.jhtml

Thirty Years of Relational: The First Three Normal Forms, Part 2 (By C.J. Date)

http://web.archive.org/web/20050307044845/www.intelligententerprise.com/db_area/archives/1999/992004/online2.jhtml

Et aussi, concernant Date :

Normalization Is No Panacea :

<http://web.archive.org/web/20030218095332/http://www.dbpd.com/vault/9804date.htm>

Etc. :

http://web.archive.org/web/20050426234346/www.intelligententerprise.com/authors/search_Date.jhtml

B. Notation des opérateurs relationnels (Tutorial D)

L'objet n'est pas de décrire ici l'ensemble des opérateurs de l'algèbre relationnelle, mais de revoir certains d'entre eux dans le cadre du Modèle Relationnel, car leur notation (Tutorial D) diffère quelque peu de celle dont on dispose en SQL.

Voir le mapping des opérateurs Tutorial D / SQL : www.dcs.warwick.ac.uk/~hugh/CS252/CS252-TD-to-SQL.pdf

Restriction

La restriction permet, à partir d'une relation R, de produire une relation R' ayant le même en-tête que celui de R et dont le corps en est un sous-ensemble ($\text{cardinal de } R' \leq \text{cardinal de } R$). L'opération met en jeu une condition impliquant un ou plusieurs attributs de R et devant être vérifiée par chacun de ses tuples.

La restriction est ainsi formulée :

R WHERE condition

Par exemple, en reprenant la base de données de la Figure 2.6 :

F WHERE Ville = "Paris" ;

est une restriction de F sur l'attribut Ville, auquel on applique la condition d'égalité « Ville = "Paris" » :

Traduction en SQL :

```
SELECT *
FROM R
WHERE Ville = "Paris" ;
```

Four_No	Four_Nom	Statut	Ville
S2	Jean	10	Paris
S3	Bernard	30	Paris

Figure B.1 - Résultat de la restriction

Projection

La projection permet, à partir d'une relation R, de produire une relation R' dont l'en-tête est un sous-ensemble de celui de R ($\text{degré de } R' \leq \text{degré de } R$) et dont le cardinal est celui de R (ou inférieur si des redondances sont éliminées).

La projection est ainsi formulée :

R { X, Y, ..., Z }

expression dans laquelle X, Y, ..., Z sont des attributs de R.

Par exemple

F { Four_No, Statut } ;

est une projection de F sur les attributs Four_No et Statut :

Four_No	Statut
S1	20
S2	10
S3	30
S4	20
S5	30

Figure B.2 - Résultat de la projection

Traduction en SQL :

```
SELECT DISTINCT Four_No, Statut
FROM F ;
```


Rename

L'opérateur Rename permet, à partir d'une relation R, de produire une nouvelle relation égale en valeur (même corps), mais avec tout ou partie des noms des attributs ayant été renommés.

Par exemple, pour produire (au moins conceptuellement) une relation identique à la relation P de la Figure 2.6, tout en renommant les attributs Piece_Nom et Poids respectivement en Libellé_Pièce et Poids_Pièce :

P RENAME (Piece_Nom AS Libellé_Pièce, Poids AS Poids_Pièce)

<u>Piece No</u>	Libellé_Pièce	Couleur	Poids_Pièce	Ville
P1	Ecrou	Rouge	12,0	Lille
P2	Boulon	Vert	17,0	Paris
P3	Vis	Bleu	17,0	Orsay
P4	Vis	Rouge	14,0	Lille
P5	Clou	Bleu	12,0	Paris
P6	Rouage	Rouge	19,0	Lille

Figure B.3 - Résultat du RENAME

Jointure (naturelle)

Soit RA et RB deux relations dont les attributs sont respectivement les suivants

X1, X2, ..., Xm, Y1, Y2, ..., Yn

et

Y1, Y2, ..., Yn, Z1, Z2, ..., Zp

Y1, Y2, ..., Yn représentant les seuls attributs ayant le même nom dans les deux relations. On observera que :

- Après RENAME si nécessaire, aucun des attributs X1, X2, ..., Xm n'a le même nom qu'un des attributs Z1, Z2, ..., Zp.
- Par définition, chaque attribut Yk (k = 1, 2, ..., n) est du **même type** dans les deux relations.

Résumons maintenant {X1, X2, ..., Xm}, {Y1, Y2, ..., Yn} et {Z1, Z2, ..., Zp} respectivement par X, Y, Z. Alors, la jointure naturelle de RA et RB

RA JOIN RB

est une relation d'en-tête {X, Y, Z} et dont le corps est constitué de tous les tuples {X x, Y y, Z z} tels que chacun de ces tuples apparaît dans RA avec X ayant la valeur x et Y ayant la valeur y d'une part et apparaît dans RB avec Y ayant la valeur y et Z ayant la valeur z d'autre part.

Par exemple, dans le cas des relations F1 et P1 ci-dessous : X correspond à la paire {Four_No, Four_Nom}, Y au singleton {Ville} et Z à la paire {Piece_No, Piece_Nom} :

F1

<u>Four_No</u>	Four_Nom	Ville
S1	Salsa	Lille
S2	Jean	Paris
S4	Catherine	Lille
S5	Alain	Arles

P1

<u>Piece_No</u>	Piece_Nom	Ville
P1	Ecrou	Lille
P2	Boulon	Paris
P3	Vis	Orsay

Figure B.4 - Relations F1 et P1

La jointure naturelle

F1 JOIN P1

produit la relation :

<u>Four_No</u>	Four_Nom	Ville	<u>Piece_No</u>	Piece_Nom
S1	Salsa	Lille	P1	Ecrou
S4	Catherine	Lille	P1	Ecrou
S2	Jean	Paris	P2	Boulon

Figure B.5 - Jointure naturelle des relations F1 et P1

Équivalent SQL :

```
SELECT  Four_No, Four_Nom, F1.Ville, Piece_No, Piece_Nom
FROM    F1 NATURAL JOIN P1 ;
```

Union, Intersection, Différence

Il n'y a rien de particulier à dire concernant les opérateurs UNION, INTERSECT, MINUS, si ce n'est que les opérandes doivent être du même type (donc avoir **exactement le même en-tête** : mêmes noms des attributs et mêmes types de référence pour ces derniers).

Soit donc F1 et F2 deux relations de même type :

F1	<table> <tr> <th><u>Four_No</u></th><th>Four_Nom</th><th>Statut</th><th>Ville</th></tr> <tr> <td>S1</td><td>Salsa</td><td>20</td><td>Lille</td></tr> <tr> <td>S4</td><td>Catherine</td><td>20</td><td>Lille</td></tr> </table>	<u>Four_No</u>	Four_Nom	Statut	Ville	S1	Salsa	20	Lille	S4	Catherine	20	Lille	F2	<table> <tr> <th><u>Four_No</u></th><th>Four_Nom</th><th>Statut</th><th>Ville</th></tr> <tr> <td>S1</td><td>Salsa</td><td>20</td><td>Lille</td></tr> <tr> <td>S2</td><td>Jean</td><td>10</td><td>Paris</td></tr> </table>	<u>Four_No</u>	Four_Nom	Statut	Ville	S1	Salsa	20	Lille	S2	Jean	10	Paris
<u>Four_No</u>	Four_Nom	Statut	Ville																								
S1	Salsa	20	Lille																								
S4	Catherine	20	Lille																								
<u>Four_No</u>	Four_Nom	Statut	Ville																								
S1	Salsa	20	Lille																								
S2	Jean	10	Paris																								

Figure B.6 - Union des relations F1 et P1 : les opérandes sont du même type

L'union de F1 et F2 s'écrit

F1 UNION F2

et produit un résultat de même type :

<u>Four_No</u>	Four_Nom	Statut	Ville
S1	Salsa	20	Lille
S4	Catherine	20	Lille
S2	Jean	10	Paris

Figure B.7 - Résultat de l'union des relations F1 et P1

Équivalent SQL :

```
SELECT  Four_No, Four_Nom, Statut, Ville
FROM    F1
UNION
SELECT  Four_No, Four_Nom, Statut, Ville
FROM    F2 ;
```

EXTEND

L'opérateur EXTEND permet, à partir d'une relation R de produire une relation dont l'en-tête est celui de R, augmenté d'un attribut (liste d'attributs dans le cas général, se reporter à [Date 2006] et [Date 2010]). La valeur prise par chaque tuple de la nouvelle relation pour cet attribut est le résultat d'une opération portant sur les valeurs prises par les autres attributs pour ce même tuple.

Plus précisément, la valeur de l'extension

EXTEND R ADD *exp* AS Z

est une relation :

- Dont l'en-tête est celui de R, étendu de l'attribut Z.
- Dont le corps est constitué de tous les tuples *t*, tels que *t* est un tuple de R étendu avec une valeur de Z résultant de l'évaluation de *exp* pour le tuple *t*.

L'en-tête de la relation R ne doit pas comporter d'attribut nommé Z et *exp* ne doit pas mentionner Z. Le type de Z est celui de *exp*.

Exemple

Considérons la relvar des pièces :

P	<u>Piece_No</u>	Piece_Nom	Couleur	Poids	Ville
	P1	Ecrou	Rouge	12,0	Lille
	P2	Boulon	Vert	17,0	Paris
	P3	Vis	Bleu	17,0	Orsay
	P4	Vis	Rouge	14,0	Lille
	P5	Clou	Bleu	12,0	Paris
	P6	Rouage	Rouge	19,0	Lille

Figure B.8 - Relvar des pièces

Supposons qu'à partir de P, on veuille fournir une relation comportant le poids en grammes des pièces (dans P, ce poids est exprimé en livres anglaises et une livre = 454 grammes). On utilise alors ainsi l'opérateur EXTEND :

EXTEND P ADD (Poids * 454) AS Poids_Grm

<u>Piece_No</u>	Piece_Nom	Couleur	Poids	Ville	Poids_Grm
P1	Ecrou	Rouge	12,0	Lille	5448,0
P2	Boulon	Vert	17,0	Paris	7718,0
P3	Vis	Bleu	17,0	Orsay	7718,0
P4	Vis	Rouge	14,0	Lille	6356,0
P5	Clou	Bleu	12,0	Paris	5448,0
P6	Rouage	Rouge	19,0	Lille	8626,0

Figure B.9 - Extension de P (poids en grammes)

C. Notes concernant la première forme normale et la logique du deuxième ordre

Comme on l'a vu lors de l'étude la première forme normale (cf. paragraphe 2.1), Ted Codd avait écrit en 1969 :

Le calcul des prédicats du deuxième ordre est nécessaire (plutôt que celui du premier ordre) parce que les domaines sur lesquels les relations sont définies peuvent à leur tour contenir des éléments qui sont des relations.

Tandis qu'il se ravisait un an plus tard (cf. paragraphe 2.2) :

L'utilisation d'un modèle relationnel de données ... permet de développer un sous-langage universel basé sur le calcul des prédicats. Si la collection des relations est en forme normale, alors un calcul des prédicats du premier ordre est suffisant.

Et il montrait comment normaliser une relation contenant d'autres relations.

Indépendamment de ces considérations coddienues, on peut se demander ce qu'est la logique du 2e ordre. N'étant pas logicien, je me garderai de développer le sujet, mais, à la demande de tel et tel relecteurs, me contenterai simplement de quelques indications, en me basant essentiellement sur l'ouvrage *Méthodes de logique* [Quine 1972] de W.V.O. Quine, qui est considéré comme l'un des plus grands philosophes américains du XX^e siècle, sinon le plus grand.

C.1. Univers du discours

Quelques concepts importants sont d'abord à rappeler, tel que celui d'*univers du discours*.

On doit ce concept à Augustus De Morgan, qui le présente dans un article de 1846, [On the Structure of the Syllogism](#), puis dans son traité de 1847, [Formal Logic](#) (page 55). A l'époque, il utilise l'expression *Univers d'une proposition*, qu'il abrège en *Univers*.

En logique, l'univers du discours représente une collection d'objets, d'éléments dont nous sommes convenus de partager la compréhension, sans chercher nécessairement à les énumérer tous. L'intérêt de cet univers est qu'il permet, de façon décisive, de dépasser les limites de la stricte logique aristotélicienne. Citons l'article de 1846, page 380 :

Writers on logic [...] give an indefinite negative character to the contrary, as Aristotle when he said that not-man was not the name of anything. Let the universe in question be "man:" then Briton and alien are simple contraries; alien has no meaning of definition except not-Briton...

La contribution d'A. De Morgan est capitale, ne serait-ce que parce qu'implicitement, elle sous-entend le concept de *Complémentaire*, indispensable à la théorie des ensembles (étant donné un sous-ensemble *A* d'un ensemble *E*, on appelle complémentaire de *A* dans *E* l'ensemble des éléments de *E* qui n'appartiennent pas à *A*).

En tout état de cause, nous pouvons parler de l'univers des membres de DVP. De même, un MCD, un diagramme de classes, un MLD sont des exemples d'univers. Par le biais de l'univers du discours nous convenons de ce dont nous parlons, tout en nous imposant *de facto* des limites : si l'on parle de géométrie euclidienne, on se restreint par rapport à un univers plus vaste, comprenant les géométries non euclidiennes, pour lesquelles les axiomes définis par Euclide ne sont plus tous applicables. Et puis, nos capacités à raisonner ne sont pas infinies.

Quoi qu'il en soit, dans le cadre de la logique et pour reprendre la définition de Quine, l'univers du discours est le parcours (*range*) d'objets *x* convenant à l'argument logique que nous nous proposons de mener à bonne fin.

Par exemple, dans le contexte des bases de données, la table Membre des membres de Developpez.com (cf. paragraphe 2.2) peut être considérée comme un univers de *n*-uplets que l'on peut parcourir. Dans la notation du langage QUEL :

```
RANGE OF x IS Membre
RETRIEVE (x ALL)
```

(RETRIEVE (x ALL) peut être traduit par SELECT * en SQL).

C.2. Termes généraux

Parlons maintenant des termes généraux. A cet effet, considérons les objets de notre univers des membres de DVP : Antoine, Bruno, Frédéric, Philippe, ... A la suite de Quine, pour dire, de manière générale, qu'un membre de cet univers assure la fonction de modérateur nous utilisons ce que l'on appelle un *terme général* :

Quelqu'un est modérateur.

On peut à cette occasion remplacer le pronom « quelqu'un » par une variable :

x est modérateur.

Et produire un *schéma de phrase* (dans lequel la lettre « F » symbolise le terme général lui-même) :

Fx .

De même, pour dire, dans cet univers, qu'untel a la grande vertu d'être patient, on utilise un autre terme général :

x est patient.

Ce que l'on peut symboliser une fois de plus au moyen d'un schéma de phrase : $\cdot \cdot$

Gx .

De tels schémas peuvent être combinés à l'aide de fonctions de vérité (conjonction, disjonction, négation...) Ainsi, pour affirmer que quelqu'un est à la fois modérateur et patient, on pourrait être tenté d'écrire :

$Fx \cdot Gx$ (Le point symbolise la conjonction : « Fx et Gx »).

Mais attention, il y a un piège à utiliser deux fois la variable « x », car en réalité, cette combinaison de schémas doit être lue ainsi :

Quelqu'un est modérateur et quelqu'un est patient.

Ce qui signifie tout à fait autre chose, par exemple qu'Antoine est modérateur et Bruno patient : ça n'est pas parce qu'on a utilisé systématiquement « x » dans les schémas de phrases que l'on peut en déduire que l'on parle de la même personne. Les termes « x est modérateur », « x est patient » représentent encore ce qu'on appelle des *phrases ouvertes*, groupes de mots ni vrais ni faux, et qui n'attendent qu'une seule chose, être transformées en *phrases closes*, c'est-à-dire contrôlées par des quantificateurs permettant cette fois-ci de composer les schémas sans erreur d'interprétation et préciser si la variable « x » y représente effectivement le même objet.

Ainsi, grâce aux quantificateurs universel « $(\forall x)$ » et existentiel « $(\exists x)$ » (Merci, M. Frege !), on peut enfin combiner des schémas de phrases pour symboliser, sans ambiguïté, des compositions *vérifonctionnelles* de phrases (c'est-à-dire pouvant être vraies ou fausses). Par exemple :

$(\exists x)(Fx \cdot Gx)$ Quelques modérateurs sont patients. (Le point symbolise la conjonction « et »).

$(\exists x)(Fx \cdot \neg Gx)$ Quelques modérateurs ne sont pas patients. (« \neg » symbolise la négation).

$(\forall x)(Fx \rightarrow Gx)$ Tous les modérateurs sont patients. (La flèche symbolise le conditionnel « si ... alors »).

$(\forall x)(Fx \rightarrow \neg Gx)$ Aucun modérateur n'est patient.

Des variables telles que « x », « y », « z », etc., ne sont en fait que de simples pronoms utilisés pour renvoyer aux quantificateurs.

Quand une variable est soumise au contrôle d'un quantificateur, elle est dite *liée*, sinon elle est dite *libre*. Pour reprendre l'exemple des modérateurs patients, dans l'expression :

$(\exists x)(Fx \cdot Gx)$

la variable « x » est liée et l'on doit lire : « Il y a au moins quelqu'un qui est à la fois modérateur et patient ».

Mais attention, si l'on écrit :

$(\exists x)(Fx) \cdot Gx$

le schéma « Gx » n'est plus sous le contrôle du quantificateur, et si la variable « x » est liée dans « Fx », elle a été rendue libre dans « Gx ». Autrement dit, cette combinaison aurait pu être écrite ainsi, de façon strictement équivalente :

$$(\exists x)(Fx) \cdot Gy$$

C'est-à-dire, dans les deux cas : « Il y a au moins un modérateur et, par ailleurs, quelqu'un est patient ».



Les termes généraux « x est modérateur » et « x est patient » sont qualifiés de *monadiques* car ils ne mettent en jeu qu'une seule variable. Mais un terme général peut être affecté de deux, trois, n variables, auquel cas on dit qu'il est dyadique, triadique, n -adique (polyadique). Par exemple « x est F à l'égard de y » peut être noté « Fxy » et « x donne y à z » peut être noté « $Gxyz$ ».

A titre d'exemple, comme le propose Quine, penchons-nous sur l'expression (dans laquelle sont mis en jeu un terme général monadique et un terme général dyadique) :

$$(\forall x)[Fx \rightarrow (\exists y)(Fy \cdot Gxy)].$$

Si l'on interprète « Fx » comme « x est un nombre » et « Gxy » comme « x est plus petit que y », alors cette expression va signifier :

Tout nombre est tel qu'un nombre le dépasse.

De même, l'un des axiomes de l'identité est ainsi formulé :

$$(\forall x)(\forall y)(Fx \cdot Gxy \rightarrow Fy).$$

Où « Gxy » symbolise « $x = y$ », et l'on peut du reste préférer écrire simplement :

$$(\forall x)(\forall y)(Fx \cdot (x = y) \rightarrow Fy).$$

C.3. Concept de classe

Il est un concept qui ne nous est pas utile en logique du 1er ordre, celui de *classe*, mais qui permettra ultérieurement de traiter de problèmes qui échappent à cette logique. La notion de classe est une généralisation de la notion d'ensemble, mais pouvant aussi être perçue comme le nom d'un terme général : de même que « est membre du Club des développeurs » est un terme général, de même « DVP » peut être considéré comme le nom d'une classe.

En logique du 1er ordre, on n'a pas besoin de faire appel à la notion d'ensemble et à l'appartenance d'un élément à un ensemble, mais rien n'empêche d'écrire « $x \in \text{DVP}$ ». S'intéresser aux propriétés communes aux membres de DVP, peut, si on le souhaite, conduire à représenter ces propriétés par des classes et noter « $x \in P$ » le fait que l'objet x a la propriété P .

A la limite, un univers du discours U peut lui-même faire l'objet d'une classe universelle P_U pour laquelle on vérifierait la propriété universelle « est vrai de chaque objet ».

A partir de la table Membre (cf. paragraphe 2.2), créons maintenant des vues SQL correspondant à des restrictions sur la localisation des membres :

```
CREATE VIEW Toulouse
(MbrId, Pseudo, DateInscr, Loc, AdrCourriel) AS
SELECT  MbrId, Pseudo, DateInscr, Loc, AdrCourriel
FROM    Membre
WHERE   Localisation = 'Toulouse' ;

CREATE VIEW Bordeaux
(MbrId, Pseudo, DateInscr, Loc, AdrCourriel) AS
SELECT  MbrId, Pseudo, DateInscr, Loc, AdrCourriel
FROM    Membre
WHERE   Localisation = 'Bordeaux' ;
```

Etc.

Ces vues sont des tables virtuelles, donc des relations, donc des ensembles, donc des classes.

Intéressons-nous à une classe en particulier, par exemple celle des Toulousains. Pour utiliser QUEL une fois de plus :

```
RANGE OF x IS Toulouse
RETRIEVE (x ALL)
```

Instruction qui, comme on le sait, sera du reste transformée ainsi par le système relationnel :

```
RANGE OF x IS Membre
RETRIEVE (x ALL)
WHERE Localisation = 'Toulouse'
```

La variable « x » parcourt un univers que, pour les besoins de l'exemple, on a spécifié et nommé « Membre ». On pourrait aussi définir des variables « α », « β », etc., afin de pouvoir parcourir un univers $U_Localisation$ dont les éléments seraient les vues, donc les classes Toulouse, Bordeaux, etc., c'est-à-dire une vue qui serait une restriction portant sur la vue INFORMATION_SCHEMA.TABLE_CONSTRAINTS fournie par le système relationnel (cf. la norme SQL) :

```
RANGE OF  $\alpha$  IS U_Localisation
RETRIEVE ( $\alpha$  ALL)
```

C.4. Vers la logique du deuxième ordre : termes généraux considérés comme des objets

Examinons maintenant l'expression :

$$(\forall x)(\forall y)[(x = y) \leftrightarrow (\forall F)(Fx \leftrightarrow Fy)] \quad (\ll \leftrightarrow \gg \text{ symbolise le biconditionnel « si et seulement si »}).$$

Ce qui se lit : Quels que soient x et y , x est égal à y si et seulement si quel que soit le terme général F , il revient au même d'appliquer ce terme général à x ou à y . Initialement, les objets sur lesquels portait la quantification étaient Antoine, Bruno, etc. (voire des nombres), bref, des objets concrets et maintenant, on quantifie sur des termes généraux, abstraits, considérés à leurs tour comme des objets alors qu'ils représentent des propriétés des objets par ailleurs quantifiés (par exemple, être membre du Club des développeurs). On change de niveau, on entre en fait dans le domaine du 2e ordre.

Quantifier sur des termes généraux ne se fait pas à la légère. Procéder ainsi pourrait prêter à confusion et Quine attire notre attention à ce sujet. Dans ce qui suit, je fais encore référence à son ouvrage, plus précisément au chapitre 43, « Les classes ».

Supposons que la lettre schématique « F » tienne la place du terme général « aimer les chats ». La traduction en logique (du 1er ordre) de l'énoncé « si tout le monde aime les chats, alors quelqu'un aime les chats » est la suivante :

$$(\forall x)Fx \rightarrow (\exists x)Fx.$$

Si l'on écrit (en fait, dans l'esprit du 2e ordre) :

$$(\forall F)[(\forall x)Fx \rightarrow (\exists x)Fx]$$

alors pour Quine, « $(\forall F)$ » ne peut pas être interprété comme « chaque terme général F est tel que » car, je cite :

« Pouvons-nous adopter pour ' $(\forall F)$ ' la lecture 'chaque terme général (ou prédicat) F est tel que', et pour ' $(\exists F)$ ' une lecture correspondante ? Non, car ceci serait une confusion. ' F ' n'a jamais été conçu comme référant à des termes généraux (et donc comme tenant la place de *noms* de termes généraux), mais seulement comme tenant la place de termes généraux. S'il existait des objets d'un type spécial, mettons des garigous, dont des termes généraux seraient les noms, alors les lectures correctes de ' $(\forall F)$ ' et ' $(\exists F)$ ' seraient 'chaque garigou F est tel que' et 'quelque garigou F est tel que'. Mais la difficulté est que les termes généraux ne sont aucunement des noms. »

Si l'on veut que « F » prenne le statut de variable — ce qui revient à procéder à un détournement de son usage habituel — Quine propose alors d'utiliser des classes comme objets du parcours de cette variable. Je cite :

« Nous pouvons lire ' $(\forall F)$ ' et ' $(\exists F)$ ' respectivement 'chaque classe F est telle que' et 'quelque classe F est telle que', à condition simplement que pour les besoins de la cause nous relisions aussi ' Fx ' comme ' x est un membre de la classe F '. »

Pour éviter toute confusion, Quine propose de substituer à l'utilisation du terme général « F » celle d'un nom de classe, par exemple « α ».

Ainsi, au lieu de « $(\forall F)[(\forall x)Fx \rightarrow (\exists x)Fx]$ », on écrira « $(\forall \alpha)[(\forall x)(x \in \alpha) \rightarrow (\exists x)(x \in \alpha)]$ ».

Pour chaque objet x appartenant à la classe α , la variable « x » parcourt ici un certain univers U et la variable « α » un univers différent U_1 , dont chaque objet est une classe, à savoir une certaine propriété partagée par les objets de U .

Mais à propos du schéma :

$$(\forall \alpha)[(\forall x)(x \in \alpha) \rightarrow (\exists x)(x \in \alpha)]$$

on ne voit pas quelle en serait la valeur ajoutée par rapport au tout premier « $(\forall x)Fx \rightarrow (\exists x)Fx$ ». Quine exprime la chose ainsi :

« Si tous les énoncés formulables dans notre notation pour la théorie des classes pouvaient ainsi être ramenés à des expressions consistantes et valides de la théorie de la quantification, nous pourrions regarder notre théorie des classes comme une simple transcription pittoresque de notre théorie de la quantification, les classes n'auraient nul besoin d'être reconnues comme des entités sérieusement présupposées. »

Jusqu'ici, on pouvait donc se limiter aux termes généraux de la logique du 1er ordre et s'abstenir d'en passer par des classes. Par contre, Quine précise que la théorie des classes devient nécessaire, quand parmi les quantificateurs préfixes (c'est-à-dire quand les quantificateurs ont tous été regroupés en tête d'une formule quantifiée) figurent à la fois des quantificateurs universels et existentiels. Par exemple :

$$(\forall \alpha)(\exists \beta)(\forall x)[(x \in \alpha) \leftrightarrow (x \in \beta)]$$

$$(\forall x)(\forall y)(\exists \alpha)[(x \in \alpha) \leftrightarrow (y \in \alpha)]$$

On débouche en fait dans un système qui cette fois-ci est hors de portée de la logique du premier ordre.

Quine donne deux exemples nécessitant l'utilisation des classes. Par exemple, celui de la généalogie : « x est un ancêtre de y » (la récursivité montre le bout de son nez...), ou celui des gens qui s'admirent mutuellement (certaines personnes s'admirent l'une l'autre et n'en admirent aucune autre). A partir de l'exemple des ancêtres, on déboule en fait tout droit dans la théorie générale des classes, ou théorie des ensembles. La construction utilisée dans la définition du terme ancêtre fut utilisée par Frege (1879) pour être appliquée aux nombres, considérés comme classes de classes.

Il est légitime de penser que, dans le cadre du Modèle Relationnel de Données, il n'est pas nécessaire d'en arriver là. De fait, en 1970, Codd en est revenu à la logique du 1er ordre. S'il en était resté à son énoncé de 1969, il est vraisemblable que le Modèle Relationnel de Données aurait intéressé les chercheurs, mais n'aurait pas connu le succès qui fut le sien¹, et le plus célèbre de ses avatars, le modèle SQL, n'aurait pas vu le jour. Pour nos bases de données, à moins que les SGBD à Orientation Objet aient réussi à s'imposer, nous continuerions à exploiter essentiellement des systèmes hiérarchiques, listes inverses ou CODASYL. Pour avoir les avoir beaucoup fréquentés, j'en ai les jambes qui flageolent...

C.5. L'exemple de la généalogie et ses applications

Pour mieux percevoir la nécessité dans certaines circonstances d'utiliser des classes, on peut se pencher sur le cas de la généalogie proposé par Frege. Quine présente ainsi la chose, je cite :

« Comprenons 'ancêtre' dans un sens légèrement élargi, en comptant comme les ancêtres d'une personne non seulement ses parents, ses grands-parents, et ainsi de suite, mais aussi cette personne elle-même. Représentons 'parent' par ' F ', de façon que ' Fxy ' signifie ' x est parent de y '. [...]

Un trait important de la classe des ancêtres de ' y ' est que tous les parents de membres de la classe en sont membres à leur tour. Un autre de ses traits est que y lui même en fait partie. Mais ces deux traits ne déterminent pas encore la classe des ancêtres de y de façon unique, il existe des classes plus vastes qui contiennent à la fois y et tous les parents de leurs membres. Une classe de ce type est la classe des ancêtres des petits-fils de y . Un autre exemple est la classe combinée des ancêtres de y et des cravates ; car les

¹ Certains ont trouvé le Modèle relationnel trop simple et ont proposé de le remplacer par des modèles plus « puissants », mais qui au final sont restés dans des niches limitées à des sujets particuliers.

cravates étant sans parents, leur inclusion ne change rien au fait que tous les parents des membres sont des membres. Il est clair, en revanche, que toute classe qui contient y et tous les parents de ses membres devra au moins contenir tous les ancêtres de y , quelles que soient les autres choses qu'il puisse lui arriver de contenir. En outre, l'une de ces classes contient exclusivement les ancêtres de y . D'où la conséquence que pour être un ancêtre de y il est nécessaire et suffisant d'appartenir à toute classe contenant y et tous les parents de ses membres. En conséquence ' x est un ancêtre de y ' pourra s'écrire ainsi :

x appartient à toute classe contenant y et tous les parents de ses membres ;

C'est-à-dire :

$(\forall \alpha) [(y \in \alpha) \cdot (\text{tous les parents des membres de } \alpha \text{ appartiennent à } \alpha) \rightarrow (x \in \alpha)] ;$

et donc :

$(\forall \alpha) \{ (y \in \alpha) \cdot (\forall z)(\forall w) [(w \in \alpha) \cdot Fzw \rightarrow (z \in \alpha)] \rightarrow (x \in \alpha) \}.$

Cette construction ingénieuse est susceptible de nombreuses applications hors de la généalogie. [...] Mais l'aspect significatif de cette construction pour notre présente démarche est qu'elle fait un usage essentiel de la quantification d'une variable de classe ' α '. »

L'avant-dernier énoncé fourni par Quine peut être ainsi paraphrasé :

Quelle que soit la classe α , si y appartient à cette classe et si tous les parents des membres de cette classe en font aussi partie, alors le parent x de y appartient lui aussi à cette classe.

Dans le dernier énoncé, bien que cela ne soit pas strictement nécessaire, Quine procède à un changement de variables (apparition des variables w et z), sans doute pour mettre en évidence le fait que x et y sont libres d'un côté et liées quand elles sont utilisées avec des quantificateurs.

Dans la foulée, et toujours en se référant à Frege, Quine propose à son tour la formule qui sous-tend l'induction mathématique (« Nx » se lit « x est un nombre ») :

$Nx \leftrightarrow (\forall \alpha) \{ (0 \in \alpha) \cdot (\forall z) [(z \in \alpha) \rightarrow (1+z \in \alpha)] \rightarrow (x \in \alpha) \}$

C'est-à-dire : Pour toute classe α , si 0 appartient à α et si la proposition « si tout nombre z appartient à α alors son successeur appartient aussi à α » alors tout nombre appartient à α .

Autrement dit, être un nombre c'est appartenir à chaque classe à laquelle appartient 0 ainsi que le successeur de chaque membre de la classe.

C.6. Au sujet des RVA de Date et Darwen

Au paragraphe 2.6, on a vu que pour Date et Darwen, un attribut peut prendre une relation pour valeur, à condition de respecter la 1NF pour laquelle est mieux cerné le principe de l'atomicité (cf. paragraphe 2.7) : dans chaque tuple, chaque attribut contient exactement une valeur (groupes répétitifs interdits). Il n'y a pas *a priori* de problème de dérive vers la logique du deuxième ordre, car les opérateurs relationnels déjà en place suffisent : si on a besoin de manipuler des données emboîtées, telles que les livraisons de pièces (cf. « Inconvénients des RVA » au paragraphe 2.6), on utilise à cet effet le couple d'opérateurs GROUP/UNGROUP, lesquels ne sont jamais que des combinaisons d'opérateurs existants, des raccourcis, du « *syntactic sugar* ». Appliquée aux RVA, la formulation de requêtes du genre : « Quels fournisseurs ont livré quelles pièces ? », « Quelles pièces ont été livrées par quels fournisseurs ? », etc. ne pose aucun problème.

Il faudrait passer à une logique du 2e ordre si l'on était confronté à des problèmes comparables à ceux posés par A. Tarski. Je cite Gianbruno Guerrierio dans *Gödel : logique à la folie* (Pour la science, numéro 20) :

« La distinction entre axiomes du 1er ordre et du 2e ordre a été établie par le logicien polonais Alfred Tarski pour distinguer le langage-objet d'une étude, c'est-à-dire le langage utilisé pour parler d'objets quelconques, du métalangage correspondant, c'est-à-dire du langage utilisé pour parler du langage objet. Il existe même un méta-métalangage qui parle du métalangage, etc. »

Tarski a prouvé qu'à défaut, on tomberait dans le paradoxe du menteur (« Je mens »).



Cela dit, les RVA ne contiennent que des **valeurs** (relations), jamais de **variables** (relvars), et les logiciens que Date a consultés n'ont pas pu montrer que l'utilisation des RVA nécessitait d'en passer par une logique du 2e ordre.

D. La normalisation et le bonhomme NULL

Dans les énoncés des formes normales, il n'est pas fait mention des effets que peut produire la présence du bonhomme NULL dans les dépendances. Il n'est pas inutile de connaître quelques unes des réflexions de Codd et Date à ce sujet. On peut aussi consulter les ouvrages ou communications de chercheurs et auteurs tels que Yannis Vassiliou ou David Maier [Maier 1983]. Rappelons que ce que Codd appelle une A-mark est une marque (ou un marqueur) « applicable », dont on se sert quand une valeur est inconnue, marque qui peut être remplacée par une valeur réelle à tout moment (cas par exemple du numéro de SIRET de telle entreprise, numéro provisoirement inconnu). Une I-mark est une marque (ou un marqueur) « inapplicable » quand il n'y a pertinemment pas de valeur à fournir (cas par exemple du nom marital pour les salariés du sexe masculin).

Concernant Codd

Voici ce que l'on peut lire dans [Codd 1990] au paragraphe 8.21 « Normalization », page 193, traduisons :

Les concepts de dépendance fonctionnelle, multivaluée et de jointure, ainsi que les règles régissant celles-ci ont été développés sans que soient mentionnés les problèmes pouvant résulter de l'absence de valeurs.

Les formes normales basées sur ces dépendances ont été développées sans tenir compte non plus de l'absence éventuelle de valeurs. Est-ce que la possibilité de la présence de marques dans certaines colonnes (chaque marque signalant l'absence de valeur) peut saper l'ensemble de ces concepts et des théorèmes qui les utilisent ? Heureusement, la réponse est non : une marque n'est pas une valeur. Plus précisément, une marque dans une colonne C est sémantiquement différente d'une valeur dans C. Ainsi, d'une manière générale, les concepts utilisés dans la normalisation ne s'appliquent pas et ne devraient pas s'appliquer aux combinaisons de lignes et de colonnes contenant des marques. En revanche :

- *Les concepts de la normalisation devraient être utilisés au stade de la **version conceptuelle** de la base de données, auquel cas les lignes contenant des informations du type absent-mais-applicable dans les colonnes concernées ne sont pas à prendre en compte ;*
- *Ces concepts devraient en outre s'appliquer quand on cherche à remplacer une marque par une valeur.*

Lors d'une tentative d'insertion d'une ligne dans une relation, si une valeur est absente, il est sans objet pour le système de chercher à accepter ou rejeter cette ligne sur la base du respect ou du non respect des contraintes d'intégrité liées à une dépendance auxquelles est soumise telle ou telle colonne. Ce n'est que lorsqu'on cherche à remplacer une marque par une valeur que le système doit prendre les mesures qui s'imposent, en fonction de ces contraintes.

On pourrait être tenté de traiter les I-marks de façon différente des A-marks. Cependant, certains utilisateurs peuvent être autorisés à remplacer une I-mark par une valeur. Toutes ces marques doivent être traitées de la même façon, quel que soit leur type, en matière de test de contrainte de dépendance, que cette dernière soit fonctionnelle, multivaluée, de jointure ou d'inclusion. Pour chaque ligne contenant une marque dans les colonnes impliquées, le SGBD ne devrait réagir que lorsqu'une tentative est faite de remplacer une marque par une valeur réelle.

Quand Codd écrit :

*Les concepts de la normalisation devraient être utilisés au stade de la **version conceptuelle** de la base de données...*

on peut penser qu'il s'adresse à ceux qui conçoivent des diagrammes du type Entité/Relation. Quoi qu'il en soit, les concepteurs doivent effectivement être compétents en ce qui concerne la normalisation, ou à tout le moins pratiquer la double approche descendante/ascendante et ne jamais autoriser la présence du bonhomme NULL.

Codd poursuit (page 200 du même ouvrage), manifestement à l'attention de Date :

Les détracteurs semblent avoir rejeté, sans en fournir la raison, une troisième option qui est celle adoptée par le Modèle Relationnel. C'est-à-dire qu'à chaque fois que le composant A d'une ligne est manquant (ou le devient), le SGBD n'a pas à vérifier la dépendance fonctionnelle $A \rightarrow B$, tant qu'une tentative de remplacement de la marque (null) dans la colonne A par une valeur réelle n'a pas eu lieu.

L'exemple suivant a pour objet d'illustrer l'absence d'effet des valeurs absentes sur la normalisation.

La relation EMP identifie et décrit les employés. Trois de ses colonnes sont présentées :

E# : Numéro de l'employé (clé primaire)
D# : Numéro du département
CT : Type de contrat

Dans l'entreprise, à un département on affecte exactement un type de contrat. On fera référence à cette contrainte en l'appelant Règle 1. Une conséquence de cette règle est que la relation EMP n'est pas en troisième forme normale. Il existe les dépendances fonctionnelles suivantes :

$E\# \rightarrow D\# \rightarrow CT$

Il est à noter que le département D# auquel est affecté l'employé E# est une propriété immédiate de l'employé, tandis que le type de contrat CT est une propriété immédiate du département. Dans la colonne CT, figurent les valeurs g et n. Elles dénotent deux types de contrats, respectivement passés avec l'État ou non.

EMP	(E#	...	D#	CT)
	e1	...	d5	g
	e2	...	??	g
	e3	...	d2	n
	e4	...	d3	n
	e5	...	d2	n
	e6	...	??	n
	e7	...	d8	g

Dans cet exemple, les deux numéros manquants doivent être distincts, en vertu de la règle 1 et aussi parce que les types de contrats dans les deux lignes concernées sont différents. Les problèmes liés au contrôle des dépendances fonctionnelles quand il y a des valeurs absentes peuvent être totalement évités en différant le contrôle de conformité de chaque ligne avec la dépendance fonctionnelle $D\# \rightarrow CT$ jusqu'à ce que se produise une tentative de remplacement d'un numéro de département absent par une valeur réelle.

Quoi qu'il en soit, le piège s'est refermé, on ne peut plus normaliser EMP : le théorème de Heath n'est plus applicable, **intégrité d'entité** oblige (cf. paragraphe 3.2.6). Comme le reconnaît Codd, la décomposition de EMP en (E#, D#, ...) et (D#, CT) aurait dû avoir lieu dès le niveau conceptuel (MCD Merise, diagramme de classes UML...)

Concernant Date

Le « détracteur » avait déjà attiré notre attention sur le point que nous venons d'évoquer. Il présente un schéma fort intéressant, à la page 219 de [Date 1985] paragraphe 5.5 « Null values », et là encore le théorème de Heath est mis en échec, traduisons :

Si la relation R (A, B, C) satisfait à la dépendance fonctionnelle $A \rightarrow B$, alors R peut être décomposée sans perte [de contenu] selon ses projections R1 (A, B) et R2 (A, C), c'est-à-dire que l'on peut retrouver R par la jointure naturelle de R1 et de R2 sur A. Il est facile de vérifier que [ce] théorème ne tient plus si A peut prendre des valeurs nulles. Par exemple, si R est ainsi représentée :

R	A	B	C
	a1	b1	c1
	?	b2	c2

alors les projections R1 et R2 ainsi que leur jointure naturelle sont les suivantes :

R1	A	B
	a1	b1
	?	b2

R2	A	C
	a1	c1
	?	c2

R1 JOIN R2	A	B	C
	a1	b1	c1

⇒ Une fois de plus, anticipons au niveau conceptuel.

E. Fermeture des dépendances fonctionnelles, axiomes d'Armstrong, ensemble irréductible

E.1. Fermeture d'un ensemble de dépendances fonctionnelles

Avant de démontrer qu'une relvar R est en forme normale de Boyce-Codd (BCNF), il faut en passer par une tâche (théoriquement) incontournable (et prodigieusement ennuyeuse...), à savoir produire l'ensemble des dépendances fonctionnelles (DF) vérifiées par R . Pour cela nous disposons initialement de l'ensemble S des DF vérifiées par R , issues directement des règles de gestion des données de l'entreprise, et nous complétons cet ensemble en y ajoutant toutes les DF que l'on peut en inférer grâce à un système de règles, connues sous le nom d'**axiomes d'Armstrong**.

L'ensemble que nous obtenons au final est appelé **fermeture** (*closure*) de S , noté S^+ .

L'exercice consistant à calculer la fermeture en jonglant avec ces règles se révèle être rarement trivial. Chris Date décrit avec humour la méthode à employer : « *Appliquez ces règles de façon répétitive, jusqu'à ce qu'elles ne produisent plus de nouvelles DF...* », ce qui peut laisser sous-entendre que, plus le nombre d'attributs impliqués est élevé, ainsi que celui des DF, plus la recherche peut être longue et décourageante, autrement dit, calculer S^+ est incomparablement plus facile à dire qu'à faire, et l'exercice qui suit (cf. paragraphe E.3) en donne un avant-goût (fermeture de près de 30 éléments pour seulement trois attributs et deux DF...) On peut aussi s'exercer avec l'exemple proposé par Jeff Ullman (au départ, six attributs et 8 DF, cf. paragraphe E.5.1). Quoi qu'il en soit...

E.2. Axiomes d'Armstrong

Soit A, B, C, D des sous-ensembles quelconques d'attributs d'une relvar donnée R et notons AB l'union de A et de B . Les règles permettant de produire de nouvelles DF à partir d'un ensemble donné de DF sont les suivantes, et sont connues sous le nom d'axiomes d'Armstrong [Armstrong 1974], lesquels sont complets et valides [Ullman 1982] :

1. **Réflexivité** : si B est un sous-ensemble (non nécessairement strict) de A , alors $A \rightarrow B$.
2. **Augmentation** : si $A \rightarrow B$, alors $AC \rightarrow BC$
3. **Transitivité** : si $A \rightarrow B$ et $B \rightarrow C$ alors $A \rightarrow C$

- Ces axiomes sont **complets** en ce sens qu'ils permettent, à partir d'un ensemble donné S de DF vérifiées par R , de produire toutes les DF pouvant en être inférées, c'est-à-dire la fermeture S^+ de S .
- Ces axiomes sont **valides**, car ils ne produisent pas de DF parasites, c'est-à-dire non impliquées par S .

Des règles fort utiles peuvent être inférées des axiomes :

4. **Décomposition** : si $A \rightarrow BC$, alors $A \rightarrow B$ et $A \rightarrow C$
5. **Union** : si $A \rightarrow B$ et $A \rightarrow C$, alors $A \rightarrow BC$
6. **Pseudo-transitivité** : si $A \rightarrow B$ et $BC \rightarrow D$, alors $AC \rightarrow D$
7. **Composition** : si $A \rightarrow B$ et $C \rightarrow D$, alors $AC \rightarrow BD$

A titre d'exemple, la règle de décomposition peut être établie ainsi :

1. $A \rightarrow BC$ (donné)
2. $BC \rightarrow B$ (réflexivité)
3. $A \rightarrow B$ (1, 2 et transitivité)
4. $BC \rightarrow C$ (réflexivité)
5. $A \rightarrow C$ (1, 4 et transitivité)

De même, pour la règle de composition :

1. $A \rightarrow B$ (donné)
2. $AC \rightarrow BC$ (augmentation)
3. $C \rightarrow D$ (donné)
4. $BC \rightarrow BD$ (augmentation)
5. $AC \rightarrow BD$ (2, 4 et transitivité)

Ou encore, pour la règle de pseudo-transitivité :

1. $A \rightarrow B$ (donné)
2. $AC \rightarrow BC$ (augmentation)
3. $BC \rightarrow D$ (donné)
4. $AC \rightarrow D$ (2, 3 et transitivité)

E.3. Application des axiomes, calcul de la fermeture des DF

Concernant la recherche pour une relvar R de la fermeture S^+ à partir d'un ensemble S de DF, on se convainc rapidement que la tâche est de longue haleine. A titre d'exercice, prenons le cas de la bien modeste relvar EMP, utilisée pour décrire l'enseignement des matières à des étudiants par des professeurs, et qui ne contient que trois attributs (cf. paragraphe 3.7). Remplaçons les noms des attributs Etudiant, Matiere et Professeur respectivement par E, M et P. L'ensemble donné S des DF est le suivant :

$$\{E, M\} \rightarrow \{P\}$$

$$\{P\} \rightarrow \{M\}$$

a) Indépendamment de l'ensemble S, par application de l'axiome de **réflexivité**, on produit les DF triviales :

$$DF01 : \{E, M, P\} \rightarrow \{E, M, P\}$$

$$DF02 : \{E, M, P\} \rightarrow \{E, M\}$$

$$DF03 : \{E, M, P\} \rightarrow \{E, P\}$$

$$DF04 : \{E, M, P\} \rightarrow \{M, P\}$$

$$DF05 : \{E, M, P\} \rightarrow \{E\}$$

$$DF06 : \{E, M, P\} \rightarrow \{M\}$$

$$DF07 : \{E, M, P\} \rightarrow \{P\}$$

$$DF11 : \{E, P\} \rightarrow \{E, P\}$$

$$DF12 : \{E, P\} \rightarrow \{E\}$$

$$DF13 : \{E, P\} \rightarrow \{P\}$$

$$DF14 : \{M, P\} \rightarrow \{M, P\}$$

$$DF15 : \{M, P\} \rightarrow \{M\}$$

$$DF16 : \{M, P\} \rightarrow \{P\}$$

$$DF08 : \{E, M\} \rightarrow \{E, M\}$$

$$DF09 : \{E, M\} \rightarrow \{E\}$$

$$DF10 : \{E, M\} \rightarrow \{M\}$$

$$DF17 : \{E\} \rightarrow \{E\}$$

$$DF18 : \{M\} \rightarrow \{M\}$$

$$DF19 : \{P\} \rightarrow \{P\}$$

b) Les deux DF non triviales qui suivent composent S, elles appartiennent donc à S^+ :

$$DF20 : \{E, M\} \rightarrow \{P\}$$

$$DF21 : \{P\} \rightarrow \{M\}$$

c) Par application de l'axiome d'**augmentation** à DF20, on produit les DF suivantes :

$$DF22 : \{E, M\} \rightarrow \{E, P\} \quad (\text{augmentation par E})$$

$$DF23 : \{E, M\} \rightarrow \{M, P\} \quad (\text{augmentation par M})$$

$$DF24 : \{E, M\} \rightarrow \{E, M, P\} \quad (\text{augmentation par E et M})$$

d) Par application du même axiome à DF21, on produit les DF suivantes :

$$DF25 : \{P\} \rightarrow \{M, P\} \quad (\text{augmentation par P})$$

$$DF26 : \{E, P\} \rightarrow \{E, M\} \quad (\text{augmentation par E})$$

$$DF27 : \{E, P\} \rightarrow \{E, M, P\} \quad (\text{augmentation par E et P})$$

e) Par application de l'axiome de **transitivité** à DF13 et DF21, on produit la DF suivante :

$$DF28 : \{E, P\} \rightarrow \{M\}$$

f) Par application de l'axiome de **transitivité** à DF26 et DF23, on produit la DF suivante :

$$DF29 : \{E, P\} \rightarrow \{M, P\}$$

Ainsi, grâce aux seuls axiomes (réflexivité, augmentation, transitivité), on est à même de calculer la fermeture S^+ .

Pour en revenir au problème initial, à savoir prouver qu'une relvar R est en BCNF, on sait qu'il suffit que le déterminant de chaque DF non triviale vérifiée par R en soit un surclé (ou encore que le déterminant de chaque DF irréductible à gauche soit une clé candidate). Il était donc probablement inutile de fournir ici la (longue) liste des DF triviales, mais celles-ci pouvaient être utiles, à l'instar de DF13 qui a été impliquée dans la production de DF28. Bien sûr, on aurait pu obtenir ce résultat en appliquant, par exemple, à DF26 la règle de décomposition (mais celle-ci est inférée de l'axiome de réflexivité). Tous les coups sont bons. Quoi qu'il en soit, on peut désormais se concentrer sur l'ensemble des DF non triviales :

```
DF20 :    {E,M} → {P}
DF22 :    {E,M} → {E,P}
DF23 :    {E,M} → {M,P}
DF24 :    {E,M} → {E,M,P}
DF21 :    {P} → {M}
DF25 :    {P} → {M,P}
DF28 :    {E,P} → {M}
DF26 :    {E,P} → {E,M}
DF27 :    {E,P} → {E,M,P}
DF29 :    {E,P} → {M,P}
```

Dans ce sous-ensemble, on repère facilement les surclés (donc les clés candidates) de la relvar EMP. D'après DF24 et DF27, il s'agit de $\{E,M\}$ et $\{E,P\}$, car les dépendants de ces DF ont pour éléments tous les attributs de l'en-tête de EMP.

Par ailleurs, on voit tout de suite que la BCNF est violée car, par exemple, le déterminant $\{P\}$ de la DF DF21 n'est pas surclé. Certes on le savait dès le départ, mais pour autant, dans le cas général, en l'absence de S^+ — ou si l'on n'est pas certain de sa complétude — d'éventuelles DF fautives peuvent passer au travers des mailles du filet, auquel cas on ne peut pas jurer de la normalité d'une relvar. Toutefois, sachons qu'il est possible de s'assurer du respect de la BCNF tout en faisant l'impasse sur S^+ . En se basant seulement sur S, on peut adopter une stratégie moins coûteuse en temps de calcul, plus sûre et plus efficace, grâce à l'algorithme précieux et en fait incontournable qui suit.

E.4. Fermeture d'un ensemble d'attributs, l'algorithme du seau à dépendants

Comme on vient de le voir, calculer la fermeture S^+ d'un ensemble S de DF vérifiées par une relvar R peut très rapidement s'avérer mission impossible, lorsque le nombre d'attributs de l'en-tête de R croît, ainsi que le nombre de DF qu'elle vérifie.

Heureusement, étant donné un sous-ensemble Z des attributs de l'en-tête de R et l'ensemble S des DF vérifiées par R, on dispose d'un algorithme (conçu par Philip Bernstein [Bernstein 1976]), permettant de déterminer de façon très simple l'ensemble des attributs de R dépendant fonctionnellement de Z, ce que l'on appelle **la fermeture Z^+ de Z par rapport à S**. Sans avoir à en passer par les axiomes d'Armstrong, on pourra de façon mécanique, confirmer ou infirmer l'existence d'une hypothétique DF $X \rightarrow Y$, selon qu'au résultat cette DF appartient ou non à la fermeture X^+ de X par rapport S. Et, cerise sur le gâteau, si Y contient l'ensemble des attributs de l'en-tête de R, alors (par référence à la définition de la surclé) X est surclé.

L'algorithme en question permet donc de trouver l'ensemble des attributs qui sont fonctionnellement dépendants de Z. Le résultat est fourni dans une variable, appelons-la V. La méthode est la suivante (voir par exemple [Date 2004], paragraphe 11.5, [Maier 1983], « Algorithm 4.2 », [Ullman 1982], « Algorithm 7.1 », « Theorem 7.2 » pour en démontrer la validité) :

```
V := Z ;
Repeat
  For each dépendance fonctionnelle  $X \rightarrow Y$  appartenant à S
    Do
      Begin
        If  $X \subseteq V$ 
          Then  $V := V \cup Y$  ;
      End
Until V n'a pas changé au cours de l'itération
```


Illustrons à l'aide d'un exemple utilisé par Chris Date. La relvar R est composée des attributs A, B, C, D, E, F. L'ensemble S de DF donné est le suivant :

$\{A\} \rightarrow \{B,C\}$
 $\{E\} \rightarrow \{C,F\}$
 $\{B\} \rightarrow \{E\}$
 $\{C,D\} \rightarrow \{E,F\}$

Calculons par exemple la fermeture $\{A,B\}^+$ par rapport à S :

1. On initialise la variable V avec la valeur " $\{A,B\}$ ". De façon imagée, V est comme un seau dans lequel on jette la paire $\{A,B\}$. Le but est d'amorcer la pompe qui servira ensuite à y aspirer un maximum d'attributs de l'en-tête de R, lesquels pourront ainsi constituer un dépendant pour une DF ayant $\{A,B\}$ comme déterminant. Cet amorçage met systématiquement à profit l'axiome de réflexivité, laquelle nous permet d'affirmer que $\{A,B\} \rightarrow \{A,B\}$ donc, avec une pelle à dépendants, ramasser le dépendant de cette DF, c'est-à-dire la paire $\{A,B\}$ et la jeter dans le seau.

Pour le moment, on sait simplement que $\{A,B\} \rightarrow \{A,B\}$, ce qui est trivial, mais il faut un début à tout.

2. Suite des opérations :

Pour la première DF, $\{A\} \rightarrow \{B,C\}$, parce que le déterminant $\{A\}$ est déjà dans le seau, on peut y jeter à son tour le dépendant $\{B,C\}$. Par union avec $\{A,B\}$, le seau contient maintenant $\{A,B,C\}$. Cette capture est valide du fait que :

- $\{A,B\} \rightarrow \{A\}$ (réflexivité) ;
- La DF $\{A\} \rightarrow \{B,C\}$ est donnée, donc par transitivité on produit la DF $\{A,B\} \rightarrow \{B,C\}$;
- Et par augmentation : $\{A,B\} \rightarrow \{A,B,C\}$.

On sait maintenant que $\{A,B\} \rightarrow \{A,B,C\}$, ce qui n'est pas forcément trivial à montrer sans l'algorithme.

3. Pour la DF suivante $\{E\} \rightarrow \{C,F\}$, le déterminant $\{E\}$ n'étant pas dans le seau, on ne peut pas y jeter C et F.
4. $\{A,B,C\}$ constitue le contenu actuel du seau. Pour la DF suivante $\{B\} \rightarrow \{E\}$, le déterminant $\{B\}$ étant dans le seau en tant que sous-ensemble de $\{A,B,C\}$, on peut y jeter le dépendant $\{E\}$. Le seau contient maintenant $\{A,B,C,E\}$. La validité de la capture met encore en jeu l'enchaînement réflexivité, transitivité, augmentation :
 - $\{A,B,C\} \rightarrow \{B\}$ (réflexivité) ;
 - La DF $\{B\} \rightarrow \{E\}$ est donnée, donc par transitivité on produit la DF $\{A,B,C\} \rightarrow \{E\}$;
 - Et par augmentation, $\{A,B,C\} \rightarrow \{A,B,C,E\}$.
5. Pour la DF suivante $\{C,D\} \rightarrow \{E,F\}$, comme $\{C,D\}$ n'est pas un sous-ensemble de $\{A,B,C,E\}$, on ne peut que passer à la suite.
6. Chaque DF a été examinée et l'on repart pour un tour de manège. A la première itération, le contenu du seau ne change pas. A la deuxième itération, du fait que le déterminant $\{E\}$ de la DF $\{E\} \rightarrow \{C,F\}$ est cette fois-ci dans le seau, toujours en enchaînant réflexivité, transitivité et augmentation, on peut y jeter $\{C,F\}$. Le contenu du seau devient $\{A,B,C,E,F\}$. Les troisième et quatrième itérations laissent le contenu du seau inchangé.
7. On repart pour un tour de manège complet, lequel n'apporte rien de nouveau : le traitement est terminé avec un seau ayant pour contenu $\{A,B,C,E,F\}$.

On a prouvé que $\{A,B\} \rightarrow \{A,B,C,E,F\}$ et qu'on ne peut pas faire mieux avec le déterminant $\{A,B\}$.

Ce qui est sûr et constitue une information précieuse, c'est qu'à partir de l'ensemble S des DF qui ont été données, $\{A,B\}$ ne peut pas constituer une surclé (*a fortiori* une clé candidate), car en fin de parcours le seau ne contient pas tous les attributs de R : l'attribut D manque en effet à l'appel.

Grâce à l'algorithme, on peut aussi montrer que $\{A\} \rightarrow \{A,B,C,E,F\}$, c'est-à-dire que $\{A\}^+ = \{A,B\}^+$ par rapport à S.

Méthode pratique

Pour ne pas perdre de temps on peut procéder de façon pragmatique. Prenons par exemple le cas de $\{A\}^+$:

Prévoir dans le seau une place pour chaque attribut de la relvar R et amorcer la pompe à partir de la DF triviale $\{A\} \rightarrow \{A\}$. Le contenu du seau ressemble à ceci :

A _ _ _ _ _

Capturer les attributs B et C parce que l'attribut A est dans le seau et qu'il existe la DF $\{A\} \rightarrow \{B,C\}$:

A B C _ _ _

Capturer l'attribut E parce que l'attribut B est dans le seau et qu'il existe la DF $\{B\} \rightarrow \{E\}$:

A B C _ E _

Capturer l'attribut F parce que l'attribut E est dans le seau et qu'il existe la DF $\{E\} \rightarrow \{C,F\}$:

A B C _ E F

En revanche, rien ne permet de capturer l'attribut D : fin des opérations concernant $\{A\}^+$.

Autre exemple. On montre sans difficulté que $\{A,D\}$ est une **surclé** de la relvar R en procédant ainsi :

Amorcer la pompe :

A _ _ D _ _

Capturer les attributs B et C parce que l'attribut A est dans le seau et qu'il existe la DF $\{A\} \rightarrow \{B,C\}$:

A B C D _ _

Capturer les attributs E et F parce que les attributs C et D sont dans le seau et qu'il existe la DF $\{C,D\} \rightarrow \{E,F\}$:

A B C D E F

Autrement dit, $\{A,D\} \rightarrow \{A,B,C,D,E,F\}$ et $\{A,D\}$ est une surclé pour R (et c'est aussi une clé candidate, car $\{A,D\}^+$ est différent de $\{A\}^+$ et $\{D\}^+$). Par comparaison, on se convaincra facilement que sans l'algorithme, parvenir à ce résultat demande beaucoup plus de réflexion. Par exemple :

1. $\{A,D\} \rightarrow \{A\}$ (réflexivité)
2. $\{A\} \rightarrow \{B,C\}$ (donné)
3. $\{A,D\} \rightarrow \{B,C,D\}$ (2, augmentation)
4. $\{A,D\} \rightarrow \{A,B,C,D\}$ (3, augmentation)
5. $\{A\} \rightarrow \{B\}$ (2, décomposition)
6. $\{B\} \rightarrow \{E\}$ (donné)
7. $\{A\} \rightarrow \{E\}$ (5, 6, transitivité)
8. $\{A,D\} \rightarrow \{D,E\}$ (7, augmentation)
9. $\{A,D\} \rightarrow \{A,B,C,D,E\}$ (4, 8, union)
10. $\{E\} \rightarrow \{C,F\}$ (donné)
11. $\{B\} \rightarrow \{C,F\}$ (6, 10, transitivité)
12. $\{A\} \rightarrow \{C,F\}$ (5, 11, transitivité)
13. $\{A,D\} \rightarrow \{C,D,F\}$ (12, augmentation)
14. $\{A,D\} \rightarrow \{A,B,C,D,E,F\}$ (9, 13, union)

Supposons encore que l'on ait l'intuition que la DF $\{C,D\} \rightarrow \{F\}$ puisse être inférée d'autres DF et donc être évacuée de l'ensemble S de DF proposé ci-dessus, lequel se réduirait par conséquent à T :

$\{A\} \rightarrow \{B,C\}$
 $\{E\} \rightarrow \{C,F\}$
 $\{B\} \rightarrow \{E\}$
 $\{C,D\} \rightarrow \{E\}$

On montre qu'il en est ainsi, à partir de la fermeture $\{CD\}^+$ par rapport à T. Au départ, le seau contient les attributs C et D :

_ _ C D _ _

L'attribut E peut être capturé parce que les attributs C et D sont dans le seau et qu'il existe la DF $\{C,D\} \rightarrow \{E\}$, obtenue en appliquant la règle de décomposition à la DF $\{C,D\} \rightarrow \{E,F\}$:

_ _ C D E _

L'attribut F peut être capturé parce l'attribut E est dans le seau et qu'il existe la DF $\{E\} \rightarrow \{C,F\}$:

_ _ C D E F

En conséquence, $\{C,D\} \rightarrow \{C,D,E,F\}$. On applique alors la règle de décomposition, et l'on confirme ainsi que $\{C,D\} \rightarrow \{F\}$. La réduction du nombre de DF fait l'objet de la recherche par algorithme des ensembles irréductibles de DF (cf. paragraphe E.6).

N.B. Pour en revenir à l'exemple de la relvar EMP (cf. paragraphe E.3), la technique du seau permet de découvrir facilement quelque chose qui n'est pas forcément intuitif, à savoir que $\{E,P\}$ en est une clé candidate (donc une surclé).

E.5. Inventaire des clés et surclés. Quelques observations.

E.5.1. La technique du rouleau compresseur

A l'aide de l'algorithme du seau on vérifie rapidement si au sein de l'ensemble S des DF associé à une relvar R, il en est qui provoquent un viol de BCNF. Partons d'un exemple proposé par Jeff Ullman (cf. [Ullman 1982], « Computing closures »), dans lequel l'ensemble S des DF d'une relvar U $\{A, B, C, D, E, G\}$ est le suivant :

DF01 : $\{A,B\} \rightarrow \{C\}$	DF05 : $\{D\} \rightarrow \{E,G\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF06 : $\{B,E\} \rightarrow \{C\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF07 : $\{C,G\} \rightarrow \{B,D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF08 : $\{C,E\} \rightarrow \{A,G\}$

On met très vite en évidence le fait que DF02 et DF05 sont fautives car, contrairement aux autres DF, la fermeture de leur déterminant n'est pas égale à $\{A,B,C,D,E,G\}$. On pourrait en rester là et normaliser, mais on peut aussi pousser la curiosité jusqu'à dresser l'inventaire complet des surclés de U (en fait, la connaissance de ses clés candidates suffit, puisque par augmentation/décomposition chaque surclé est inférée d'une clé candidate).

Toujours à l'aide de l'algorithme du seau et en passant en revue les fermetures, on peut dresser cet inventaire de façon mécanique, sans se poser de questions, pas plus qu'on ne s'en pose avec un rouleau compresseur pour écraser des noix. Roulons donc...

1) On traite par exemple en premier la fermeture par rapport à S de chaque singleton pour vérifier s'il peut être clé candidate :

$\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{G\}$

et l'on constate très rapidement qu'aucun d'eux ne fait l'objet d'une telle clé : $\{A\} \nrightarrow \{B,C,D,E,G\}$, etc.

2) On passe ensuite en revue la fermeture par rapport à S de chaque paire :

$\{A,B\}, \{A,C\}, \{A,D\}, \{A,E\}, \{A,G\}, \{B,C\}, \{B,D\}, \{B,E\}, \{B,G\}, \{C,D\}, \{C,E\}, \{C,G\}, \{D,E\}, \{D,G\}, \{E,G\}$.

Grâce à l'algorithme, du seau, on découvre que 7 de ces 15 paires, à savoir $\{A,B\}, \{B,C\}, \{B,D\}, \{B,E\}, \{C,D\}, \{C,E\}, \{C,G\}$ constituent autant de clés candidates. Ainsi $\{A,B\}^+ = \{A,B,C,D,E,G\}$, c'est-à-dire $\{A,B\} \rightarrow \{A,B,C,D,E,G\}$, etc.

3) On passe ensuite en revue la fermeture par rapport à S de chaque triplet qui ne constitue pas une surclé pouvant être inférée des clés candidates que l'on vient d'énumérer :

$\{A,D,E\}, \{A,D,G\}, \{A,E,G\}, \{D,E,G\}$.

Toujours avec le même algorithme, on montre qu'aucun de ces triplets ne constitue une clé candidate (donc une surclé). Le travail est *de facto* terminé (et s'il avait fallu poursuivre, on aurait soumis au même régime les quadruplets, puis les quintuplets, etc.)

Au final, on a mis en évidence toutes les clés candidates de U (soit 7 clés), sans avoir à calculer la fermeture S^+ . L'exercice n'était pas inutile, car au moins sait-on dresser l'inventaire complet des clés candidates de la relvar, grâce auquel on peut mener à son terme le travail de normalisation.

E.5.2. Cas des attributs ne figurant pas dans les dépendants des DF

Considérons une relvar R dotée d'un ensemble S de DF non triviales, dans lequel il existe des attributs qui sont des éléments de déterminants de ces DF sans être des éléments des dépendants d'autres DF (n'est donc pas concerné l'exemple du paragraphe E.5.1, dans lequel tous les attributs qui sont des éléments des déterminants des DF sont aussi des éléments des dépendants d'autres DF). Si cette situation se présente, on peut accélérer la constitution de la liste des clés (et surclés) de R.

Reprenons l'exemple fourni par Chris Date (paragraphe E.4). A l'aide du rouleau compresseur et du seau à dépendants, on constate rapidement qu'aucun des singletons $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}$ n'est clé.

Passons aux paires :

$\{A,B\}, \{A,C\}, \{A,D\}, \{A,E\}, \{A,F\}, \{B,C\}, \{B,D\}, \{B,E\}, \{B,F\}, \{C,D\}, \{C,E\}, \{C,F\}, \{D,E\}, \{D,F\}, \{E,F\}$.

A l'exception de $\{A,D\}$, inutile de les soumettre à l'épreuve du seau. On observe en effet que ni l'attribut A ni l'attribut D ne sont des éléments du dépendant de quelque DF que ce soit de S, en conséquence de quoi, l'algorithme ne pourra jamais faire tomber ces deux attributs dans le seau à partir des paires énumérées autres que $\{A,D\}$. Par exemple, la fermeture $\{A,B\}^+$ nous permet au mieux de produire $\{A,B,C,E,F\}$ dont D est exclu ; même principe pour l'ensemble des paires autres que $\{A,D\}$, la seule finalement qui puisse constituer une clé candidate (ou participer à une surclé) : soumettre les autres paires à l'épreuve du seau serait peine perdue. Pour la même raison, inutile de passer en revue quelque triplet, quadruplet, quintuplet ou sextuplet que ce soit. En revanche, on peut énumérer les surclés de R : la paire $\{A,D\}$ et ses sur-ensembles : les triplets $\{A,B,D\}$, $\{A,C,D\}$, $\{A,E,D\}$ et $\{A,F,D\}$, les quadruplets $\{A,B,D,E\}$, $\{A,B,D,F\}$, $\{A,C,D,E\}$, $\{A,C,D,F\}$ et $\{A,D,E,F\}$, les quintuplets $\{A,B,C,D,E\}$, $\{A,B,C,D,F\}$ et $\{A,C,D,E,F\}$, et enfin le sextuplet $\{A,B,C,D,E,F\}$. Et l'on peut ainsi dénombrer les infractions concernant le respect de la BCNF...

En résumé :



Si un attribut X d'une relvar R appartient au déterminant d'une DF quelconque de l'ensemble S des DF (non triviales) vérifiées par R, mais n'appartient à aucun dépendant d'une de ces DF, alors il participe à chaque clé candidate de R. Reste à chercher les éventuels attributs en compagnie desquels X pourrait entrer dans la composition d'une clé candidate (plus généralement d'une surclé), tâche grandement facilitée par l'utilisation de l'algorithme du seau.

E.5.3. Surclés n'ayant pas d'attributs en commun et utilisation de l'algorithme du seau

Attention quand même aux chausse-trapes et aux trous dans le seau. Reprenons l'exemple fourni par Chris Date (paragraphe E.4). La relvar R $\{A,B,C,D,E,F\}$ a pour seule clé la paire $\{A,D\}$ — ce qui est facile à vérifier à l'aide de l'algorithme du seau. Mais certains concepteurs ont la manie des clés (primaires au sens SQL) singletons : à cet effet, ils ajoutent d'office un attribut, appelons-le K, tel que R a désormais pour en-tête $\{A,B,C,D,E,F,K\}$ tout en étant dotée cette fois-ci de deux clés candidates $\{A,D\}$ et $\{K\}$. Attention, dans l'exemple de Chris Date, l'ensemble S des DF initial est le suivant :

$\{A\} \rightarrow \{B,C\}$
 $\{E\} \rightarrow \{C,F\}$
 $\{B\} \rightarrow \{E\}$
 $\{C,D\} \rightarrow \{E,F\}$

Mais pour tenir compte de K, S doit être aménagé, sinon, en l'état et en utilisant l'algorithme du seau, il n'y aura toujours qu'une clé pour R, à savoir le triplet $\{A,D,K\}$, ce qui n'est pas franchement le but recherché. Pour que l'algorithme fonctionne, S doit être enrichi en tenant compte des clés à produire :

$\{A\} \rightarrow \{B,C\}$
 $\{E\} \rightarrow \{C,F\}$
 $\{B\} \rightarrow \{E\}$
 $\{C,D\} \rightarrow \{E,F\}$
 $\{A,D\} \rightarrow \{K\}$
 $\{K\} \rightarrow \{A,D\}$

Ainsi, $\{K\}^+ = \{A,D\}^+ = \{A,B,C,D,E,F,K\}$, donc $\{K\}$ et $\{A,D\}$ sont bien clés candidates et le concepteur est satisfait.

E.5.4. Clés oubliées

Bien que son objet premier ne soit pas la production de la fermeture de l'ensemble des DF associées à une relvar, l'algorithme du seau se révèle indispensable car, comme on l'a vu, il permet de réduire de façon très simple et pragmatique le champ de recherche des clés et surclés. Pour reprendre l'exemple du paragraphe E.4 mais en tenant compte des observations faites dans le paragraphe E.5.2, découvrir que la paire d'attributs $\{A,D\}$ constitue une clé candidate (donc une surclé) de R n'est pas intuitif, mais se fait de façon naturelle grâce à l'algorithme et au rouleau compresseur. Une fois les techniques (quand même simples) maîtrisées, et les chausse-trapes potentielles identifiées, trouver la liste exhaustive des clés et surclés est généralement rapide. Il est probable que nombre de bases de données en production contiennent des clés que l'on n'a malheureusement pas déclarées au SGBD, parce que l'on n'aura pas cherché à les repérer : pour chaque table SQL, on se sera contenté de définir une clé primaire (singleton de surcroît), mais pour le reste...

Quoi qu'il en soit, l'algorithme du seau est au cœur de la méthode utilisée pour réduire l'ensemble des DF associées à une relvar, sujet traité dans le paragraphe E.6.

E.6. Ensemble irréductible de dépendances fonctionnelles

E.6.1. Ensembles de DF et redondances

Considérons la relvar R {A,B,C,D} dotée de l'ensemble S de DF :

DF01 : {A} → {B}
 DF02 : {B,C} → {D}
 DF03 : {A,C} → {D}

DF03 peut être inférée de DF01 et DF02. En effet :

1	{A} → {B}	(donné)
2	{A,C} → {B,C}	(augmentation)
3	{B,C} → {D}	(donné)
4	{A,C} → {D}	(2, 3 et transitivité)

S est donc **réductible** et peut être remplacé par le sous-ensemble I :

DF01 : {A} → {B}
 DF02 : {B,C} → {D}

Autrement dit, les fermetures S^+ et I^+ par rapport à R sont égales. (Au passage, on aura reconnu la règle de pseudo-transitivité, qui sert à illustrer notre propos). L'important dans cette affaire est qu'en remplaçant S par I, lorsqu'on demandera au SGBD de garantir DF01 et DF02, automatiquement il garantira aussi DF03, sans qu'on lui impose explicitement le surcroît de travail qui résulterait de notre ignorance de l'existence de I.

Le but de la manœuvre est donc de s'assurer que l'on demande au SGBD de veiller seulement au respect d'un **ensemble irréductible** de dépendances fonctionnelles, qu'on lui soumet sous forme de contraintes. Comme on vient de le voir, il serait en effet dommage de le surcharger inutilement à cause de contraintes **redondantes** car pouvant être inférées.



Maintenant, Jeff Ullman écrit : « Étant donnés les ensembles de dépendances S et I, ces ensembles sont équivalents si et seulement si chaque dépendance appartenant à S appartient aussi à I^+ et chaque dépendance appartenant à I appartient aussi à S^+ » ([Ullman 1982], « Covers of Sets of Dependencies »).

Concrètement, on doit donc chercher (s'il existe) le plus petit sous-ensemble I de S dont la fermeture I^+ est égale à S^+ , pour ainsi se débarrasser de toutes les DF inutiles en remplaçant S par I.

L'utilisation des seuls axiomes d'Armstrong et des règles qui en sont inférées est évidemment possible pour résoudre ce genre de problème, mais on perd le plus souvent beaucoup trop de temps, ne serait-ce que pour traiter de cas *a priori* réputés simples. Ainsi, après l'exemple que nous venons de voir, considérons celui-ci :

Soit la relvar R {A,B,C,D} et l'ensemble S de DF { {A,C,D} → {B}, {C} → {A} }. La DF {A,C,D} → {B} est réductible à {C,D} → {B}, car par augmentation {C} → {A} engendre {C,D} → {A,C,D}, et comme {A,C,D} → {B}, par transitivité {C,D} → {B}. A moins de se sentir inspiré, on se munira d'un tube aspirine avant de se lancer dans la résolution d'exercices plus corsés. Mais heureusement, on peut contourner les axiomes et règles d'Armstrong. Il existe pour cela une alternative qui consiste à mettre en oeuvre la méthode décrite dans le paragraphe suivant (encore un genre de rouleau compresseur...)

N.B. Nous reprenons de Chris Date l'expression « ensemble irréductible », mais on trouve chez les auteurs des expressions équivalentes : « couverture irréductible », « couverture minimale », « couverture irredondante », etc.)

E.6.2. Propriétés d'un ensemble irréductible de DF

Un ensemble I de dépendances fonctionnelles associé à une relvar R est dit irréductible sous certaines conditions :

1. Le dépendant (partie droite) de chaque DF appartenant à I ne doit contenir qu'un seul attribut.
2. Le déterminant D (partie gauche) de chaque DF appartenant à I doit être irréductible, c'est-à-dire qu'aucun attribut ne peut être éliminé de D s'il en résulte une transformation de I en J telle que $J^+ \neq I^+$ par rapport à R. Une telle DF est dite **irréductible à gauche** (*left irreducible*), ou **élémentaire** ou **totale**. (Se reporter aussi au paragraphe 3.2.2). Inversement, si l'élimination d'un attribut de D (voire plusieurs) donne lieu à une transformation de I en J telle que $J^+ = I^+$, alors la DF est réductible.
3. Une DF ne peut être éliminée de I s'il en résulte une transformation de I en J telle que $J^+ \neq I^+$ par rapport à R.

Le point 1 ne se justifie que pour faciliter la production de I. Une fois le travail terminé, on pourra regrouper dans un même dépendant tous les dépendants singletons ayant même déterminant (règle d'union).

Quand on traite du point 2, il se peut qu'on produise plus d'un ensemble irréductible (ou candidat à l'irréductibilité). Ainsi, étant donnée $R \{A,B,C\}$ et l'ensemble de dépendances fonctionnelles $S = \{\{A,B\} \rightarrow \{C\}, \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{A\}\}$, on produit les ensembles irréductibles : $I = \{\{A\} \rightarrow \{C\}, \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{A\}\}$ et $J = \{\{B\} \rightarrow \{C\}, \{A\} \rightarrow \{B\}, \{B\} \rightarrow \{A\}\}$.

Quand on traite du point 3, on peut là aussi produire plus d'un ensemble irréductible (voir le paragraphe E.6.3, où l'on reprend l'exemple proposé par Ullman et dont on s'est déjà servi au paragraphe E.5.1).

Illustration de la méthode

Pour bien comprendre la méthode utilisée en relation avec ces trois conditions et aboutir à un résultat conforme, servons-nous encore d'un exemple proposé par Chris Date.

Soit une relvar $R \{A,B,C,D\}$ et S l'ensemble de DF que l'on demande au système de garantir.

$$S = \{ \{A\} \rightarrow \{B,C\} \\ \{B\} \rightarrow \{C\} \\ \{A\} \rightarrow \{B\} \\ \{A,B\} \rightarrow \{C\} \\ \{A,C\} \rightarrow \{D\} \}$$

Cherchons un ensemble irréductible I de DF.

Première étape

On transforme S , de telle sorte que le dépendant de chaque DF soit singleton (application de la règle de décomposition) :

$$S = \{ \text{DF01} : \{A\} \rightarrow \{B\} \\ \text{DF02} : \{A\} \rightarrow \{C\} \\ \text{DF03} : \{B\} \rightarrow \{C\} \\ \text{DF04} : \{A\} \rightarrow \{B\} \\ \text{DF05} : \{A,B\} \rightarrow \{C\} \\ \text{DF06} : \{A,C\} \rightarrow \{D\} \}$$

La DF $\{A\} \rightarrow \{B\}$ étant présente deux fois, on n'en conserve qu'une seule occurrence, disons DF01 :

$$S = \{ \text{DF01} : \{A\} \rightarrow \{B\} \\ \text{DF02} : \{A\} \rightarrow \{C\} \\ \text{DF03} : \{B\} \rightarrow \{C\} \\ \text{DF05} : \{A,B\} \rightarrow \{C\} \\ \text{DF06} : \{A,C\} \rightarrow \{D\} \}$$

Deuxième étape

On réduit le déterminant de chaque DF pour laquelle c'est possible :

Sont à considérer les DF dont le déterminant comporte plus d'un attribut, à savoir DF05 et DF06.

a) Cas de DF05 : $\{A,B\} \rightarrow \{C\}$.

DF05 est-elle réductible à $\{A\} \rightarrow \{C\}$? Considérons le sous-ensemble I de S , dans lequel on remplace DF05 par DF05a : $\{A\} \rightarrow \{C\}$. S et I sont respectivement représentés par :

$S = \{ \text{DF01} : \{A\} \rightarrow \{B\} \\ \text{DF02} : \{A\} \rightarrow \{C\} \\ \text{DF03} : \{B\} \rightarrow \{C\} \\ \text{DF05} : \{A,B\} \rightarrow \{C\} \\ \text{DF06} : \{A,C\} \rightarrow \{D\} \}$	$I = \{ \text{DF01} : \{A\} \rightarrow \{B\} \\ \text{DF02} : \{A\} \rightarrow \{C\} \\ \text{DF03} : \{B\} \rightarrow \{C\} \\ \text{DF05a} : \{A\} \rightarrow \{C\} \\ \text{DF06} : \{A,C\} \rightarrow \{D\} \}$
---	--

Par référence à Ullman (cf. paragraphe E.6.1) les ensembles S et I sont équivalents si et seulement si chaque DF appartenant à S appartient aussi à I^+ et chaque DF appartenant à I appartient aussi à S^+ . Dans le cas de DF05 et DF05a, S et I sont donc équivalents si et seulement si la fermeture $\{A\}^+$ de {A} par rapport à S est égale à la fermeture $\{A\}^+$ de {A} par rapport à I. S'il y a inégalité, cela est dû à la composition différente des déterminants de DF05 et DF05a (seules DF à présenter une différence). De fait, contrairement au déterminant {A} de DF05a, la partie {A} du déterminant {A,B} de DF05 ne permet pas à elle seule de capturer l'attribut C : si possible, une autre DF devra compenser pour rattraper le coup.

Qu'en est-il ? Au moyen de l'algorithme du seau, calculons $\{A\}^+$ par rapport à S :

$\{A\}^+ = \{A\}$, puis {A,B} du fait de DF01, puis {A,B,C} du fait de DF02, puis {A,B,C,D} du fait de DF06.

De même, calculons $\{A\}^+$ par rapport à I :

$\{A\}^+ = \{A\}$, puis {A,B} du fait de DF01, puis {A,B,C} du fait de DF02, puis {A,B,C,D} du fait de DF06.

On constate ainsi que $\{A\}^+$ par rapport à S égale $\{A\}^+$ par rapport à I : Les deux ensembles S et I sont équivalents par rapport à leurs fermetures et I peut donc remplacer S. Comme par ailleurs DF05a existe déjà en tant que DF02 (ce qui fait qu'on aurait pu se dispenser de calculer, mais il faut bien illustrer la méthode), l'ensemble S se réduit à :

S = { DF01 : {A} → {B}
 DF02 : {A} → {C}
 DF03 : {B} → {C}
 DF06 : {A,C} → {D} }

A noter que, grâce à DF03, DF05 était aussi réductible à {B} → {C}. En effet, partant de DF03, par augmentation on produit {A,B} → {A,C}, puis par décomposition on produit {A,B} → {C}, c'est-à-dire DF05.

b) Cas de DF06 : **{A,C} → {D}**.

DF06 est-elle réductible à {A} → {D} ? Considérons le sous-ensemble I de S, dans lequel on remplace DF06 par DF06a : {A} → {D}.

Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{A\}^+$ de {A} par rapport à S est égale à la fermeture $\{A\}^+$ de {A} par rapport à I. S et I sont respectivement représentés par :

S = { DF01 : {A} → {B}	I = { DF01 : {A} → {B}
DF02 : {A} → {C}	DF02 : {A} → {C}
DF03 : {B} → {C}	DF03 : {B} → {C}
DF06 : {A,C} → {D} }	DF06a : {A} → {D} }

DF06a ayant pour déterminant {A}, les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{A\}^+$ de {A} par rapport à S est égale à la fermeture $\{A\}^+$ de {A} par rapport à I.

Au moyen de l'algorithme du seau, calculons $\{A\}^+$ par rapport à S :

$\{A\}^+ = \{A\}$, puis {A,B} du fait de DF01, puis {A,B,C} du fait de DF02, puis {A,B,C,D} du fait de DF06.

De même, calculons $\{A\}^+$ par rapport à I :

$\{A\}^+ = \{A\}$, puis {A,B} du fait de DF01, puis {A,B,C} du fait de DF02, puis {A,B,C,D} du fait de DF06a.

$\{A\}^+$ par rapport à S est égal à $\{A\}^+$ par rapport à I : Les deux ensembles S et I sont équivalents par rapport à leurs fermetures et I peut donc remplacer S qui devient :

S = { DF01 : {A} → {B}
 DF02 : {A} → {C}
 DF03 : {B} → {C}
 DF06a : {A} → {D} }

Pour sa part, DF06 n'est pas réductible à {C} → {D}, car les fermetures $\{C\}^+$ par rapport à I : {C,D} et $\{C\}^+$ par rapport à S : {C} ne sont pas égales.

Troisième étape

On cherche quelles DF peuvent disparaître, sans conséquence sur S.

Commençons par tenter de supprimer la 1^{re} DF qui se présente, à savoir DF01 pour réduire S à $I = S - \{DF01\}$.

Concernant S, DF01 a pour déterminant $\{A\}$. En utilisant l'algorithme du seau, calculons la fermeture $\{A\}^+$ par rapport à S. Au départ, $\{A\}^+ = \{A\}$. Ensuite, $\{A\}^+ = \{A,B\}$ du fait de DF01 (présente dans S), puis $\{A,B,C\}$ du fait de DF02, puis $\{A,B,C,D\}$ du fait de DF06a.

Calculons la fermeture $\{A\}^+$ par rapport à I. Au départ, $\{A\}^+ = \{A\}$. Ensuite, $\{A\}^+ = \{A,C\}$ du fait de DF02, puis $\{A,C,D\}$ du fait de DF06a, puis plus rien ne se passe puisque DF01 est absente de I.

$\{A\}^+$ par rapport à S diffère de $\{A\}^+$ par rapport à I, donc DF01 ne peut pas être éliminée de S.

Tentons de supprimer la DF suivante, DF02 pour réduire S à $I = S - \{DF02\}$.

Concernant S, DF02 a pour déterminant $\{A\}$. En utilisant l'algorithme du seau, calculons la fermeture $\{A\}^+$ par rapport à S. Au départ, $\{A\}^+ = \{A\}$. Ensuite, $\{A\}^+ = \{A,B\}$ du fait de DF01, puis $\{A,B,C\}$ du fait de DF02 (présente dans S), puis $\{A,B,C,D\}$ du fait de DF06a.

Calculons la fermeture $\{A\}^+$ par rapport à I. Au départ, $\{A\}^+ = \{A\}$. Ensuite, $\{A\}^+ = \{A,B\}$ du fait de DF01, puis $\{A,B,C\}$ du fait de DF03, puis $\{A,B,C,D\}$ du fait de DF06a.

$\{A\}^+$ par rapport à S égale $\{A\}^+$ par rapport à I, donc DF02 peut être éliminée de S qui devient :

S = { DF01 : $\{A\} \rightarrow \{B\}$
 DF03 : $\{B\} \rightarrow \{C\}$
 DF06a : $\{A\} \rightarrow \{D\}$ }

L'ensemble de DF est maintenant irréductible. En effet, on a déjà montré que DF01 ne pouvait pas être supprimée. Sans même calculer, on sait que DF03 ne peut pas l'être non plus, car l'attribut C fait partie de sa fermeture $\{B\}^+$ mais est absent de DF01 et de DF06a. A son tour, DF06a ne peut pas non plus être supprimée, car l'attribut D fait partie de sa fermeture $\{A\}^+$ mais est absent de DF01 et de DF03.

Le travail est terminé et, de façon mécanique, on a réduit l'ensemble S de DF de la relvar R de façon conséquente.



Plutôt qu'utiliser l'algorithme du seau on aurait pu en passer par les axiomes d'Armstrong et les règles qui en sont inférées, mais on conviendra que la tâche devient d'autant plus ardue que l'on manque d'entraînement. Ainsi, revenons-en à l'ensemble des DF obtenu après la première étape :

DF01 : $\{A\} \rightarrow \{B\}$
 DF02 : $\{A\} \rightarrow \{C\}$
 DF03 : $\{B\} \rightarrow \{C\}$
 DF05 : $\{A,B\} \rightarrow \{C\}$
 DF06 : $\{A,C\} \rightarrow \{D\}$

Concernant la 2e étape :

a) Montrer que la DF $\{A,B\} \rightarrow \{C\}$ peut être inférée

- 1 $\{A\} \rightarrow \{C\}$ (donné)
- 2 $\{A,B\} \rightarrow \{B,C\}$ (1, augmentation)
- 3 $\{A,B\} \rightarrow \{C\}$ (2, décomposition)

b) Montrer que la DF $\{A\} \rightarrow \{D\}$ peut être inférée

- 1 $\{A\} \rightarrow \{C\}$ (donné)
- 2 $\{A\} \rightarrow \{A,C\}$ (1, augmentation)
- 3 $\{A,C\} \rightarrow \{D\}$ (donné)
- 4 $\{A\} \rightarrow \{D\}$ (2, 3, transitivité)

Concernant la 3e étape :

Montrer que la DF $\{A\} \rightarrow \{C\}$ peut être inférée

- 1 $\{A\} \rightarrow \{B\}$ (donné)
- 2 $\{B\} \rightarrow \{C\}$ (donné)
- 3 $\{A\} \rightarrow \{C\}$ (1, 2, transitivité)

A chacun sa stratégie : se passer d'un rouleau compresseur rustique et d'un seau à dépendants, préférer mettre en œuvre les axiomes d'Armstrong est certes possible et c'est plus élégant. En tout cas, l'essentiel est que le travail soit mené à bien.

E.6.3. Pluralité des ensembles irréductibles pour une relvar

Pour une relvar et un ensemble donné de DF, on peut avoir plus d'un ensemble irréductible. Reprenons l'exemple proposé par Ullman (cf. annexe E.5.1). L'ensemble S des DF pour la relvar R {A,B,C,D,E,G} est le suivant :

S = { DF01 : {A,B} → {C}	DF05 : {D} → {E,G}
DF02 : {C} → {A}	DF06 : {B,E} → {C}
DF03 : {B,C} → {D}	DF07 : {C,G} → {B,D}
DF04 : {A,C,D} → {B}	DF08 : {C,E} → {A,G} }

Cherchons à réduire S.

Première étape

On transforme S, de telle sorte que le dépendant de chaque DF soit singleton (application de la règle de décomposition) :

S = { DF01 : {A,B} → {C}	DF07 : {B,E} → {C}
DF02 : {C} → {A}	DF08 : {C,G} → {B}
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}
DF04 : {A,C,D} → {B}	DF10 : {C,E} → {A}
DF05 : {D} → {E}	DF11 : {C,E} → {G} }
DF06 : {D} → {G}	

Deuxième étape

Quand c'est possible, on réduit le déterminant de chaque DF :

Sont à considérer les DF dont le déterminant comporte plus d'un attribut, à savoir DF01, DF03, DF04, DF07, DF08, DF09, DF10, DF11.

a) Cas de DF01 : {A,B} → {C}.

DF01 est-elle réductible à {A} → {C} ? Considérons le sous-ensemble I de S, où l'on remplace DF01 par DF01a, {A} → {C} :

S = { DF01 : {A,B} → {C}	DF07 : {B,E} → {C}
DF02 : {C} → {A}	DF08 : {C,G} → {B}
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}
DF04 : {A,C,D} → {B}	DF10 : {C,E} → {A}
DF05 : {D} → {E}	DF11 : {C,E} → {G} }
DF06 : {D} → {G}	
I = { DF01a : {A} → {C}	DF07 : {B,E} → {C}
DF02 : {C} → {A}	DF08 : {C,G} → {B}
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}
DF04 : {A,C,D} → {B}	DF10 : {C,E} → {A}
DF05 : {D} → {E}	DF11 : {C,E} → {G} }
DF06 : {D} → {G}	

DF01a a pour déterminant {A}. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{A\}^+$ de {A} par rapport à S est égale à la fermeture $\{A\}^+$ de {A} par rapport à I.

Au moyen de l'algorithme du seau, calculons $\{A\}^+$ par rapport à S : $\{A\}^+ = \{A\}$, puis on en reste là.

Calculons $\{A\}^+$ par rapport à I : $\{A\}^+ = \{A,C\}$ du fait de DF01a.

$\{A\}^+$ par rapport à S n'est pas égal à $\{A\}^+$ par rapport à I : S ne peut pas être remplacé par I.

DF01 est-elle réductible à $\{B\} \rightarrow \{C\}$? Considérons le sous-ensemble I de S, où l'on remplace DF01 par DF01b, $\{B\} \rightarrow \{C\}$:

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	
I = { DF01b : $\{B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	

DF01b a pour déterminant $\{B\}$. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{B\}^+$ de $\{B\}$ par rapport à S est égale à la fermeture $\{B\}^+$ de $\{B\}$ par rapport à I.

Au moyen de l'algorithme du seau, calculons $\{B\}^+$ par rapport à S :

$\{B\}^+ = \{B\}$, puis on en reste là.

Calculons $\{B\}^+$ par rapport à I :

$\{B\}^+ = \{B,C\}$ du fait de DF01a, puis $\{B,C,D\}$ du fait de DF03, etc.

$\{B\}^+$ par rapport à S n'est pas égal à $\{B\}^+$ par rapport à I qui ne peut donc pas remplacer S.

b) Cas de DF03 : $\{B,C\} \rightarrow \{D\}$.

DF03 est-elle réductible à $\{B\} \rightarrow \{D\}$? Considérons le sous-ensemble I de S, où l'on remplace DF03 par DF03a, $\{B\} \rightarrow \{D\}$:

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	
I = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03a : $\{B\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	

DF03a a pour déterminant $\{B\}$. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{B\}^+$ de $\{B\}$ par rapport à S est égale à la fermeture $\{B\}^+$ de $\{B\}$ par rapport à I.

Au moyen de l'algorithme du seau, calculons $\{B\}^+$ par rapport à S : $\{B\}^+ = \{B\}$, puis on en reste là.

Calculons $\{B\}^+$ par rapport à I : $\{B\}^+ = \{B,D\}$ du fait de DF03a, puis $\{B,D,E\}$ du fait de DF05, etc.

$\{B\}^+$ par rapport à S n'est pas égal à $\{B\}^+$ par rapport à I : S ne peut pas être remplacé par I.

DF03 est-elle réductible à $\{C\} \rightarrow \{D\}$? Considérons le sous-ensemble I de S, où l'on remplace DF03 par DF03b, $\{C\} \rightarrow \{D\}$:

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	
I = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03b : $\{C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	

DF03b a pour déterminant $\{C\}$. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{C\}^+$ de $\{C\}$ par rapport à S est égale à la fermeture $\{C\}^+$ de $\{C\}$ par rapport à I.

Calculons $\{C\}^+$ par rapport à S : $\{C\}^+ = \{A,C\}$ du fait de DF02, puis on en reste là.

Calculons $\{C\}^+$ par rapport à I : $\{C\}^+ = \{C,D\}$ du fait de DF03b, puis $\{A,C,D\}$ du fait de DF02, etc.

$\{C\}^+$ par rapport à S n'est pas égal à $\{C\}^+$ par rapport à I qui ne peut donc pas remplacer S.

c) Cas de DF04 : $\{A,C,D\} \rightarrow \{B\}$.

DF04 est-elle réductible à $\{A,C\} \rightarrow \{B\}$? Considérons le sous-ensemble I de S, où l'on remplace DF04 par DF04a, $\{A,C\} \rightarrow \{B\}$:

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	
I = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04a : $\{A,C\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	

DF04a a pour déterminant $\{A,C\}$. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{A,C\}^+$ de $\{A,C\}$ par rapport à S est égale à la fermeture $\{A,C\}^+$ de $\{A,C\}$ par rapport à I.

Calculons $\{A,C\}^+$ par rapport à S : $\{A,C\}^+ = \{A,C\}$, puis on en reste là.

Calculons $\{A,C\}^+$ par rapport à I : $\{A,C\}^+ = \{A,B,C\}$ du fait de DF04a, puis $\{A,B,C,D\}$ du fait de DF03, etc.

$\{A,C\}^+$ par rapport à S n'est pas égal à $\{A,C\}^+$ par rapport à I qui ne peut donc pas remplacer S.

DF04 est-elle réductible à $\{A,D\} \rightarrow \{B\}$? Considérons le sous-ensemble I de S, où l'on remplace DF04 par DF04b, $\{A,D\} \rightarrow \{B\}$:

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	
I = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04b : $\{A,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	

DF04b a pour déterminant $\{A,D\}$. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{A,D\}^+$ de $\{A,D\}$ par rapport à S est égale à la fermeture $\{A,D\}^+$ de $\{A,D\}$ par rapport à I.

Calculons $\{A,D\}^+$ par rapport à S : $\{A,D\}^+ = \{A,D\}$, puis $\{A,D,E,G\}$ du fait de DF05 et DF06.

Calculons $\{A,D\}^+$ par rapport à I : $\{A,D\}^+ = \{A,B,D\}$ du fait de DF04b, puis $\{A,B,C,D\}$ du fait de DF01, etc.

$\{A,D\}^+$ par rapport à S n'est pas égal à $\{A,D\}^+$ par rapport à I qui ne peut donc pas remplacer S.

DF04 est-elle réductible à $\{C,D\} \rightarrow \{B\}$? Considérons le sous-ensemble I de S, où l'on remplace DF04 par DF04c, $\{C,D\} \rightarrow \{B\}$:

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{A,C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	
I = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04c : $\{C,D\} \rightarrow \{B\}$	DF10 $\{C,E\} \rightarrow \{A\}$
DF05 : $\{D\} \rightarrow \{E\}$	DF11 $\{C,E\} \rightarrow \{G\}$ }
DF06 : $\{D\} \rightarrow \{G\}$	

DF04c a pour déterminant $\{C,D\}$. Les deux ensembles S et I sont équivalents si et seulement si la fermeture $\{C,D\}^+$ de $\{C,D\}$ par rapport à S est égale à la fermeture $\{C,D\}^+$ de $\{C,D\}$ par rapport à I.

Calculons $\{C,D\}^+$ par rapport à S : $\{C,D\}^+ = \{C,D\}$, puis $\{A,C,D\}$ du fait de DF02, puis $\{A,C,D,E,G\}$ du fait de DF05 et DF06, puis $\{A,B,C,D,E,G\}$ du fait de DF08 (on notera que $\{C,D\}$ représente une clé candidate pour R).

Calculons $\{C,D\}^+$ par rapport à I : $\{C,D\}^+ = \{B,C,D\}$ du fait de DF04c, puis $\{A,B,C,D\}$ du fait de DF01, puis $\{A,B,C,D,E,G\}$ du fait de DF05 et DF06.

$\{C,D\}^+$ par rapport à S est égal à $\{C,D\}^+$ par rapport à I : S peut être remplacé par I et la dépendance fonctionnelle DF04 par DF04c = $\{C,D\} \rightarrow \{B\}$.



Mais le déterminant de DF04c comporte deux attributs et il faut donc vérifier si, à son tour, cette DF peut se simplifier en $\{C\} \rightarrow \{B\}$ ou $\{D\} \rightarrow \{B\}$. Concernant $\{C\} \rightarrow \{B\}$, la fermeture $\{C\}^+$ par rapport à S est égale à $\{A,C\}$ et la fermeture $\{C\}^+$ par rapport à I est égale à $\{A,B,C,D,E,G\}$. Ces fermetures ne sont pas égales, donc la simplification n'est pas possible.

Concernant $\{D\} \rightarrow \{B\}$, la fermeture $\{D\}^+$ par rapport à S est égale à $\{D,E,G\}$ et la fermeture $\{C\}^+$ par rapport à I est égale à $\{A,B,C,D,E,G\}$: Ces fermetures n'étant pas égales, la simplification n'est pas possible.

- d) Cas de DF07 à DF11 : Toujours avec la même méthode, on montre ensuite que ces dépendances fonctionnelles sont irréductibles, à l'exception de DF10 pour laquelle $\{C\}^+$ par rapport à S est égale à $\{C\}^+$ par rapport à I. DF10 est réductible à $\{C\} \rightarrow \{A\}$ c'est-à-dire à DF02 : DF10 disparaît.

En fin d'étape 2, l'ensemble S à été réduit à :

S = { DF01 : $\{A,B\} \rightarrow \{C\}$	DF07 : $\{B,E\} \rightarrow \{C\}$
DF02 : $\{C\} \rightarrow \{A\}$	DF08 : $\{C,G\} \rightarrow \{B\}$
DF03 : $\{B,C\} \rightarrow \{D\}$	DF09 : $\{C,G\} \rightarrow \{D\}$
DF04 : $\{C,D\} \rightarrow \{B\}$	DF11 : $\{C,E\} \rightarrow \{G\}$
DF05 : $\{D\} \rightarrow \{E\}$	
DF06 : $\{D\} \rightarrow \{G\}$	

Troisième étape

On cherche quelles DF peuvent disparaître, sans conséquence pour S.

Commençons par tenter de supprimer la 1re DF qui se présente, à savoir DF01 pour réduire S à $I = S - \{DF01\}$.

Concernant S, le déterminant $\{A,B\}$ de DF01 a d'abord pour fermeture $\{A,B\}^+ = \{A,B,C\}$ — du fait de DF01 elle-même —, puis $\{A,B,C,D\}$ du fait de DF03, puis $\{A,B,C,D,E,G\}$ du fait de DF05 et DF06.

Concernant I, on cherche donc à retrouver $\{A,B\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF01. Au départ, $\{A,B\}^+ = \{A,B\}$, mais rien d'autre ne peut tomber dans le seau :

$\{A,B\}^+$ par rapport à S n'est pas égal à $\{A,B\}^+$ par rapport à I, donc DF01 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF02 pour réduire S à $I = S - \{DF02\}$.

Concernant S, le déterminant $\{C\}$ de DF02 a d'abord pour fermeture $\{C\}^+ = \{AC\}$ — du fait de DF02 elle-même — mais rien d'autre ne peut tomber dans le seau.

Concernant I, on cherche à retrouver $\{C\}^+ = \{AC\}$ sans utiliser DF02. Au départ, $\{C\}^+ = \{C\}$, mais rien d'autre ne peut tomber dans le seau :

$\{C\}^+$ par rapport à S n'est pas égal à $\{C\}^+$ par rapport à I, donc DF02 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF03 pour réduire S à $I = S - \{DF03\}$.

Concernant S, le déterminant $\{B,C\}$ de DF03 a d'abord pour fermeture $\{B,C\}^+ = \{B,C,D\}$ — du fait de DF03 elle-même —, puis $\{A,B,C,D\}$ du fait de DF02, puis $\{A,B,C,D,E,G\}$ du fait de DF05 et DF06.

Concernant I, on cherche à retrouver $\{B,C\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF03. Au départ, $\{B,C\}^+ = \{B,C\}$ puis $\{A,B,C\}$ du fait de DF02, mais sans DF03 rien d'autre ne peut tomber dans le seau :

$\{B,C\}^+$ par rapport à S n'est pas égal à $\{B,C\}^+$ par rapport à I, donc DF03 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF04 pour réduire S à $I = S - \{DF04\}$.

Concernant S, le déterminant $\{C,D\}$ de DF04 a d'abord pour fermeture $\{C,D\}^+ = \{B,C,D\}$ — du fait de DF04 elle-même —, puis $\{A,B,C,D\}$ du fait de DF02, puis $\{A,B,C,D,E,G\}$ du fait de DF05 et DF06.

Concernant I, on cherche à retrouver $\{C,D\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF04. Au départ, $\{C,D\}^+ = \{C,D\}$ puis $\{A,C,D\}$ du fait de DF02, puis $\{A,C,D,E,G\}$ du fait de DF05 et DF06, puis $\{A,B,C,D,E,G\}$ du fait de DF08 :

$\{C,D\}^+$ par rapport à S est égal à $\{C,D\}^+$ par rapport à I, donc DF04 peut disparaître (puisque I comporte la DF $\{C,D\} \rightarrow \{A,B,C,D,E,G\}$, en vertu de la règle de décomposition on infère la DF $\{C,D\} \rightarrow \{B\}$).

Dans ces conditions, S devient :

S = { DF01 : {A,B} → {C}	DF07 {B,E} → {C}
DF02 : {C} → {A}	DF08 {C,G} → {B}
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}
DF05 : {D} → {E}	DF11 {C,E} → {G} }
DF06 : {D} → {G}	

Tentons de supprimer la DF suivante, à savoir DF05 pour réduire S à $I = S - \{DF05\}$.

Concernant S, le déterminant {D} de DF05 a d'abord pour fermeture $\{D\}^+ = \{D,E\}$ — du fait de DF05 elle-même —, puis {D,E,G} du fait de DF06, mais rien d'autre ne peut tomber dans le seau.

Concernant I, on cherche à retrouver $\{D\}^+ = \{D,E,G\}$ sans utiliser DF05. Au départ, $\{D\}^+ = \{D\}$ puis {D,G} du fait de DF06, mais rien d'autre ne peut tomber dans le seau :

$\{D\}^+$ par rapport à S n'est pas égal à $\{D\}^+$ par rapport à I, donc DF05 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF06 pour réduire S à $I = S - \{DF06\}$.

Concernant S, le déterminant {D} de DF06 a d'abord pour fermeture $\{D\}^+ = \{D,G\}$ — du fait de DF06 elle-même —, puis {D,E,G} du fait de DF05, mais rien d'autre ne peut tomber dans le seau.

Concernant I, on cherche à retrouver $\{D\}^+ = \{D,E,G\}$ sans utiliser DF06. Au départ, $\{D\}^+ = \{D\}$ puis {D,E} du fait de DF05, mais rien d'autre ne peut tomber dans le seau :

$\{D\}^+$ par rapport à S n'est pas égal à $\{D\}^+$ par rapport à I, donc DF06 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF07 pour réduire S à $I = S - \{DF07\}$.

Concernant S, le déterminant {B,E} de DF07 a d'abord pour fermeture $\{B,E\}^+ = \{B,C,E\}$ — du fait de DF07 elle-même —, puis {A,B,C,E} du fait de DF02, puis {A,B,C,D,E} du fait de DF03, puis {A,B,C,D,E,G} du fait de DF06.

Concernant I, on cherche à retrouver $\{B,E\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF07. Au départ, $\{B,E\}^+ = \{B,E\}$, mais rien d'autre ne peut tomber dans le seau :

$\{B,E\}^+$ par rapport à S n'est pas égal à $\{B,E\}^+$ par rapport à I, donc DF07 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF08 pour réduire S à $I = S - \{DF08\}$.

Concernant S, le déterminant {C,G} de DF08 a d'abord pour fermeture $\{C,G\}^+ = \{B,C,G\}$ — du fait de DF08 elle-même —, puis {A,B,C,G} du fait de DF02, puis {A,B,C,D,G} du fait de DF03, puis {A,B,C,D,E,G} du fait de DF06.

Concernant I, on cherche à retrouver $\{C,G\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF08. Au départ, $\{C,G\}^+ = \{C,G\}$ puis {A,C,G} du fait de DF02, puis {A,C,D,G} du fait de DF09, puis {A,C,D,E,G} du fait de DF05, mais rien d'autre ne peut tomber dans le seau (sinon, il faudrait ressusciter DF04) :

$\{C,G\}^+$ par rapport à S n'est pas égal à $\{C,G\}^+$ par rapport à I, donc DF08 doit continuer à faire partie de S.

Tentons de supprimer la DF suivante, à savoir DF09 pour réduire S à $I = S - \{DF09\}$.

Concernant S, le déterminant {C,G} de DF09 a d'abord pour fermeture $\{C,G\}^+ = \{C,D,G\}$ — du fait de DF09 elle-même —, puis {A,C,D,G} du fait de DF02, puis {A,C,D,E,G} du fait de DF05, puis {A,B,C,D,E,G} du fait de DF08.

Concernant I, on cherche à retrouver $\{C,G\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF09. Au départ, $\{C,G\}^+ = \{C,G\}$ puis {A,C,G} du fait de DF02, puis {A,B,C,G} du fait de DF08, puis {A,B,C,D,G} du fait de DF03, puis {A,B,C,D,E,G} du fait de DF05 :

$\{C,G\}^+$ par rapport à S est égal à $\{C,G\}^+$ par rapport à I, donc DF09 peut disparaître (puisque I comporte la DF $\{C,G\} \rightarrow \{A,B,C,D,E,G\}$, en vertu de la règle de décomposition on infère la DF $\{C,G\} \rightarrow \{D\}$).

Dans ces conditions, S devient :

S = { DF01 : {A,B} → {C}	DF07 {B,E} → {C}
DF02 : {C} → {A}	DF08 {C,G} → {B}
DF03 : {B,C} → {D}	DF11 {C,E} → {G} }
DF05 : {D} → {E}	
DF06 : {D} → {G}	

Tentons de supprimer la DF suivante, à savoir DF11 pour réduire S à $I = S - \{DF11\}$.

Concernant S, le déterminant {C,E} de DF11 a d'abord pour fermeture $\{C,E\}^+ = \{C,E,G\}$ — du fait de DF11 elle-même —, puis {A,C,E,G} du fait de DF02, puis {A,B,C,E,G} du fait de DF08, puis {A,B,C,D,E,G} du fait de DF03.

Concernant I, on cherche à retrouver $\{C,E\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF11. Au départ, $\{C,E\}^+ = \{C,E\}$ puis {A,C,E} du fait de DF02, mais rien d'autre ne peut tomber dans le seau :

$\{C,E\}^+$ par rapport à S n'est pas égal à $\{C,E\}^+$ par rapport à I, donc DF11 doit continuer à faire partie de S.

Au final

Il n'y a plus d'autre DF à chercher à éliminer : l'ensemble S proposé pour la 3e étape était réductible du fait de la présence de DF redondantes : DF04 = {A,C,D} → {B} (réduite d'abord à {C,D} → {B} ce qui du reste n'était pas indispensable) et DF09 = {C,G} → {D}. S est maintenant irréductible.

S =	DF01 : {A,B} → {C}	DF06 : {D} → {G}	1er ensemble irréductible
	DF02 : {C} → {A}	DF07 {B,E} → {C}	
	DF03 : {B,C} → {D}	DF08 {C,G} → {B}	
	DF05 : {D} → {E}	DF11 {C,E} → {G}	



Rappelons en passant que {A,B}, {B,C}, {B,D}, {B,E}, {C,D}, {C,E}, {C,G} sont les clés candidates de la relvar R (Cf. paragraphe E.5.2).

Par ailleurs, les dépendants singletons ayant même déterminant peuvent maintenant être regroupés : {D} → {E,G}.

Recherche d'un autre ensemble irréductible

Un ensemble irréductible n'est pas nécessairement unique et, dans cet exemple, tout peut dépendre de l'ordre dans lequel on élimine les redondances. Reprenons la 3e étape à son début :

S = { DF01 : {A,B} → {C}	DF07 {B,E} → {C}
DF02 : {C} → {A}	DF08 {C,G} → {B}
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}
DF04 : {C,D} → {B}	DF11 {C,E} → {G} }
DF05 : {D} → {E}	
DF06 : {D} → {G}	

Et commençons, par chercher à éliminer DF08 : {C,G} → {B} pour réduire S à $I = S - \{DF08\}$. :

Concernant S, le déterminant {C,G} de DF08 a d'abord pour fermeture $\{C,G\}^+ = \{B,C,G\}$ — du fait de DF08 elle-même —, puis {A,B,C,G} du fait de DF02, puis {A,B,C,D,G} du fait de DF03, puis {A,B,C,D,E,G} du fait de DF05.

Concernant I, on cherche à retrouver $\{C,G\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF08. Au départ, $\{C,G\}^+ = \{C,G\}$, puis {A,C,G} du fait de DF02, puis {A,C,D,G} du fait de DF09, puis {A,B,C,D,G} du fait de DF04 (ce qui n'avait pas été possible lors du tour précédent, DF04 ayant d'abord été éliminée de S) , puis {A,B,C,D,E,G} du fait de DF05 :

$\{C,G\}^+$ par rapport à S est égal à $\{C,G\}^+$ par rapport à I, donc DF08 peut disparaître (puisque I comporte la DF {C,G} → {A,B,C,D,E,G}, en vertu de la règle de décomposition on infère la DF {C,G} → {B}).

Dans ces conditions, S devient :

S = { DF01 : {A,B} → {C}	DF06 : {D} → {G}
DF02 : {C} → {A}	DF07 : {B,E} → {C}
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}
DF04 : {C,D} → {B}	DF11 : {C,E} → {G} }
DF05 : {D} → {E}	

On sait désormais que le nouvel ensemble est nécessairement différent du 1er ensemble irréductible, puisque que l'on vient d'éliminer DF08 qui fait partie du 1er.

On repart pour un tour complet, et l'on montre à nouveau que DF01, DF02, DF03, DF05, DF06, DF07 et DF11 ne sont pas redondantes et ne peuvent pas être éliminées. Mais on montre aussi que DF04 et DF09, qui ne font pas partie du 1er ensemble irréductible, ne pourront pas cette fois-ci être éliminées.

Cas de DF04 :

Concernant S, le déterminant {C,D} de DF04 a d'abord pour fermeture $\{C,D\}^+ = \{B,C,D\}$ — du fait de DF04 elle-même —, puis {A,B,C,D} du fait de DF02 puis {A,B,C,D,E,G} du fait de DF05 et DF06.

Concernant I, on cherche à retrouver $\{C,D\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF04. Au départ, $\{C,D\}^+ = \{C,D\}$ puis {A,C,D} du fait de DF02, puis {A,C,D,E,G} du fait de DF05 et DF06, mais rien d'autre ne peut tomber dans le seau (outre la tentative d'élimination de DF04, comme on vient de procéder à celle de DF08, l'attribut B ne peut plus être capturé) :

$\{C,D\}^+$ par rapport à S n'est pas égal à $\{C,D\}^+$ par rapport à I, donc DF04 doit continuer à faire partie de S.

Cas de DF09 :

Concernant S, le déterminant {C,G} de DF09 a d'abord pour fermeture $\{C,G\}^+ = \{C,D,G\}$ — du fait de DF09 elle-même —, puis {A,C,D,G} du fait de DF02 puis {A,B,C,D,G} du fait de DF04, puis {A,B,C,D,E,G} du fait de DF05.

Concernant I, on cherche à retrouver $\{C,G\}^+ = \{A,B,C,D,E,G\}$ sans utiliser DF09. Au départ, $\{C,G\}^+ = \{C,G\}$ puis {A,C,G} du fait de DF02, mais rien d'autre ne peut tomber dans le seau (outre la tentative d'élimination de DF09, comme on vient de procéder à celle de DF08, l'attribut B ne peut plus être capturé) :

$\{C,G\}^+$ par rapport à S n'est pas égal à $\{C,G\}^+$ par rapport à I, donc DF09 doit continuer à faire partie de S.

Ainsi, on dispose d'un deuxième ensemble irréductible :

S = { DF01 : {A,B} → {C}	DF06 : {D} → {G}	2e ensemble irréductible
DF02 : {C} → {A}	DF07 : {B,E} → {C}	
DF03 : {B,C} → {D}	DF09 : {C,G} → {D}	
DF04 : {C,D} → {B}	DF11 : {C,E} → {G} }	
DF05 : {D} → {E}		

On a détaillé le plus possible une technique de recherche des ensembles irréductibles. On laissera aux plus astucieux ou courageux le soin de vérifier s'il existe d'autres ensembles de ce type. On peut aussi, à l'aide de son langage préféré, développer un programme idoine pour traiter ensuite le problème en un clic de souris. Quoi qu'il en soit, le but de la manœuvre est bien de chercher à réduire S, afin d'éliminer les DF redondantes et réduire ainsi la charge de travail du SGBD quant aux contrôles qu'on lui demandera d'assumer.

E.7. Décompositions sans perte

E.7.1. Préservation du contenu de la base de données

La stricte application du théorème de Heath permet de préserver le contenu de la base de données quand on décompose une relvar. Voici une extension de ce théorème (cf. [Ullman 1982], « Theorem 7.5 ») :

Si $\rho = (R_1, R_2)$ est une décomposition de R , et S est un ensemble de dépendances fonctionnelles, alors ρ préserve le contenu de la base de données par rapport à S si et seulement si on vérifie les dépendances fonctionnelles $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ ou $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$. Ces dépendances n'appartiennent pas nécessairement à S , il suffit qu'elles appartiennent à S^+ .

Considérons par exemple la relvar suivante : $R \{A,B,C\}$ et l'ensemble de DF associé : $S = \{\{A\} \rightarrow \{B\}\}$.

Si on décompose R en $R_1 \{A,B\}$ et $R_2 \{A,C\}$, le contenu de la base de données est préservé, car d'une part $\{A,B\} \cap \{A,C\} = \{A\}$ et d'autre part $\{A,B\} - \{A,C\} = \{B\}$, donc on vérifie bien $\{A\} \rightarrow \{B\}$.

En revanche, si l'on décompose R en $R_1 \{A,B\}$ et $R_2 \{B,C\}$, alors $\{A,B\} \cap \{B,C\} = \{B\}$, $\{A,B\} - \{B,C\} = \{A\}$ et $\{B,C\} - \{A,B\} = \{C\}$: $(R_1 \cap R_2)$ ne détermine fonctionnellement ni $(R_1 - R_2)$ ni $(R_2 - R_1)$, le contenu de la base de données n'est donc pas préservé. On peut du reste le vérifier à l'aide d'un exemple :

Soit $r = \{\{a_1,b_1,c_1\}, \{a_2,b_1,c_2\}\}$ une valeur prise par R . Les projections $R_1 = \pi_{AB}(r)$ et $R_2 = \pi_{BC}(r)$ sont respectivement égales à $\{\{a_1,b_1\}, \{a_2,b_1\}\}$ et $\{\{b_1,c_1\}, \{b_1,c_2\}\}$ tandis que leur jointure naturelle est égale à $\{\{a_1,b_1,c_1\}, \{a_1,b_1,c_2\}, \{a_2,b_1,c_1\}, \{a_2,b_1,c_2\}\}$, laquelle n'est pas égale à r .

Maintenant, peut-on produire des décompositions préservant le contenu de la base de données sans appliquer le théorème de Heath ? Pour répondre à cette question, intéressons-nous à un exemple proposé par Jeff Ullman (cf. [Ullman 1982], « Algorithm 7.2, Testing Lossless Joins ») :

Soit la relvar $R \{A,B,C,D,E\}$. L'ensemble S de DF associé est le suivant :

$$S = \{\{A\} \rightarrow \{C\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{D\}, \{D,E\} \rightarrow \{C\}, \{C,E\} \rightarrow \{A\}\}$$

Et la décomposition proposée est la suivante :

$$\rho = (\{A,D\}, \{A,B\}, \{C,D,E\}, \{A,E\}, \{B,E\})$$

La méthode proposée de vérification de la préservation du contenu de la base de données est la suivante : on met en oeuvre un tableau dans lequel chaque colonne correspond à un attribut de R et chaque ligne correspond à un élément de ρ . Soit A_j le nom de l'attribut figurant dans la colonne j et R_i le nom de l'élément figurant dans la ligne i . Si A_j appartient à R_i alors à l'intersection de la ligne i et de la colonne j on fait figurer le symbole a_j . Si A_j n'appartient pas à R_i alors à cette intersection on fait figurer le symbole b_{ij} :

	A	B	C	D	E
$\{A,D\}$	a_1	b_{12}	b_{13}	a_4	b_{15}
$\{A,B\}$	a_1	a_2	b_{23}	b_{24}	b_{25}
$\{C,D,E\}$	b_{31}	b_{32}	a_3	a_4	a_5
$\{A,E\}$	a_1	b_{42}	b_{43}	b_{44}	a_5
$\{B,E\}$	b_{51}	a_2	b_{53}	b_{54}	a_5

A partir de là, on applique un algorithme permettant de faire évoluer le contenu du tableau en fonction des DF figurant dans S , le but étant d'arriver à mettre en évidence une ligne qui ne contienne que des symboles a_j auquel cas — et seulement dans ce cas — on pourra affirmer que la décomposition préserve le contenu de la base de données (démonstration : voir le théorème 7.4 figurant dans [Ullman 1982]).

Utilisons la 1re DF appartenant à S, $\{A\} \rightarrow \{C\}$. Dans le tableau ci-dessus, l'attribut A (1re colonne) appartient au déterminant de la DF et comporte le même symbole a_1 aux lignes 1, 2 et 4. Selon l'algorithme, parce que l'attribut C (3e colonne du tableau) appartient au dépendant de la DF, pour ces trois lignes on peut se limiter à un seul et même symbole, qui peut être arbitrairement b_{13} , b_{23} ou b_{43} . Retenons b_{13} auquel cas le tableau devient :

	A	B	C	D	E
{A,D}	a_1	b_{12}	b_{13}	a_4	b_{15}
{A,B}	a_1	a_2	b_{13}	b_{24}	b_{25}
{C,D,E}	b_{31}	b_{32}	a_3	a_4	a_5
{A,E}	a_1	b_{42}	b_{13}	b_{44}	a_5
{B,E}	b_{51}	a_2	b_{53}	b_{54}	a_5

On utilise la même technique avec la DF suivante, $\{B\} \rightarrow \{C\}$. Du fait que l'attribut B comporte deux fois le symbole a_2 (lignes 2 et 5), le symbole b_{13} (ligne 2) peut remplacer le symbole b_{53} (ligne 5) :

	A	B	C	D	E
{A,D}	a_1	b_{12}	b_{13}	a_4	b_{15}
{A,B}	a_1	a_2	b_{13}	b_{24}	b_{25}
{C,D,E}	b_{31}	b_{32}	a_3	a_4	a_5
{A,E}	a_1	b_{42}	b_{13}	b_{44}	a_5
{B,E}	b_{51}	a_2	b_{13}	b_{54}	a_5

Même chose avec la DF suivante, $\{C\} \rightarrow \{D\}$. Du fait que l'attribut C comporte quatre fois le symbole b_{13} (lignes 1, 2, 4 et 5), le symbole a_4 (ligne 1) peut remplacer les symboles b_{24} (ligne 2), b_{44} (ligne 4) et b_{54} (ligne 5) :

	A	B	C	D	E
{A,D}	a_1	b_{12}	b_{13}	a_4	b_{15}
{A,B}	a_1	a_2	b_{13}	a_4	b_{25}
{C,D,E}	b_{31}	b_{32}	a_3	a_4	a_5
{A,E}	a_1	b_{42}	b_{13}	a_4	a_5
{B,E}	b_{51}	a_2	b_{13}	a_4	a_5

Avec la DF suivante, $\{D,E\} \rightarrow \{C\}$, on a trois fois la paire de symboles $\langle a_4, a_5 \rangle$ (lignes 3, 4, 5) pour la paire de colonnes $\{D,E\}$. Le symbole a_3 (ligne 3) peut remplacer le symbole b_{13} (lignes 4 et 5) :

	A	B	C	D	E
{A,D}	a_1	b_{12}	b_{13}	a_4	b_{15}
{A,B}	a_1	a_2	b_{13}	a_4	b_{25}
{C,D,E}	b_{31}	b_{32}	a_3	a_4	a_5
{A,E}	a_1	b_{42}	a_3	a_4	a_5
{B,E}	b_{51}	a_2	a_3	a_4	a_5

Avec la DF $\{C,E\} \rightarrow \{A\}$, on a trois fois la paire de symboles $\langle a_3, a_5 \rangle$ (lignes 3, 4, 5) pour la paire de colonnes $\{C,E\}$. Le symbole a_1 (ligne 4) peut remplacer les symboles b_{31} (ligne 3) et b_{51} (ligne 5) :

	A	B	C	D	E
{A,D}	a_1	b_{12}	b_{13}	a_4	b_{15}
{A,B}	a_1	a_2	b_{13}	a_4	b_{25}
{C,D,E}	a_1	b_{32}	a_3	a_4	a_5
{A,E}	a_1	b_{42}	a_3	a_4	a_5
{B,E}	a_1	a_2	a_3	a_4	a_5

La dernière ligne du tableau ne comporte que des symboles a_i : la décomposition préserve le contenu de la base de données, mais attention, il est possible qu'elle ne préserve pas les dépendances fonctionnelles (cf. paragraphe suivant), donc certaines règles de gestion des données que ces dépendances représentent...

E.7.2. Préservation des dépendances fonctionnelles

Reprenons la règle de pseudo-transitivité présentée au paragraphe E.2 et qui nous a servi pour présenter la réduction des ensembles de DF : Si $\{A\} \rightarrow \{B\}$ et $\{B,C\} \rightarrow \{D\}$, alors $\{A,C\} \rightarrow \{D\}$.

A partir de là, définissons une relvar $R \{A,B,C,D\}$ à laquelle est associé le même ensemble S de DF :

$$S = \{\{A\} \rightarrow \{B\}, \{B,C\} \rightarrow \{D\}, \{A,C\} \rightarrow \{D\}\}$$

Le sous-ensemble I de S qui suit peut remplacer S puisqu'ils ont la même fermeture par rapport à R , $I^+ = S^+$:

$$I = \{\{A\} \rightarrow \{B\}, \{B,C\} \rightarrow \{D\}\}$$

R a pour seule clé candidate $\{A,C\}$, mais à cause de la DF $\{A\} \rightarrow \{B\}$ la 2NF est violée. Normalisons en mettant à profit le théorème de Heath. A partir de cette DF, on produit deux relvars respectant la BCNF (clés soulignées) :

$$R1\{\underline{A},B\}, R2\{\underline{A},\underline{C},D\} ;$$

Mais on perd la DF $\{B,C\} \rightarrow \{D\}$ ce qui entraîne la mise en œuvre d'une contrainte inter-relvars (cf. paragraphe 3.7). Par exemple, $R1$ ayant pour corps $\{\{a1, b1\}, \{a2, b1\}\}$ et $R2$ ayant pour corps $\{\{a1, c1, d1\}, \{a2, c1, d2\}\}$, le corps de la jointure naturelle de $R1$ et $R2$ est égal à $\{\{a1, b1, c1, d1\}, \{a2, b1, c1, d2\}\}$, en contradiction avec $\{B,C\} \rightarrow \{D\}$.

Cherchons donc une décomposition meilleure. A partir de la DF $\{B,C\} \rightarrow \{D\}$, R est décomposable de la façon suivante :

$$R3\{\underline{B},\underline{C},D\}, R4\{\underline{A},\underline{B},C\} ;$$

Toutefois $R4$ viole à son tour la 2NF, mais on peut poursuivre le travail de normalisation en décomposant $R4$:

$$R1\{\underline{A},B\}, R3\{\underline{B},\underline{C},D\}, R5\{\underline{A},\underline{C}\} ;$$

Cette fois-ci la BCNF est respectée (ainsi d'ailleurs que la 6NF), mais on peut se poser la question de savoir si la DF $\{A,C\} \rightarrow \{D\}$ est préservée : on peut le prouver à l'aide d'un théorème dont l'algorithme qui l'accompagne est le suivant (cf. [Ullman 1982], « Algorithm 7.3 : Testing Preservation of Dependencies ») :

```

Z = X
while changes to Z occur do
  for i = 1 to k do
    Z = Z ∪ ((Z ∩ Ri)+ ∩ Ri)

```

A cet effet on définit d'abord un ensemble G qui est l'union des projections sur S de $R1$, $R3$ et $R5$:

$$G = \pi_{AB}(S) \cup \pi_{BCD}(S) \cup \pi_{AC}(S).$$

Dans l'algorithme, X est le déterminant d'une DF $X \rightarrow Y$ appartenant à S . R_i représente successivement les éléments de la décomposition $\rho = (R1, R3, R5)$, à savoir $\{A,B\}$, $\{B,C,D\}$ et $\{A,C\}$. $(Z \cap R_i)^+$ est la fermeture de $(Z \cap R_i)$ par rapport à S . En fin de parcours, si Y est un sous-ensemble de Z , alors $X \rightarrow Y$ appartient à G . Si chaque DF telle que $X \rightarrow Y$ appartenant à S appartient aussi à G , alors ρ préserve S , dans le cas contraire il y a perte de DF.

Dans notre cas, la DF dont on veut s'assurer de la préservation est donc $\{A,C\} \rightarrow \{D\}$. On initialise Z avec le déterminant $\{A,C\}$ de cette DF et R_i est successivement égal à $R1$, $R3$ et $R5$ (puis $R1$ et $R3$ à nouveau s'il le faut).


Lors de la 1re itération, la situation évolue ainsi :

$$\begin{aligned}
Z &= \{A,C\} \cup ((\{A,C\} \cap \{A,B\})^+ \cap \{A,B\}) \\
&= \{A,C\} \cup (\{A\}^+ \cap \{A,B\}) \\
&= \{A,C\} \cup (\{A,B\} \cap \{A,B\}) && \text{Car } \{A\}^+ \text{ par rapport à } S = \{A,B\} \\
&= \{A,C\} \cup \{A,B\} \\
&= \{A,B,C\}
\end{aligned}$$

Lors de la 2e itération, la situation évolue ainsi :

$$\begin{aligned}
Z &= \{A,B,C\} \cup ((\{A,B,C\} \cap \{B,C,D\})^+ \cap \{B,C,D\}) \\
&= \{A,B,C\} \cup (\{B,C\}^+ \cap \{B,C,D\}) \\
&= \{A,B,C\} \cup (\{B,C,D\} \cap \{B,C,D\}) && \text{Car } \{B,C\}^+ \text{ par rapport à } S = \{B,C,D\} \\
&= \{A,B,C\} \cup \{B,C,D\} \\
&= \{A,B,C,D\}
\end{aligned}$$

Le dépendant $\{D\}$ de la DF $\{A,C\} \rightarrow \{D\}$ constitue un sous-ensemble de Z , donc $\{A,C\} \rightarrow \{D\}$ appartient à G . On montre de la même façon que les DF $\{A\} \rightarrow \{B\}$ et $\{B,C\} \rightarrow \{D\}$ appartiennent à G , en conséquence de quoi la décomposition $(\{A,B\}, \{B,C,D\}, \{A,C\})$ est sans perte de DF.

 La décomposition d'une relvar peut très bien préserver les dépendances fonctionnelles mais pas le contenu de la base de données. Reprenons l'exemple de la relvar $R \{A,B,C\}$ dont l'ensemble de dépendances fonctionnelles associé est $S = \{\{A\} \rightarrow \{B\}\}$ (cf. paragraphe E.7.1). On a montré que si l'on n'appliquait pas le théorème de Heath en décomposant R en $R_1 \{A,B\}$ et $R_2 \{B,C\}$, le contenu de la base données n'était pas préservé. Mais, à l'aide de l'algorithme ci-dessus, on peut facilement vérifier que cette décomposition préserve la DF $\{A\} \rightarrow \{B\}$.

Reconsidérons maintenant la relvar $R \{A,B,C,D,E\}$ (cf. paragraphe E.7.1) pour laquelle l'ensemble S de DF associé est le suivant :

$$S = \{\{A\} \rightarrow \{C\}, \{B\} \rightarrow \{C\}, \{C\} \rightarrow \{D\}, \{D,E\} \rightarrow \{C\}, \{C,E\} \rightarrow \{A\}\}$$

Et pour laquelle la décomposition retenue est la suivante :

$$\rho = (\{A,D\}, \{A,B\}, \{C,D,E\}, \{A,E\}, \{B,E\})$$

Comme cette décomposition ne résulte pas de l'application du théorème de Heath, il est possible que la préservation des DF ne soit pas assurée. Vérifions.

La DF $\{A\} \rightarrow \{C\}$ est-elle préservée ? Commençons avec le 1er élément de la décomposition, $\{A,D\}$:

$$\begin{aligned} Z &= \{A\} \cup ((\{A\} \cap \{A,D\})^+ \cap \{A,D\}) \\ &= \{A\} \cup (\{A\}^+ \cap \{A,D\}) \\ &= \{A\} \cup (\{A,C,D\} \cap \{A,D\}) && \text{Car } \{A\}^+ \text{ par rapport à } S = \{A,C,D\} \\ &= \{A\} \cup \{A,D\} \\ &= \{A,D\} \end{aligned}$$

Poursuivons avec le 2e élément de la décomposition, $\{A,B\}$:

$$\begin{aligned} Z &= \{A,D\} \cup ((\{A,D\} \cap \{A,B\})^+ \cap \{A,B\}) \\ &= \{A,D\} \cup (\{A\}^+ \cap \{A,B\}) \\ &= \{A,D\} \cup (\{A,C,D\} \cap \{A,B\}) \\ &= \{A,D\} \cup \{A\} \\ &= \{A,D\} \end{aligned}$$

Z n'a pas évolué. Poursuivons avec le 3e élément de la décomposition, $\{C,D,E\}$:

$$\begin{aligned} Z &= \{A,D\} \cup ((\{A,D\} \cap \{C,D,E\})^+ \cap \{C,D,E\}) \\ &= \{A,D\} \cup (\{D\}^+ \cap \{C,D,E\}) \\ &= \{A,D\} \cup (\{D\} \cap \{C,D,E\}) && \text{Car } \{D\}^+ \text{ par rapport à } S = \{D\} \\ &= \{A,D\} \cup \{D\} \\ &= \{A,D\} \end{aligned}$$

Z n'a pas évolué. Poursuivons avec le 4e élément de la décomposition, $\{A,E\}$:

$$\begin{aligned} Z &= \{A,D\} \cup ((\{A,D\} \cap \{A,E\})^+ \cap \{A,E\}) \\ &= \{A,D\} \cup (\{A\}^+ \cap \{A,E\}) \\ &= \{A,D\} \cup (\{A,C,D\} \cap \{A,E\}) \\ &= \{A,D\} \cup \{A\} \\ &= \{A,D\} \end{aligned}$$

Z n'a pas évolué. Poursuivons avec le 5e et dernier élément de la décomposition, $\{B,E\}$:

$$\begin{aligned} Z &= \{A,D\} \cup ((\{A,D\} \cap \{B,E\})^+ \cap \{B,E\}) \\ &= \{A,D\} \cup (\emptyset \cap \{B,E\}) \\ &= \{A,D\} \end{aligned}$$

Z ne peut plus évoluer et ne contient pas l'attribut C : la DF $\{A\} \rightarrow \{C\}$ n'est pas préservée, donc la décomposition ρ ne préserve pas les DF.

E.8. Conclusion

Démontrer qu'une relvar R vérifie la BCNF passe en théorie par le calcul de la fermeture S^+ de R , c'est-à-dire l'établissement de la liste complète des dépendances fonctionnelles pouvant être inférées de l'ensemble donné S des dépendances fonctionnelles vérifiées par R .

En fait, étant donné que l'on cherche fondamentalement à établir la liste des clés candidates (donc *a fortiori* la liste des surclés) pour vérifier que les dépendances fonctionnelles composant S n'entraînent pas un viol de BCNF, on peut éviter le calcul long et particulièrement monotone de la fermeture. Il suffit en effet d'en passer par les solutions de contournement très simples, grâce à l'utilisation de l'algorithme du seau et à la technique du rouleau compresseur. On évite ainsi de s'énerver, tenter désespérément d'enchaîner les axiomes d'Armstrong, et l'on fait des économies côté aspirine. Par ailleurs, déterminer un sous-ensemble de S qui soit irréductible permet le cas échéant de réduire au strict minimum le nombre de dépendances fonctionnelles composant S , donc les contraintes que l'on demandera au système de garantir.

En réalité, dans tout cela, il est surtout à souhaiter que l'ensemble S des dépendances fonctionnelles soit pertinent, c'est-à-dire que les règles de gestion des données correspondantes, issues du travail de conception en amont soient les plus complètes qu'il soit. Mais ceci est autre histoire...

Et tant qu'à faire, on s'assurera que les décompositions auxquelles on procède préservent le contenu de la base de données ainsi que les dépendances fonctionnelles, en n'hésitant pas à vérifier qu'il en est bien ainsi, en mettant à profit les algorithmes prévus à cet effet.

F. Identification relative versus identification absolue

F.1. Consommation des ressources physiques

Reprenons les tables utilisées pour illustrer l'alternative identification relative, identification absolue (cf. paragraphe 1.7).

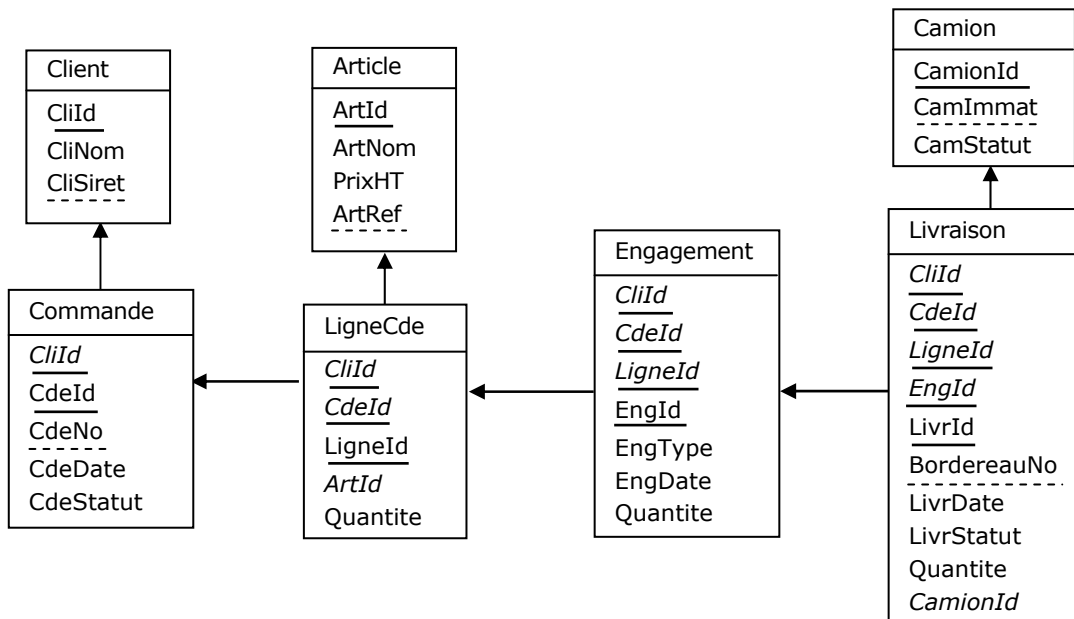


Figure F.1 - Identification relative et redondance intra-clés

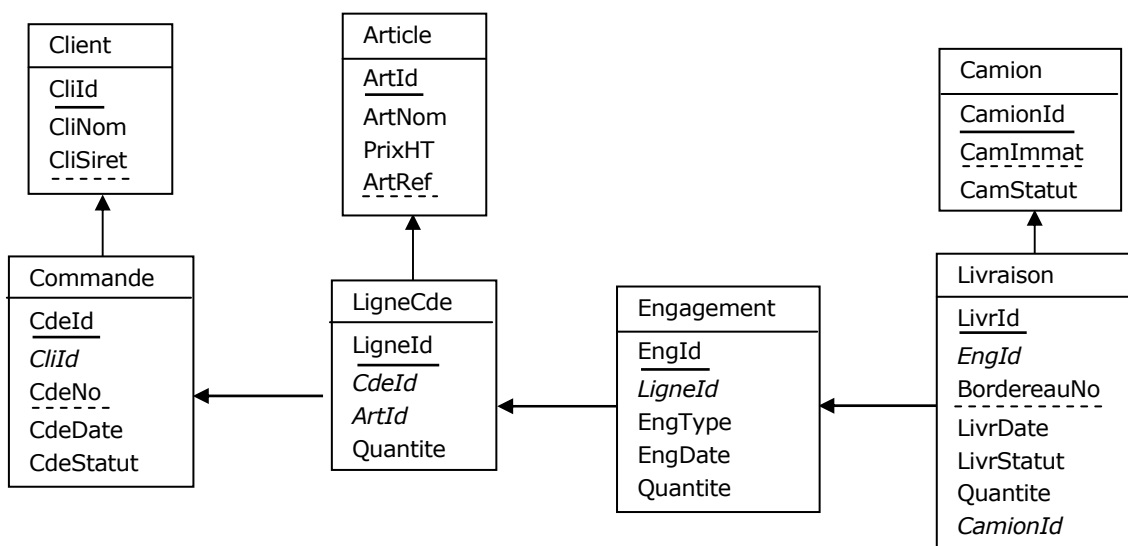


Figure F.2 - Identification absolue

Picturalement parlant, dans le cas de l'identification absolue on a diminué le nombre d'attributs des tables, mais en ce qui concerne la base de données, peut-on en conclure que cela entraîne une diminution de la volumétrie ?

Dans le cas de l'identification relative, la clé primaire de la table Livraison représente un encombrement de 12 octets (4 octets, soit 2^{32} valeurs pour l'attribut CliId et 2 octets, soit 2^{16} valeurs pour chaque autre attribut). Cette clé primaire contient la clé étrangère relative à la table Engagement, clé étrangère qui ne représente donc aucun surcoût. Dans le cas de l'identification absolue, la clé primaire {LivId} de la table Livraison représente un encombrement de 4 octets, ce à quoi il faut ajouter 4 octets pour la clé étrangère {EngId} relative à la table Engagement. Arithmétiquement parlant, dans le cas de l'identification relative, on a un surcoût de 4 octets : y a-t-il là de la gabegie ?

Quantifions maintenant la volumétrie au niveau des tables et des index et prenons l'exemple du SGBD DB2 for z/OS Version 8 (à chacun d'effectuer les calculs en fonction de son SGBD). Partons sur la base d'un million de clients, dix commandes en moyenne par client, etc., et faisons figurer les volumétries correspondantes (exprimées en giga-octets¹). Concernant les index, ne figurent que ceux qui nous intéressent, à savoir ceux qui sont imposés pour les clés primaires (PK) et ceux qui sont nécessaires pour la performance des jointures entre clés primaires (PK) et étrangères (FK) : Les index associés aux attributs correspondant à des points d'entrée à l'usage de l'utilisateur (n° Siret, n° de commande, n° de bordereau de livraison, etc.) ne sont pas pris en compte dans la mesure où ils ne jouent aucun rôle quant à la comparaison des coûts respectifs de l'identification relative par rapport à l'identification absolue. Pour mémoire — à l'intention des DBA qui comptent les accès au disque —, on a fait figurer le nombre de niveaux par index.

Identification relative					Identification absolue				
Table	Nb lignes	Table (Go)	Index PK/FK	Niveaux index	Table (Go)	Index PK	Niveaux index	Index FK	Niveaux index
Client	1 000 000	0,091	0,013	3	0,091	0,013	3		
Commande	10 000 000	0,464	0,144	3	0,485	0,122	3	0,064	3
LigneCde	50 000 000	1,357	0,830	4	1,357	0,608	4	0,720	4
Engagement	70 000 000	2,602	1,321	4	2,447	0,851	4	0,829	4
Livraison	80 000 000	4,460	1,684	4	4,075	0,973	4	1,064	4
Total (Go) :		8,974	3,992		8,455	2,567		2,677	
Tables + index : 12,966 Go					13,699 Go (1 giga-octet (Go) = 2 ³⁰ octets)				

Figure F.3 - Coûts de stockage comparés

Paradoxalement, la volumétrie est moins importante dans le cas de l'identification relative. Ceci est dû au fait que les index hébergeant les clés étrangères n'ont pas à être créés puisque « phagocytés » par ceux qui hébergent les clés primaires ; et l'on pourrait pénaliser l'identification absolue de près d'un Go supplémentaire si, en plus de la clé étrangère ciblant la table « parente », la clé de chaque index « FK » comportait la clé primaire de la table « fille » elle-même. Ainsi, l'intuition et certaines idées reçues peuvent être prises en défaut...

F.2. Performance des applications

Selon les volumes de données brassés par les applications, l'utilisation de l'identification absolue peut se révéler néfaste pour les performances. Dans le cas d'une transaction légère (*Joe transaction* par référence à une bande dessinée bien connue), l'utilisateur n'est pas pénalisé (les tableaux de bord de la Production informatique peuvent toutefois montrer une consommation globale élevée des ressources). En revanche, le temps d'exécution d'une requête lourde (*Jane Query*) et surtout d'un bon batch (*Bill Batch*) peuvent être très sensiblement pénalisés.

En effet, supposons que l'on ait besoin de savoir quels camions passeront chez quels clients. Descendons dans la soute. Si on utilise l'identification relative, grâce à l'attribut commun *ClId* on peut regrouper dans un même enregistrement sur le disque les commandes d'un client (effet *cluster* au sens DB2, voir paragraphe F.3). Même chose pour les autres tables : les lignes de chaque commande d'une commande d'un client donné se retrouvent dans un même enregistrement du fichier hébergeant ces lignes de commande, etc. Pour un client donné, la lecture d'un enregistrement physique de la table Livraison suffira (deux lectures si les données sont à cheval sur deux enregistrements) pour obtenir ce que l'on cherche (outre trois lectures pour l'index associé qui comporte quatre niveaux). Si un magasinier doit accéder aux livraisons à partir d'un numéro de bordereau de livraison et qu'il a besoin des données concernant les clients à livrer, l'effet cluster continuera à jouer pour la performance de la requête SQL correspondante.

Si on utilise l'identification absolue, pour arriver à une performance comparable en lecture, on s'en sortira pour la table Commande en rendant *cluster* (au sens DB2 ou SQL Server) l'index utilisé pour la clé étrangère {*ClId*}, mais dans le cas des tables suivantes on sera coincé (à noter qu'avec l'identification relative, on sait que l'on fait l'économie d'un index, ce qui est nettement meilleur pour la performance des mises à jour des tables). Les bienfaits du *clustering* sont détaillés dans l'annexe suivante (contexte DB2 for z/OS, mais ceci vaut vraisemblablement pour d'autres SGBD tels que SQL Server déjà cité).

¹ 1 giga-octet (Go) = 2³⁰ octets.

F.3. Le clustering selon DB2 for z/OS

Depuis sa naissance (1983), DB2 permet le regroupement et le rangement physiques (*clustering*) des lignes d'une table selon une séquence qui est celle de la clé d'un index en particulier.

Prenons le cas de la table Commande (cf. paragraphe 1.7, 4e exemple) : Nous voudrions regrouper toutes les commandes d'un client dans une même page (ou un minimum de pages physiquement adjacentes s'il y a beaucoup de commandes pour le client), afin de réduire le plus possible le nombre d'accès au disque pour manipuler tout ou partie de ces commandes. Le clustering sera effectué en fonction d'une clé qui en l'occurrence sera constituée de l'attribut CliId, servant à référencer la table Client (de clé primaire {CliId}).

Supposons que tel jour on a créé successivement des commandes pour les clients 12345, 40364, 00012, etc., logées par le SGBD dans les pages consécutives p_1 , p_2 , etc., à raison — pour fixer les idées — de 80 commandes par page.

Page p_1	Page p_2	
Une commande du client 12345	Une commande du client 50023	
Une commande du client 40364	Une commande du client 19781	
Une commande du client 00012	Une commande du client 00845	
Etc.	Etc.	...
Une commande du client 26542	Une commande du client 73251	

Figure F.4 - Hébergement physique des commandes

Les jours passant, si l'on crée une 2e commande pour le client 12345, celle-ci se trouvera dans une page p_i probablement fort éloignée de la page p_1 . Si donc on exécute une requête pour obtenir l'ensemble des commandes de ce client, il y aura deux accès au disque, un par commande, soit un coût de l'ordre de 10 millisecondes x 2 (sans tenir compte du surcoût lié à l'index). Au pire, la création de n commandes pour ce client peut se traduire par un coût d'accès de l'ordre de $10n$ millisecondes, pour peu que ses commandes se retrouvent toutes dans des pages différentes.

Page p_1	Page p_i	Page p_m
1 ^{re} commande du client 12345		
	2 ^e commande du client 12345	n ^{ième} commande du client 12345

Figure F.5 - Éparpillement des commandes

Notre objectif est donc de faire en sorte que le SGBD regroupe les commandes d'un client dans une même page. Techniquement parlant, cela implique la mise en œuvre d'un index — dit *cluster* — parmi l'ensemble des index associés à la table des commandes (en l'occurrence l'index qui sert pour la clé étrangère par rapport à la table Client). A cet effet, on dotera la table Commande d'un tel index :

```
CREATE INDEX Commandex1 (CliId, CdeId) ON Commande CLUSTER ... ;
```

En conséquence de quoi, le contenu de la page p_1 devrait théoriquement devenir le suivant :

Page p_1

1 ^{re} commande du client 12345
2 ^e commande du client 12345
3 ^e commande du client 12345
...
Dernière commande du client 12345
1 ^{re} commande du client suivant
...

Figure F.6 - Regroupement des commandes dans une même page

Mais il est évident que le client 12345 n'a pas l'exclusivité de la page p_1 et s'il passe sa 2e commande longtemps après la 1re, cette page ne pourra pas l'héberger, car contenant des commandes passées entre temps par d'autres clients. C'est pourquoi le DBA surveille les statistiques fournies par le catalogue relationnel qui permettent de savoir si l'effet cluster est effectif (*cluster ratio*) : au besoin, il procède à la réorganisation du *table space* hébergeant la table Commande, suite à quoi le clustering sera respecté à 100% (au moins pour un temps). A noter que l'on peut imposer à cette occasion que soit gelé un certain pourcentage de place libre (*free space*), récupérable seulement pour les INSERT à venir, et, tant qu'il trouvera de la place encore libre, DB2 essaiera alors d'insérer chaque nouvelle commande d'un client donné dans la page contenant celles que celui-ci a déjà passées.

Quoi qu'il en soit, suite à réorganisation, les n commandes d'un client seront regroupées dans la même page et le coût pour les lire ne sera plus de l'ordre de $10n$ millisecondes, mais plutôt de 10 millisecondes (plus environ $0,0004 \cdot (p-1)$ millisecondes si les n commandes occupent p pages adjacentes, la constante 0,0004 représentant le coût d'un accès séquentiel supplémentaire pour des disques de type IBM ESS).

Toutes choses égales par ailleurs, ce qui vaut pour les commandes vaut aussi pour les tables qui en dépendent directement ou non (table des lignes de commande, celle des engagements et celle des livraisons).

Cas de la table Client

Ce qui vaut pour la table Commande ne vaut pas pour la table Client, à savoir vouloir faire du clustering basé sur l'attribut *CliId*, au contraire. En effet, autant le regroupement des commandes d'un client se justifie d'un point de vue sémantique, ontologique, puis physique, autant le regroupement des clients sur un quelconque critère ne vaut pas (du moins intuitivement) : en effet, de quel objet les clients seraient-ils donc des propriétés multivaluées ? Seul un examen du fonctionnel, en relation avec le chef de projet de l'application, pourra guider le DBA pour mettre en évidence un critère intéressant du point de vue fonctionnel et qui permette de réduire la durée des traitements lourds (*Bill Batch*, *Jane Query*). A défaut, au moins dans le cas de DB2 — pour lequel existe exactement un index cluster par table —, on se servira d'un index autre que l'index primaire (de clé *CliId*), afin d'éviter des phénomènes de verrouillage (histoire vécue). Par exemple, si l'on retient pour être cluster l'index permettant à l'utilisateur d'accéder directement aux clients par leur numéro Siret, on a peu de risques de provoquer des verrouillages car, en cas de création simultanée et intensive de clients lors de transactions concurrentes, on a droit à un mécanisme de hachage naturel, provoquant un éparpillement bienvenu des clients dans les pages physiques. Cet éparpillement vaut non seulement pour la table Client, mais aussi pour la table Commande et pour sa descendance (du fait du 1er attribut de la clé, à savoir *CliId*), sans pour autant remettre en cause le clustering des commandes d'un client, donc la performance des traitements lourds.

Le rôle crucial de l'index cluster. I/O bound vs CPU bound.

Si un programme calcule le nombre PI avec des millions de décimales, le processeur sera très fortement mis à contribution, mais pas le disque. Dans le jargon, on dit que ce programme est *CPU bound* (*Compute bound* ou *Central Processing Unit bound*, selon les lieux et époques) : la durée du traitement dépend de la puissance du processeur. A l'opposé, il est un problème auquel sont confrontés pratiquement en permanence les DBA : celui de la lenteur de certaines transactions ou traitements lourds (encore *Jane Query* et *Bill Batch*) accédant à de grands volumes de données, auquel cas la durée du traitement est tributaire de la performance du disque ; toujours dans le jargon, on dit alors que le programme est *I/O bound* (*Input/Output bound*) ou encore *I/O Wait*.

Supposons par exemple que l'on ait une table CLIENT de 10 millions de lignes, accompagnée d'une table ADRESSE de 15 millions de lignes (un client peut avoir plus d'une adresse, telle qu'une adresse de facturation en plus de son adresse de domiciliation, et encore d'autres adresses pour d'autres usages).

Structure des tables :

```
CLIENT {ClientId, ClientNom, ...}
PRIMARY KEY {ClientId} ;

ADRESSE {AdrId, ClientId, AdrLigne1, AdrLigne2, ...}
PRIMARY KEY {AdrId}
FOREIGN KEY {ClientId} REFERENCES CLIENT ;
```

Supposons encore que l'on ait à effectuer la jointure (naturelle) de ces deux tables. Selon la syntaxe SQL :

```
SELECT      c.ClientId, c.ClientNom, ..., a.AdrId, a.AdrLigne1, a.AdrLigne2, ...
FROM CLIENT c JOIN ADRESSE a ON c.ClientId = a.ClientId ;
```

Si l'index cluster de la table Adresse a pour clé {AdrId}, à l'analyse des résultats on constatera que le processeur aura très peu travaillé et l'on aurait utilisé un processeur trois fois plus puissant que cela n'aurait pratiquement rien changé quant à la durée *elapsed* (horloge) ; en revanche, le système aura été en permanence en attente de fin des opérations de lecture sur le disque à l'origine d'un phénomène d'*I/O Wait* fort mal venu, avec une durée globale du traitement que l'on aurait volontiers vue divisée par un facteur très important...

Voici un exemple de simulation faite avec l'outil DB2 Estimator d'IBM. On a choisi le SGBD DB2 for z/OS version 8, un processeur de puissance modeste, à savoir un IBM 9672 Y16 (150 mips, 500 Mhz). Les disques sont des ESS-F20 (15 000 tours/minute). La lecture d'une page de 4K consomme 0,0004 sec. en mode séquentiel, de 0,005 à 0,010 sec. en mode aléatoire.

Estimation du coût de la requête suivante :

```
SELECT      *
FROM CLIENT c JOIN ADRESSE a ON c.ClientId = a.ClientId ;
```

Premier scénario : A la clé étrangère {ClientId} de la table ADRESSE on associe un index non cluster. La taille d'une ligne est en moyenne de 140 octets (avant compression à 80%).

Coûts estimés par l'outil : CPU Time = 00:32:23 ; **I/O Time = 20:50:17** ; Min. Elapsed Time = 21:22:23.

Ainsi, faut-il compter plus de 20 heures avant que la requête ne se termine, du fait des accès au disque (lectures synchrones). Pour réaliser la tâche, DB2 utilise la technique bien connue des DBA, dite du *Nested loop* (boucle imbriquée). DB2 utilise des techniques permettant de réduire assez sensiblement le coût des lectures synchrones, telles que celle du *List prefetch*, mais que nous ne pouvons pas aborder ici.

Deuxième scénario : L'index est rendu cluster :

Coûts estimés : CPU Time = 00:17:21 ; **I/O Time = 00:01:20** ; Min. Elapsed Time = 00:17:21.

Cette fois-ci, les accès au disque ne sont plus en cause, leur coût a été divisé par un facteur supérieur à 900, on n'est plus I/O bound, mais CPU bound : au besoin, et si on a le budget pour, la durée *elapsed* pourra être réduite en augmentant la puissance du processeur.

Là encore, DB2 utilise la technique du *Nested loop*.

Troisième scénario : Si l'index n'est pas cluster et si on peut le supprimer avant l'exécution de la requête (pour le recréer après, cf. au paragraphe 3.8 le coup de la suppression des chèques), DB2 n'utilise plus la technique du *Nested loop* mais celle du *Merge scan* (à la façon de l'appareillage de fichiers). Dans ces conditions, les coûts estimés sont les suivants :

CPU Time = 00:17:17 ; **I/O Time = 00:08:48** ; Min. Elapsed Time = 00:25:27.

Cette solution est loin d'être inintéressante. (Sans supprimer l'index, on peut aussi demander à DB2 d'utiliser la technique du Merge scan pour la requête).

Cas des prestations.

Supposons que la table CLIENT soit en relation avec une table de prestations des clients (comparable à celle des adresses quant à la taille d'une ligne et au pourcentage de compression), mais cette fois-ci pour une volumétrie de 80 000 000 de lignes.

Coût de la requête suivante :

```
SELECT      *
FROM CLIENT c JOIN PRESTATION a ON c.ClientId = a.ClientId ;
```

Premier scénario : A la clé étrangère {ClientId} de la table PRESTATION on associe un index non cluster :

L'emploi par DB2 de la technique du Nested loop donne lieu à des résultats catastrophiques, plus de 4 jours avant d'obtenir le résultat (des disques tournant à 100 000 tours/minute ne seraient pas d'un grand secours..., par contre, forcer DB2 à utiliser le Merge scan peut sauver la situation, ce qui est aussi possible s'il se sert du *list prefetch*, en stockant en mémoire les pointeurs (triés) vers les 1 000 000 de pages occupées par les 80 000 000 de prestations) :

Coûts estimés : CPU Time = 02:29:21 ; **I/O Time = 111:07:30** ; Min Elapsed Time = 113:36:01

Deuxième scénario : Si l'index est rendu cluster :

Coûts estimés : **CPU Time = 01:09:08** ; **I/O Time = 00:03:45** ; Min Elapsed Time = 01:09:08

Là encore les accès au disque ne sont plus en cause, on n'est plus I/O bound, mais bien CPU bound : une fois de plus, si cela est jugé nécessaire et si l'on a les budgets, le coût pourra être réduit en augmentant la puissance du processeur.

Conclusions

Dans le cas des traitements de masse, l'identification absolue a de fortes chances de causer de gros problèmes de performance. Les résultats présentés ne sont que les estimations de DB2 Estimator et, en situation réelle, il faudrait effectuer un EXPLAIN pour vérifier si DB2 utilise la technique du Merge scan (certes coûteuse en ressources du fait de tris inhérents) ou celle du Nested loop qui peut se révéler funeste. Et au-delà de DB2 for z/OS, cela vaut évidemment quel que soit le SGBD. Le choix de l'index cluster est déterminant pour la performance de ce type de traitements pour lesquels on accède à un nombre élevé de pages physiques, et autant on pourra arriver à réduire les durées en cas de CPU bound en augmentant la puissance du processeur, autant cela sera vraisemblablement vain en cas d'I/O bound, sauf à forcer DB2 à utiliser un Merge scan ou encore si celui-ci met en œuvre un list prefetch efficace, avec stockage en mémoire des pointeurs (triés) vers 1 000 000 de pages occupées par 80 000 000 de prestations, sans compter une cinquantaine d'autres tables de volumétrie comparable, compressées elles aussi au maximum.



La modélisation conceptuelle peut être impactée : il s'agit de l'alternative identification absolue vs identification relative : Pour reprendre l'exemple des clients et des commandes, si l'on s'en tient à l'identification absolue, on pourra rendre cluster non pas l'index primaire de la table COMMANDE mais celui qui sert pour la clé étrangère {CliId}, néanmoins on sera coincé quant à la performance des accès à la table des lignes de commande, le phénomène d'I/O bound réapparaissant inévitablement, hélas.

Et concernant l'incantation « la jointure ça coûte cher, donc il faut dénormaliser », voilà encore une légende à détruire... On peut dire que le coût de la jointure dépend très fortement de nos choix de modélisation conceptuelle et physique. Et, *Last but not least*, pour en revenir à notre sujet, n'oublions pas que la jointure est au cœur de la normalisation comme l'illustre le théorème de Heath (cf. paragraphe 3.3.2).

F.4. L'identification relative au service de l'intégrité des données

L'identification relative présente des avantages variés. Je reprends ici une discussion que l'on peut retrouver chez [DVP](#).

On traite des offres concernant des véhicules automobiles. Un des buts de la manœuvre est d'interdire par exemple que l'on associe une offre Renault à une motorisation Peugeot ou à une finition Citroën...

1er cas de figure. Identification absolue. Le MLD est le suivant (clés primaires soulignées, clés étrangères en italiques) :

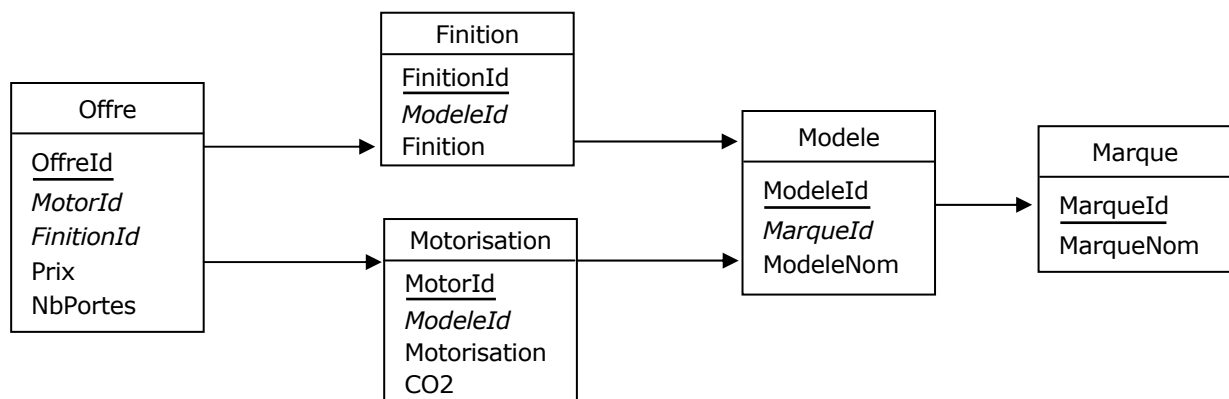


Figure F.7 - Offres - Identification absolue

Pour éviter les erreurs, il faudra programmer un trigger garantissant la cohérence des données.

2e cas de figure. On utilise l'identification relative. Le MLD devient le suivant :

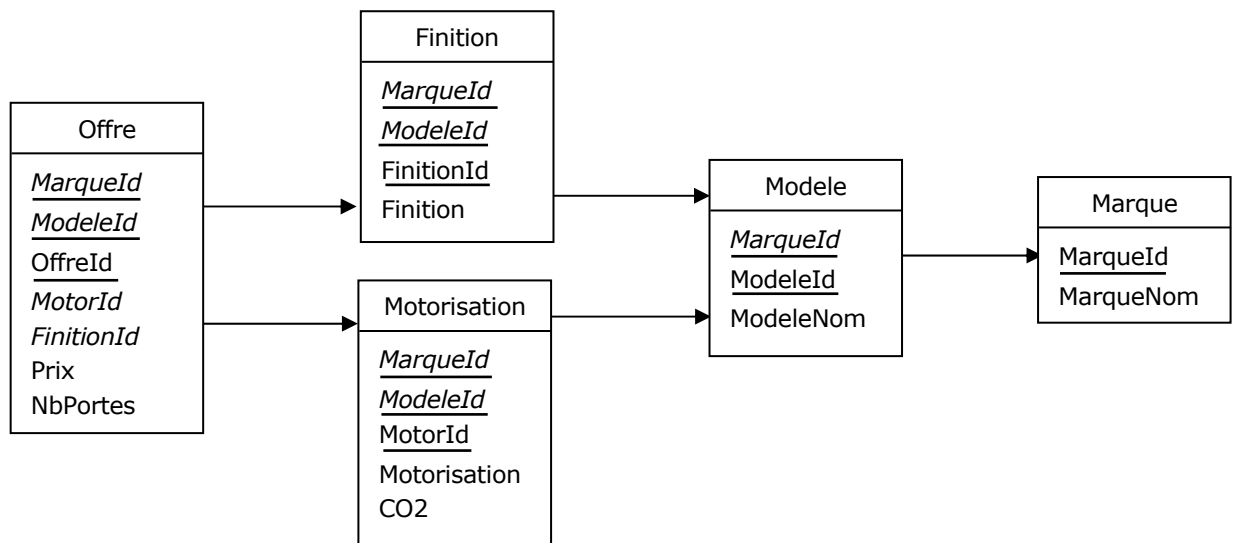


Figure F.8 - Offres - Identification relative

Du fait de l'identification relative (en association avec l'intégrité référentielle), il n'y a plus besoin de programmer un quelconque trigger pour s'assurer qu'une offre sera cohérente avec la marque, car l'attribut MarqueId est propagé jusqu'à la table *Offre*. Ainsi, chaque ligne de la table *Offre* prenant la valeur "Renault" pour l'attribut MarqueId ne peut faire référence qu'à une ligne de la table *Finition* prenant elle aussi la valeur "Renault" pour l'attribut MarqueId (intégrité référentielle oblige). A son tour, cette ligne de la table *Finition* ne peut faire référence qu'à une ligne de la table *Modele* prenant elle aussi la valeur "Renault" pour l'attribut MarqueId (intégrité référentielle oblige une fois de plus). A son tour, cette ligne de la table *Modele* ne peut faire référence qu'à une ligne de la table *Marque* prenant elle aussi la valeur "Renault" pour l'attribut MarqueId (intégrité référentielle, encore et encore). Ce qui vaut pour les liens qui unissent *Offre* et *Finition* vaut évidemment pour *Offre* et *Motorisation*.

Bibliographie

- [Abiteboul 1984] S. Abiteboul, N. Bidoit, *Non first normal form relations to represent hierarchically organized data*, ACM SIGACT-SIGMOD symposium on Principles of database systems, April 1984
- [Aho 1977] Alfred V. Aho, Catriel Beeri, Jeffrey D. Ullman. *The theory of joins in relational data bases*. *Proc. 19th IEEE Symp on Foundations of Computer Science (Oct. 1977)*. 107-113
- [Armstrong 1974] William Ward Armstrong. *Dependency structures of data base relationships*. In *Proc. of the IFIP Congress*, pages 580–583, Stockholm, Suède 1974.
- [Bernstein 1976] P.A. Bernstein, *Synthesizing third normal form relations from functional dependencies*. *ACM Transactions on Database Systems* 1:4, pp. 277-298.
- [Bouzeghoub 1990] Mokrane Bouzeghoub, Mireille Jouve, Philippe Pucheral. *Systèmes de bases de données, des techniques d'implantation à la conception de schémas*. (Eyrolles, 1990).
- [Codd 1969] E. F. Codd. « *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*. » IBM Research Report RJ599 (August 19th, 1969).
<http://portal.acm.org/citation.cfm?id=1558336>
<http://technology.amis.nl/blogacc/blog/wp-content/images/RJ599.pdf>
- [Codd 1970] E. F. Codd. « *A Relational Model of Data for Large Shared Data Banks* », *Comm. ACM*, V13, N6, June 1970, pp. 377-387.
<http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>
- [Codd 1971] E. F. Codd. « *Further Normalization of the Data Base Relational Model* ». IBM Research Report, RJ909, San Jose (August 31, 1971).
- [Codd 1974] E. F. Codd. « *Recent Investigations in Relational Data Base Systems* ». IBM Research Laboratory, San Jose (1974).
- [Codd 1980] E. F. Codd. « *Data models in Database Management* ». In *Proceedings of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling* (pp. 112-114). ACM New York, NY, USA.
- [Codd 1990] E. F. Codd. *The Relational Model for Database Management: Version 2* (Reading, Mass.: Addison-Wesley, 1990).
- [Date 1985] C. J. Date. *An Introduction to Database Systems: Volume II* (Reading, Mass.: Addison-Wesley, 1983. Reprinted with corrections July, 1985).
- [Date 1986] C.J. Date. *An Introduction to Database Systems: Volume I, 4th edition*. (Reading, Mass.: Addison-Wesley. 1986).
- [Date 1992] C. J. Date & H. Darwen. *Relational Database Writings 1989-1991* (Reading, Mass.: Addison-Wesley, 1992).
- [Date 1995] C. J. Date. *Relational Database Writings 1991-1994* (Reading, Mass.: Addison-Wesley, 1995).
- [Date 2003] C.J. Date, Hugh Darwen, Nikos Lorentzos. *Temporal Data and the Relational Model*. San Francisco, Calif.: Morgan Kaufmann (2003).
- [Date 2004] C.J. Date. *An Introduction to Database Systems, 8th edition*. (Pearson: Addison-Wesley (International Edition), 2004).
- [Date 2006] C.J. Date & Hugh Darwen. *Databases, Types, and the Relational Model, The Third Manifesto*, Third Edition (Pearson Addison-Wesley Longman, 2006).
- [Date 2007a] C.J. Date. *Date on Database : Writings 2000-2006*. (Apress, 2007).
- [Date 2007b] C.J. Date. *Logic and Databases, The Roots of Relational Theory*. (Trafford Publishing, 2007).
- [Date 2008] C.J. Date. *The Relational Database Dictionary, Extended Edition*. (Apress, 2008).
- [Date 2009] C.J. Date. *SQL and Relational Theory: How to Write Accurate SQL Code*. (O'Reilly Media, 2009).
- [Date 2010] C.J. Date and Hugh Darwen. *Database Explorations, Essays on The Third Manifesto and related topics*. (Trafford Publishing, juillet 2010).

- [Delobel 1973] C. Delobel, R.G. Casey. *Decomposition of a Data Base and the Theory of Boolean Switching Functions*. IBM Journal of Research and Development. Volume 17, Number 5, Page 374 (1973).
<http://www.research.ibm.com/journal/rd/175/ibmrd1705B.pdf>
- [Delobel 1982] C. Delobel, M. Adiba. *Bases de données et systèmes relationnels*. (Dunod informatique, 1982).
- [Diviné 1989] Michel Diviné. *Parlez-vous MERISE ?* (Eyrolles 1989).
http://michel-divine.developpez.com/#page_ParlezVousMerise
- [Fagin 1977] Ronald Fagin (IBM Research laboratory) *Multivalued Dependencies and a New Normal Form for Relational Databases* - ACM Transactions on Database Systems, Vol. 2, No. 3, September 1977, Pages 262-278.
<http://www.almaden.ibm.com/cs/people/fagin/tods77.pdf>
- [Fagin 1979] Ronald Fagin (IBM Research laboratory) *Normal forms and relational database operators*. - Proc, 1979 ACM-SIGMOD (ed. P. A. Bernstein), 153-160.
<http://www.almaden.ibm.com/cs/people/fagin/sigmod79.pdf>
- [Gardarin 1988] G. Gardarin. *Bases de données. Les systèmes et leurs langages*. (Eyrolles, 1988).
- [Jaeschke 1982] G. Jaeschke, H-J. Schek, *Remarks on the Algebra of Non First Normal Form Relations*, ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 1982.
- [Korth 1986] H.F. Korth, A. Silberschatz. *Database System Concepts*. (McGraw-Hill International Editions. 1986).
- [Korth 1988] M. Roth, H.F. Korth, A. Silberschatz. « *Extended Algebra and Calculus for Nested Relational Databases* » ACM TODS 13, No. 4, December 1988).
- [Maier 1983] David. Maier. *The Theory of Relational Databases* (Computer science Press, 1983).
<http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html>
- [Matheron 2002] Jean-Patrick Matheron. *Comprendre Merise, outils conceptuels et organisationnels*. 5e édition, revue et augmentée (Eyrolles, 2002).
- [Miranda 1986] S. Miranda et J.M. Busta. *L'art des bases de données. Tome 2. Les bases de données relationnelles*. (Eyrolles, 1986).
- [Miranda 1988] S. Miranda. *Comprendre et concevoir les bases de données relationnelles*. (éditeurs, 1988)
- [Morejon 1992] José Morejon. *Principes et conception d'une base de données relationnelle*. (Les Éditions d'organisation. 1992).
- [Nicolas 1978] J. M. Nicolas. « *Mutual dependencies and some results on indecomposable relations* ». Proc. Sept. 1978 VLDB (Berlin), 360-367.
- [Quine 1972] W.V.O. Quine. *Méthodes de logique*. (Traduction de M. Clavelin). (Armand Colin. Collection U. 1972).
- [RoMo 1989] A. Rochfeld et J. Morejon. *La Méthode MERISE. Tome 3. Gamme opératoire*. (Les Éditions d'organisation, 1989).
- [Tabourier 1986] Y. Tabourier. *De l'autre côté de MERISE* (Les Éditions d'organisation, 1986).
- [TRC 1989] H. Tardieu, A. Rochfeld, R. Colletti. *La Méthode MERISE, Tome 1. Principes et outils*. (Les Éditions d'organisation. 5ème impression, juin 1991).
- [Ullman 1982] J.D. Ullman. *Principles of DATABASE SYSTEMS, Second Edition*. (Computer Science Press. 1982).
- [Zaniolo 1981] C. Zaniolo and M.A. Melkanoff. *On the Design of Relational Database Schemata*, ACM TODS vol. 6, No. 1, March 1981.
<http://www.cs.ucla.edu/~zaniolo/papers/tods81.PDF>