Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Contribution au Processus d'Optimisation de Base de Données

Mathon, Jean-Noël

Award date:
1994

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 26. Apr. 2024

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
INSTITUT D'INFORMATIQUE
RUE GRANDGAGNAGE, 21B
B-5000 NAMUR

Contribution au Processus d'Optimisation de Base de Données

Mathon Jean-Noël

Promoteur : Mr Jean-Luc Hainaut

**Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en Informatique**

Année académique 1993-1994

ABSTRACT

Database design methods and tool-cases try to produce a correct and effective physical schema.
Nevertheless, production of physical schema restructured to encounter performance criteria such as space or time saving is not well studied so far.
This thesis is related to logical schema transformation being able to improve performance of databases. Schema restructuration such as vertical partitioning of an entity type or Relational DBMS translation transformation will be discussed.
Additionally, evaluation of some situations and some methodological recommendations will be made.

RÉSUMÉ

Les méthodes de conception de bases de données de même que les outils qui les supportent, visent à la production de schémas physiques corrects et opérationnels.
Le problème de la production de schémas physiques satisfaisant des critères de performances, tels que le temps de réponse ou le volume occupé, reste encore mal étudié à l'heure actuelle.
Ce mémoire présente des techniques de transformation du schéma logique telles que la découpe verticale d'un type d'entité ou l'élimination de structures non conformes au relationnel ainsi que leur évaluation dans différents cas de figure.
Quelques recommandations méthodologiques seront également données.

Je voudrais tout d'abord remercier Monsieur **Jean-Luc Hainaut**, Professeur aux Facultés Universitaires Notre-Dame de la Paix pour sa disponibilité et ses judicieux conseils, sans lui rien n'aurait été possible. **Merci.**

Je voudrais également remercier toute l'équipe d'OBLOG Software S.A. (Lisbonne Portugal) qui m'a accuelli si amicalement durant six mois, ainsi que Monsieur Jean-Marc Zeippen, responsable de stage. **Obrigado**.

Je voudrais aussi remercier tous ceux qui de près ou de loin ont contribué à l'élaboration de ce mémoire que ce soit par leurs conseils ou leur soutien moral. **Merci**.

Je voudrais enfin remercier tout particulièrement ma maman pour son soutien quotidien tout au long de mes études. **Merci**,

Jean-Noël

FM B16 / 1994 / 17

ERRATA

Page	Erreurs et leur correction
3.7	Cette redondance sera étudiée en 3.2.3.. Cette redondance sera étudiée en 3.2.4..
3.27	...facteur de blocage en diminution... ...facteur de blocage en augmentation...
3.35	...plus efficace avec un schéma dénormalisé. ...plus efficace avec un schéma normalisé.
3.69	...figure-?? ...figure-5
3.70	...figure-?? ...figure-6
5.2	Un "minimum locale"... Un "minimum local"... ...phase de restructuration logique.

Table des matières

Chapitre 1

Introduction

" Faites simple :
aussi simple que possible, mais pas simpliste "

Albert Einstein

1.1. Optimisation de base de données : un problème difficile

"Database tuning is the activity of making a database application run more quickly" D.E Shasha [SHA92]. "Plus rapidement" signifiera une plus grande et plus rapide absorption des traitements ou encore des temps de réponse plus courts.

Durant la conception d'une base de données (BD), la **production d'un schéma efficace** est souvent sous-estimée. Il y a plusieurs raisons à cela.

La principale raison est peut-être que l'optimisation d'une base de données n'est pas du tout chose facile. Cela nécessite en effet des connaissances dans beaucoup de domaines informatiques, allant de l'analyse fonctionnelle à la connaissance des SGBD (Système de Gestion de Base de Données) en passant par le hardware, les réseaux (quand par exemple la BD est de type distribué) ou encore le système d'exploitation.

De plus, les performances ne constituent pas le seul objectif à atteindre par un SGBD. Il faudra en effet faire la part des choses entre celles-ci et des objectifs comme la souplesse d'accès aux données, leur partage ou encore la sécurité. De plus, une amélioration globale des performances ne pourra être obtenue en additionnant des améliorations locales de performances, celles-ci étant souvent contradictoires.

Ensuite, peu de recherches sur ce sujet sont directement utilisables par le concepteur de BDs du fait de leur "trop forte mathématisation", qui implique la position d'hypothèses trop restrictives. A l'inverse, les conseils prodigués dans les guides de l'administrateur ou de tuning des SGBD sont souvent très faciles d'application mais ne mentionnent que rarement

leurs limites pourtant importantes. En outre, ils ne concernent très souvent que le design physique. Ainsi, on peut trouver des règles du genre :" Ne mettez pas d'index sur la table T si la requête sélectionne plus de 15% des articles de T ". Quoique ayant une explication et étant vraie dans beaucoup de cas, cette règle masque les mécanismes simples et sous-jacents à son énoncé. La compréhension de ces mécanismes permettrait une utilisation à bon escient de cette règle.

Enfin, peu d'outils CASE offrent une aide valable en ce domaine[1]. Ils adoptent en effet une stratégie de *draw-and-generate* qui donne l'illusion que la production d'un schéma exécutable de base de données peut être conduite par un processus direct une fois que le schéma conceptuel des données est saisi et admettent tout au plus une phase de tuning physique. Or, le schéma exécutable est le résultat final de toute une stratégie de conception dans laquelle le critère de performance pourra jouer un rôle important parmi d'autres critères.

Nous allons donc tout de suite dresser le panorama d'une stratégie de conception d'application "orientée donnée" qui supporte les exigences d'optimisations comme des temps de réponse courts, des consommations d'espace minimales, ou encore une amélioration de la sécurité, etc...

1.2. Optimisation et démarche de conception d'application orientée donnée

Le processus de conception d'une application orientée donnée est généralement considéré comme composé de deux activités parallèles, à savoir la conception de la structure de données et la conception de l'application qui utilise ces données. Ces deux activités sont intimement liées et il est impossible d'envisager une quelconque optimisation du schéma d'une base de données sans connaître l'application. Nous supposerons donc disponibles des informations en provenance de la conception de l'application.

La démarche de conception de la base de données très répandue et dans laquelle **nous essaierons de travailler**, peut être vue comme la succession de trois principaux processus utilisant chacun les résultats du précédent. Ces processus sont *l'analyse fonctionnelle, la conception logique et la conception physique*. Ils débutent avec l'analyse de l'existant et se terminent avec la production de code LDD (Langage de Description de Données) incorporant la description des structures physiques choisies. Ils transforment les produits du processus précédent en produits équivalents du point de vue du contenu sémantique mais incorporant certaines nouvelles caractéristiques répondant aux exigences qui conduisent le processus. Ces exigences, ces critères sont de natures diverses et ne sont pas pertinents pour l'ensemble des processus. La correction, **l'efficacité en terme de temps et/ou d'espace**, la conformité à un type de SGBD, ou même l'exploitation des possibilités d'un SGBD particulier, la facilité de

1 C'est par exemple le cas D'OBLOG CASE V1.00 produit par OBLOG software s.a. [OBLOG]. Après avoir introduit les spécifications complètes de l'application que l'on veut créer, on peut demander la génération totale du code de l'application, la définition de la base de données et des primitives d'accès à celle-ci. Les mérites de cet outil CASE étant certains, il est indiscutable que le processus de génération est ici vu comme une boîte noire sur laquelle le concepteur n'a aucune influence. Les concepteurs de l'outil sont conscients de ce problème et le prendront sans doute en compte lors d'une version ultérieure.

maintenance ne sont que quelques-uns des critères dont on peut tenir compte dans la conception d'une base de données.

La Figure-1 donne l'enchaînement des trois processus, leurs produits de transformation ainsi que quelques critères pouvant être pris en considération par ces processus. Nous les décrirons plus en détail à la lumière du critère d'efficacité. Ils se subdiviseront chaque fois en sous-processus.

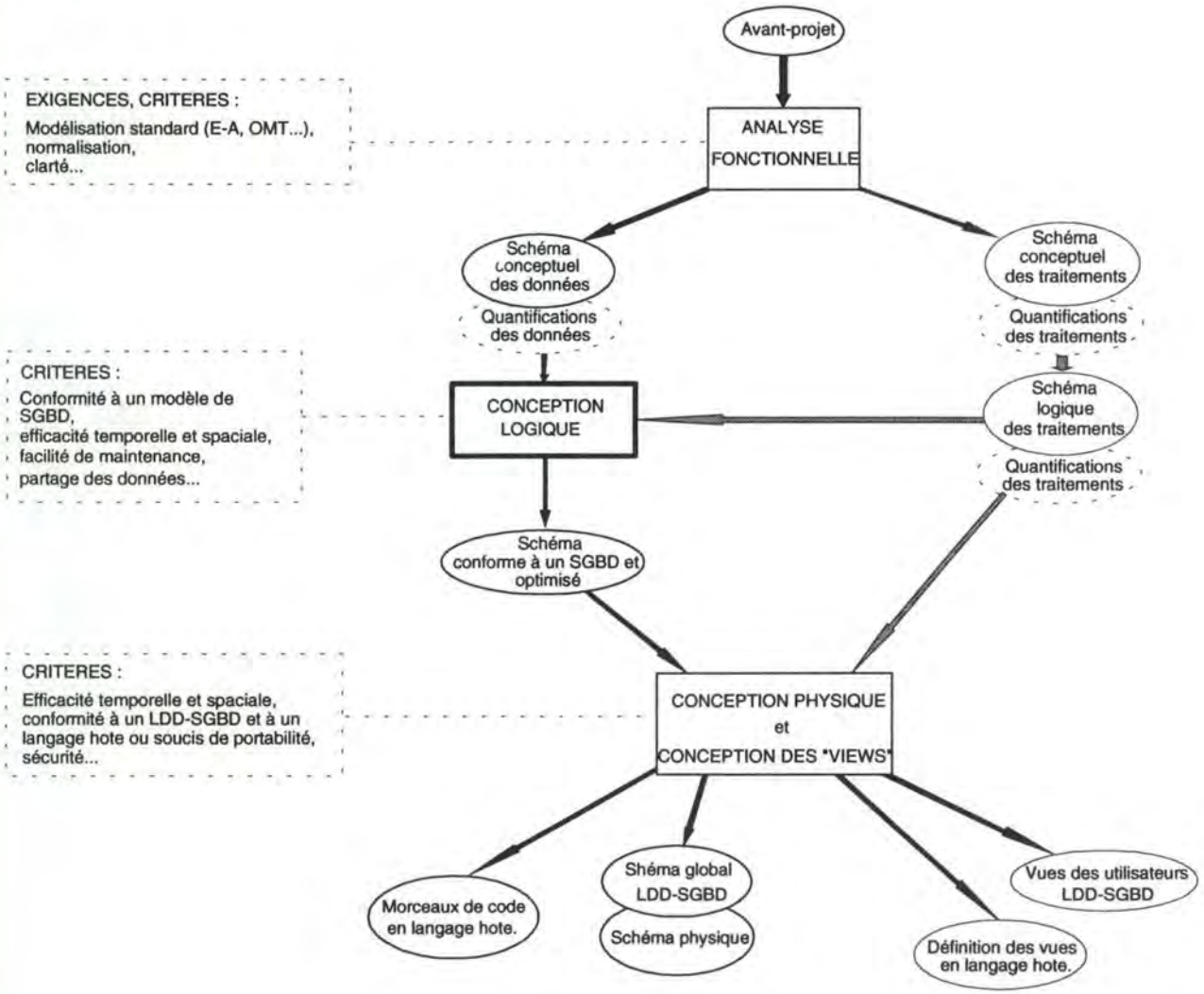


Figure-1 : *Processus de conception de base de données, une vue globale.*

1.2.1. L'analyse fonctionnelle

L'analyse fonctionnelle (conceptual design) [BOD89] transforme les exigences d'une organisation en un schéma conceptuel des données et un schéma conceptuel des traitements qui sont tous deux indépendants de toute technologie. C'est une formalisation du réel perçu. Nous supposerons que le modèle E-A (entité-association) enrichi du mécanisme de généralisation/spécialisation sera utilisé pour la description de ce réel perçu.

On ne parlera pas d'optimisation au niveau conceptuel, cela n'a pas vraiment de sens. Tout au plus peut-on parler de mauvaise conceptualisation qui pourrait entraîner des problèmes de performances.

Les critères pris en compte ici seront par exemple, la conformité à un modèle standard de spécification (comme le modèle E-A ou encore le modèle OMT[2]) qui augmente la "portabilité" de ces spécifications ou encore la normalisation[3] du schéma qui facilite la consolidation[4] de schémas, les contrôles de complétude, de cohérence,... et la clarté.

Cependant, pour pouvoir dans un premier temps évaluer la faisabilité de la solution issue du processus de conception et ensuite pouvoir améliorer les performances lors des étapes suivantes qui conduiront à la solution exécutable, il ne faudra pas négliger la collecte de différentes quantifications. Parmi ces quantifications, on trouvera d'abord les performances attendues. On pourra par exemple avoir des spécifications du genre :

```
...
La fonction Produit_a_commander doit se terminer en 5 secondes dans 95 %
des cas et ne peut jamais prendre plus de 20 secondes.
...
```

Deuxièmement, on donnera une description quantitative du schéma conceptuel des données comme :

```
...
Clients : 60000
Commandes : 160000
Commandes par client : 2.6 en moyenne, et entre 0 et 8
...
```

A partir de ces quantifications, on pourra rapidement évaluer le volume des données. On quantifiera enfin les traitements en donnant le nombre d'occurrences d'une certaine fonction par unité de temps :

```
Produit_a_commander : 100 fois par jour.
```

2 Object Modeling Technique est une méthodologie de développement de logiciel Orienté Objet qui propose un modèle de spécification des données [RUN91].

3 Un schéma est normalisé s'il ne contient pas des constructions indésirables par rapport au modèle de spécification standard choisi. Par exemple la présence de redondance est considérée comme une non-normalisation dans beaucoup de méthodologies et par exemple dans [BOD89].

4 La consolidation de schémas consiste en la fusion de schémas représentant une partie du système et qui peut-être, ont été développés séparément.

1.2.2. Conception logique

Le processus de conception logique (PCL) reçoit comme données le schéma conceptuel des données et si l'on veut optimiser quoique ce soit, les quantifications des données et des traitements sont indispensables. **C'est lors de ce PCL, que l'on pourra pour la première fois parler d'optimisation**. Le PCL peut-être décomposé en plusieurs sous-processus comme on peut le voir à la Figure-2. Il se termine avec la production d'un schéma que l'on peut traduire directement dans le LDD du SGBD choisi . C'est à peu près à la fin du PCL que s'arrête l'intervention de la version actuelle de l'outil CASE DB-main[5] lors de son utilisation en forward engineering.

Nous utiliserons pour cette phase le **modèle E-A étendu**[6] constitué d'un sous-ensemble des structures E-A de base et de deux structures d'accès qui se superposent aux premières.

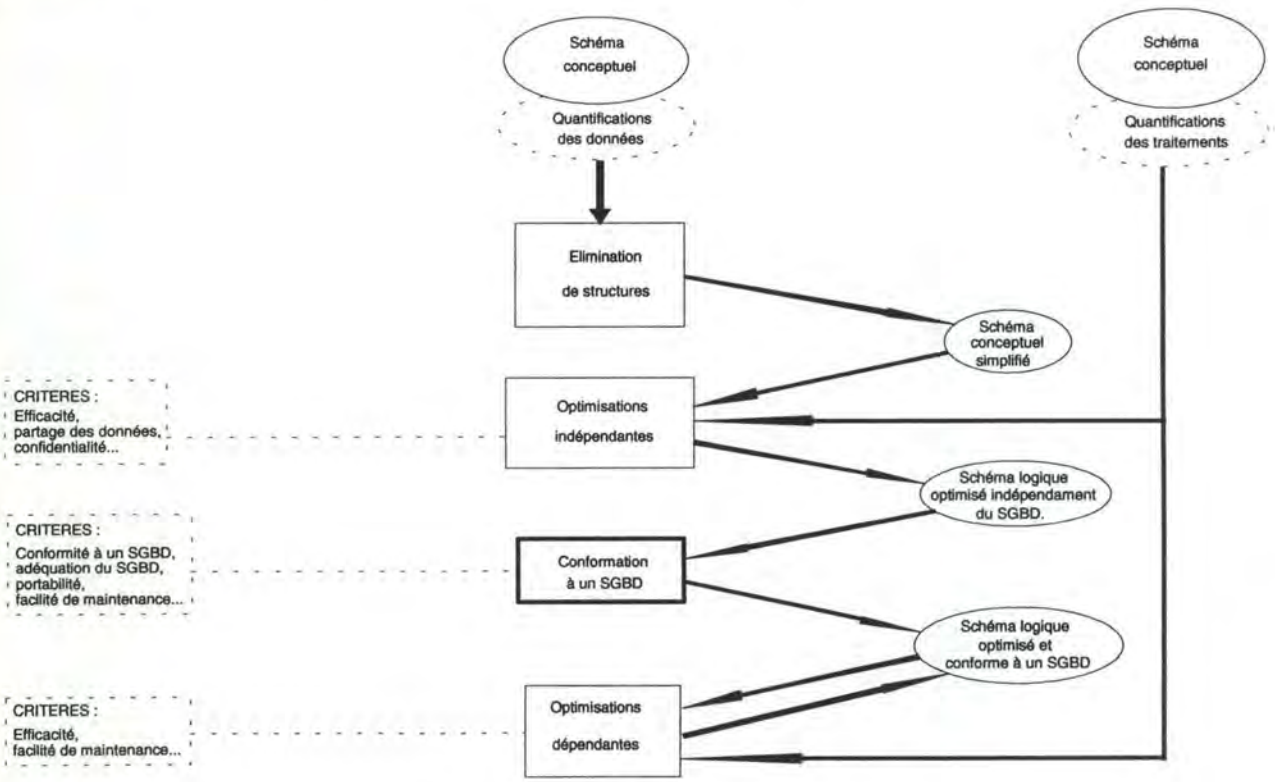


Figure-2 : *Processus de conception logique, il se résume souvent en une simple conformation à un SGBD ou un modèle particulier.*

---

5 DB-MAIN signifie **MAIN**tenance de **BD**. Cet outil supporte dans son état de développement actuel, un éditeur de schéma, une batterie de transformation de schéma à usage multiple et des procédures automatiques de traduction du schéma dans un modèle logique spécifique comme le relationnel. On consultera par exemple [DBMAIN] pour une présentation de l'outil. Cet outil est implémenté sous la direction de Monsieur Jean-Luc Hainaut, Professeur aux Facultés Universitaires Notre-Dame de la Paix à Namur.

6 Le modèle E-A étendu sera brièvement présenté à la section 2.1. L'utilisation du modèle E-A étendu améliore la continuité dans les formalismes utilisés depuis l'analyse conceptuelle jusqu'à la production du schéma conforme.

Le premier sous-processus du PCL est un processus de simplification qui transforme certaines structures avancées du modèle E-A en d'autres structures plus simples et plus adaptées aux raisonnements d'optimisations. Par exemple, on transformera une relation IS-A ou relation "d'héritage" en une relation one-to-one ou encore, une relation de degré supérieur à deux en relations binaires.

Le second sous-processus, prend en compte des optimisations indépendantes des SGBD. Il est conduit par des critères comme des temps d'accès minimaux, des consommations de ressources (espace disque) minimales, la distribution éventuelle des données ou encore la confidentialité. On essaie d'atteindre ces exigences en transformant le schéma conceptuel simplifié. Des mécanismes comme la **dénormalisation**, l'introduction de **redondances structurelles** ou des restructurations comme les **découpes verticales, horizontales** et les fusions de types d'entités seront utilisés. Ce sous-processus n'est pas souvent pris en compte.

La découpe dans les systèmes de base de données est un mécanisme fort utilisé qui permet de réduire la charge supportée par certains composants du système en la répartissant sur plusieurs ressources ou en la répartissant dans le temps. Le mécanisme de découpe n'est d'ailleurs pas seulement utilisé au niveau logique mais aussi au niveau physique.

Les optimisations indépendantes des SGBD comme la découpe sont souvent prises en considération au niveau conceptuel du fait de la stratégie de draw-and-generate des outils CASE. Elles n'y ont pourtant pas leur place vu que le but premier de l'analyse fonctionnelle est de capturer le réel sans aucune considération d'implantation.

Ce sous-processus résulte en la production d'un schéma logique optimisé indépendamment des SGBD (SLOI).

Il faudra ensuite rendre ce schéma conforme à un SGBD particulier. Cela implique le choix de ce SGBD ou plus exactement de son type, relationnel, hiérarchique ou réseau[7].

Ce choix pourra être influencé par des critères économiques comme la formation des programmeurs à un autre SGBD que celui employé d'habitude ou les coûts de maintenance moins élevés des applications utilisant un SGBD relationnel (SGBD-R). En effet, une des particularités des SGBD-R est qu'ils sont à même d'assurer une certaine optimisation des accès relatifs à une requête relationnelle. Il y a pour cela un "optimiseur[8]" qui prépare le plan[9] d'exécution de la requête au moment où elle doit être exécutée et cela en fonction d'informations statistiques et des catalogues[10].

Le programmeur peut donc souvent désigner des données sans se soucier des chemins et clés d'accès. Pour un schéma donné, l'optimisation des accès est donc confiée au SGBD.

---

7 Des SGBD-OO commencent à faire leur apparition sur le marché mais nécessitent une autre démarche de conception car par exemple l'étape d'élimination de structures avancées comme les relations is-a n'ont ici absolument aucun sens.

8 L'optimiseur est un processus du SGBD-R qui prend en charge la compilation des requêtes. C'est lui qui décide d'utiliser un index ou de plutôt lire séquentiellement toute une table, il produit un plan d'exécution de la requête.

9 Le plan d'exécution donne la manière dont va être exécutée la requête, c'est le résultat de la compilation par l'optimiseur de cette dernière. Ce plan pourra par la suite être utilisé comme outil de contrôle, comme nous le verrons.

10 Les catalogues contiennent toute la structure du schéma. Il y est par exemple indiqué que la table X possède un attribut y de type t, etc...

Ce n'est pas le cas pour les SGBD de type CODASYL par exemple. On doit ici spécifier le chemin d'accès aux données. Le programmeur est responsable de ce choix. Il fera ce choix en ayant certaines connaissances sur les données. Cette optimisation implicite, réalisée par ces derniers, aura cependant ses limites comme nous le verrons dans la partie sur le tuning des requêtes; il faudra alors "diriger" l'optimiseur.

Bien que le choix entre différents SGBD puisse être important du point de vue de l'optimisation, nous ne parlerons principalement que des SGBD-R qui deviennent de plus en plus répandus. Une fois ce choix fait, on transformera le SLOI en un schéma logique optimisé, conforme à un SGBD particulier et équivalent au premier. Ce processus de transformation respectera un ensemble de restrictions propres à un SGBD. Dans le cas du relationnel, on ne pourra, par exemple, pas avoir de type d'association. La phase de conception logique se résume souvent à ce seul et unique sous-processus de traduction.

Si maintenant, ce premier schéma conforme au relationnel est optimisé, il ne l'est que indépendamment des SGBD. Cependant, certains schémas conformes au relationnel seront plus efficaces que d'autres (tous sémantiquement équivalents entre eux) par rapport à un certain critère. On envisagera alors par exemple, différentes traductions des types d'associations, des attributs multivalués ou composés, constructions non conformes au relationnel.

Les informations quantitatives décrivant les transactions à effectuer sur la BD sont capitales au niveau logique. Sans la connaissance des fonctions qui accèdent à la base de données, la conception logique se résumerait à une conformation du schéma conceptuel à un SGBD sans qu'il soit possible de tenter une quelconque optimisation (indépendante des SGBD ou pas).

1.2.3. Conception physique

Il s'agit ici de produire une solution correcte par rapport à l'analyse conceptuelle, efficace et exécutable. Toutes les optimisations faites à ce niveau ont ceci en commun, elles n'influencent que rarement le programmeur d'application du moins pour les SGBD-R. C'est aussi à ce niveau que l'on définit les vues.

Dans la pratique, on ne pense qu'à optimiser la BD lors de cette phase de conception physique, le processus de conception logique s'étant souvent résumé à une conformation du schéma conceptuel à un modèle particulier.

Le premier sous-processus de ce processus de conception physique est le codage. Lors du codage, on donne la description du schéma logique dans le LDD propre à un type de SGBD ou même parfois à un SGBD particulier comme ORACLE V7. Ce codage peut être automatisé. Cependant, on peut décider de coder certaines contraintes d'intégrité (supportables par le SGBD) dans le programme d'application ou de laisser cette charge au SGBD. On peut par exemple ne pas vouloir laisser au SGBD le soin de vérifier qu'un attribut identifiant soit unique, si on sait pertinemment, de part la logique du programme d'application que cette unicité sera toujours vérifiée. On peut, au contraire, demander à certain SGBD de gérer des contraintes d'intégrité spécifiant par exemple qu'une commande soit toujours associée à un et un seul client (cardinalité minimale et maximale).

Des morceaux de code hôte et le texte LDD (parfois spécifique à un SGBD particulier) sont ainsi produits à l'issue de ce sous-processus de codage. Vient alors, mise à part la dérivation des vues, le sous-processus de tuning physique.

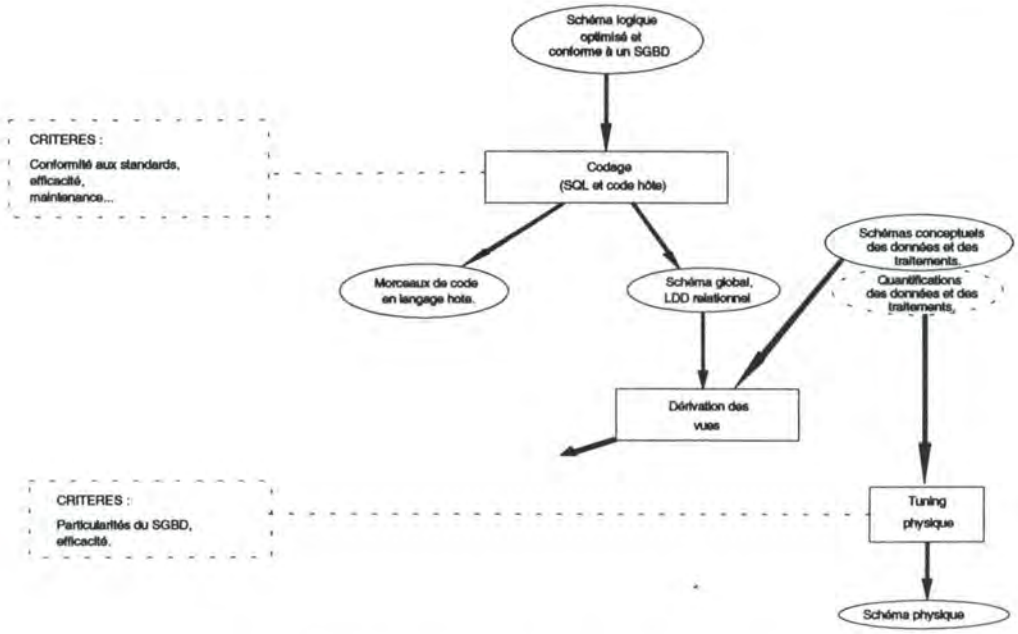


Figure-3 *: Processus de conception physique.*

On choisira entre autres lors du tuning physique :

- Les structures de stockage, les clés et chemins d'accès aux données. On les choisira en fonction des performances mais aussi de la facilité et du coût de maintenance, de reprise après incident ou encore de la sécurité. Chaque SGBD offre une plus ou moins vaste étendue de possibilité d'organisation des données et d'implémentation des chemins d'accès. On choisira par exemple entre des structures de fichier séquentiel indexé, de tas, de pile ou encore entre des B-tree et des structures de hashing, etc...
- Le taux de remplissage d'une page du disque : remplit-on la page à 100% afin de ramener le plus possible d'enregistrements à la fois en mémoire centrale ou laisse-t-on de la place pour d'éventuelles mises à jour qui pourraient faire augmenter la taille des articles ?
- La découpe des structures de données en fichiers. Une bonne politique d'allocation des fichiers pourra augmenter le parallélisme : si deux fichiers se trouvent sur deux disques différents, les accès seront plus efficaces.
- La stratégie d'allocation des extensions d'un fichier : elle peut par exemple être influencée par les prévisions d'évolution du nombre d'entités...

Ces paramètres physiques sont très nombreux et certains se retrouvent sous des appellations et syntaxes très diverses. Certains de ces paramètres sont même pour un SGBD particulier dépendants du système d'exploitation. Le tuning physique demande donc une bonne connaisance d'un SGBD particulier si l'on veut faire plus choisir un ensemble adéquat d'index, choix qui n'en est pas moins primordial.

Le tuning des mécanismes de reprises après incidents est aussi très important car il peut se révéler un très étroit goulot d'étranglement, ainsi d'ailleurs que le tuning des requêtes.

1.3. Sujet, Etat de l'art et apports

L'implémentation efficace d'une base de données étant un sujet extraordinairement large, il a fallu choisir le sujet à aborder. Ce choix s'est porté sur **l'optimisation de la BD lors des étapes de design logique** et sur le tuning des requêtes.

Ces deux sujets n'ont, à ce stade, qu'un seul point commun. Ils sont aux extrémités de l'échelle de facilité de mise en oeuvre des optimisations d'une base de données opérationnelle.

Le tuning des requêtes peut en effet être réalisé sans danger sur une application existante et apporter des gains appréciables de performances. L'optimisation logique n'est, elle, pratiquement pas concevable pour améliorer les performances d'une base de données existante.

De nombreux auteurs décrivent d'une manière très précise les modèles et les processus de conception relatifs au niveau conceptuel [BOD89], [RUN91],...

La littérature (surtout de la seconde moitié des années 70) est aussi très riche en études, modèles, **formulaires**, procédés conduisant à la définition de l'organisation et des paramètres physiques d'un fichier afin d'améliorer les performances. Le problème est que ces études mathématisées à souhait posent souvent des hypothèses qui rendent ces résultats inutilisables dans beaucoup de cas.

Par contre, **l'optimisation dans la phase de conception logique a engendré peu de recherches**. Un sujet que l'on abordera et qui est lui, traité dans la littérature, est le problème de la découpe verticale; il y est cependant traité à notre avis de manière trop "brutale".

Les phases de mise en oeuvre (design logique) sont en effet souvent considérées comme de pures formalités, de simples processus de traduction automatisables, des processus de traduction que l'on peut par exemple trouver dans l'outil DB-MAIN. Cependant, dans la pratique[11], la mise en oeuvre ne semble pas aussi simple que cela et l'optimisation dans la phase de conception logique plus importante qu'il n'y paraît.

Même le livre de D. Shasha [SHA92] utilisant l'intuition et une approche originale pour initier le lecteur aux principes de base de l'optimisation d'une base de données, ne passe en revue que très rapidement des problèmes comme la dénormalisation ou la redondance structurelle pour plutôt se consacrer au choix des index par exemple.

Ce vide dans la recherche peut se comprendre. Comme en programmation logique (Prolog) on se disait que tout irait plus vite quand les processeurs seraient plus puissants, dans la conception des bases de données on n'admet(tait) qu'une phase de tuning physique en se disant que l'on ne doit pas s'occuper des performances au niveau logique, que tout ira mieux quand les disques et CPU iront plus vite et qu'il ne faut (fallait) pas sacrifier les caractéristiques de lisibilité, ou de facilité de maintenance aux bénéfices des performances.

---

11 Nous avons pu nous en rendre compte lors du design logique de schémas partiels du repository de l'outil Oblog case V1.0 durant le stage de travail chez OBLOG software s.a. (Portugal).

Etant d'avis qu'il ne faut en effet pas sacrifier la lisibilité et la compréhension du schéma logique, nous verrons quelques optimisations logiques à même d'améliorer les performances sans (trop) rendre la compréhension du schéma hermétique et complexifier la tâche du programmeur.

La sensibilisation du concepteur aux opportunités d'optimisations logiques indépendantes de tout SGBD et propres au SGBD-R ainsi que certains apports méthodologiques, le tout sur un ton à mi-chemin entre l'intuitif et le formel, voilà les principaux intérêts de ce texte. Nous ne prétendons cependant pas avoir traité le sujet choisi dans son entièreté.

1.4. Structure du mémoire

La structure du chapitre 3 reflète la démarche de conception logique brièvement présentée en 1.2.2. Il se subdivisera en trois parties, une courte présentation des simplifications de schéma (3.1), une présentation des transformations du schéma logique destinées à améliorer les performances et ce de manière indépendante de tout SGBD (3.2), une présentation de quelques optimisations logiques propres au SGBD-R (3.3), et enfin on y ajoutera une courte mise en garde au problème pratique d'écriture des requêtes SQL (3.4).

Le chapitre 4 dressera quant à lui, une liste de sujets non explorés, et qui pourtant peuvent être la source de contre-performances, ainsi qu'une liste des fonctionnalités qui pourraient être ajoutées à l'outil DB-MAIN pour aider le concepteur dans sa tâche d'optimisation.

Le chapitre qui suit passe en revue les notions que le lecteur devrait maîtriser. Ce chapitre n'est qu'un bref rappel et n'est donc pas exhaustif.

Chapitre 2

Technologie, Modèles et Transformations

" Toutes choses sont déjà dites ; mais comme personne n'écoute, il faut toujours recommencer ".

André Gide
Traité de Narcisse

2.1. Modèle entité-association étendu

Nous allons utiliser tout au long de ce travail un modèle de type entité-association (E-A) que nous appellerons modèle entité-association étendu. Il est inspiré du modèle proposé par Chen en 1976 et présenté dans [HAI93]. On pourra trouver les fondements formels de la plupart des aspects du modèle dans [HAI89].

En fait, on pourra exprimer dans ce modèle aussi bien les schémas conceptuels, logiques ou physiques, ce qui permettra de franchir les étapes du processus de conception sans changer de modèle.

Nous allons distinguer deux couches dans le modèle, la couche conceptuelle et la couche technique.

La couche conceptuelle est constituée des concepts standards du modèle E-A enrichi de quelques extensions. Il englobe les concepts suivants, la plupart d'entre eux étant représentés à la figure-1.

- Type d'entité comprenant un nombre quelconque (zéro inclus) d'attributs;
- La relation d'héritage IS-A;

- Type d'association (appelé aussi relation) comprenant deux ou plusieurs rôles et un nombre quelconque d'attributs. Nous ne rencontrerons cependant que des associations de degré 2 (deux rôles) et sans attribut. A chaque rôle est associée une contrainte de cardinalité `[min-max]` qui indique les nombres minimal et maximal d'associations dans lesquelles une entité peut jouer un rôle; Une association de degré deux est dite être une association one-to-one si les cardinalités maximales des deux rôles sont égales à 1. Si une seule des cardinalités maximales est égale à 1, l'association est dite one-to-many. Elle est dite many-to-many dans les autres cas.

- Un attribut est soit simple ou composé; un attribut simple a un domaine de valeur; on donne à chaque attribut une contrainte de cardinalité `[min-max]` indiquant combien de valeurs peuvent être associées à son parent (type d'entité, type d'association ou encore attribut composé); un attribut multi-valué (cardinalité `max > 1`) peut être un ensemble de valeurs, un pseudo-ensemble acceptant des valeurs équivalentes en son sein ou une liste ordonnée;

- Un type d'entité peut posséder un nombre quelconque d'identifiants (candidate key), un identifiant étant constitué d'attributs et/ou de rôles; un des identifiants est déclaré primaire (primary key);

- Un type d'association a au moins un identifiant constitué de rôles et/ou d'attributs; un rôle avec une cardinalité `[min-1]` est un identifiant; quand aucun identifiant n'est spécifié ou ne peut être déduit, alors tous les rôles du type d'association forment son identifiant ;

- Des contraintes d'intégrité peuvent être associées avec ses constructions; mentionnons les contraintes d'inclusion, référentielle, de redondance, d'exclusion, de coexistence (un groupe d'attributs et/ou de rôles d'un type d'entité dont les valeurs sont simultanément présentes ou absentes) et les dépendances fonctionnelles.

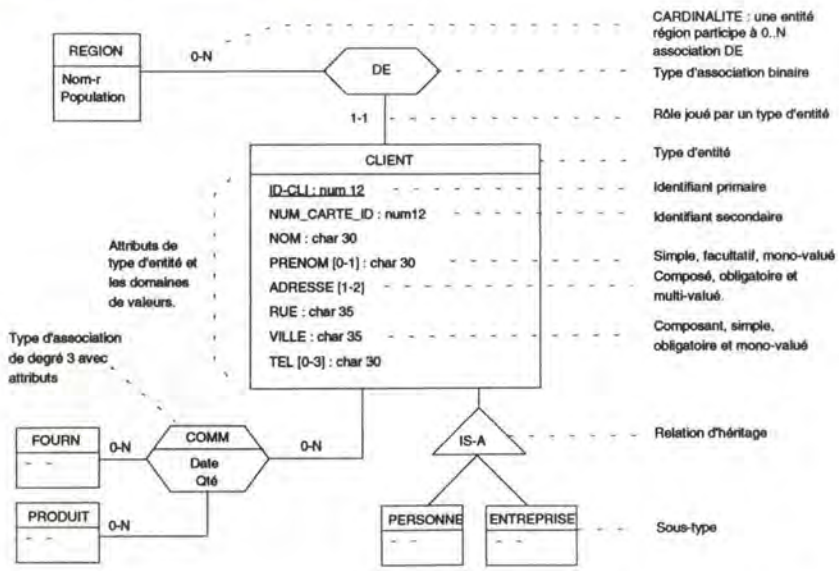


Figure-1 : *Représentation graphique de quelques éléments de la couche conceptuelle.*

La couche technique inclut des constructions qui appartiennent aux structures de descriptions logique et physique (figure-2). Ces concepts sont ajoutés à la couche conceptuelle décrite ci-dessus.

- Un fichier est un ensemble d'entités; à ce niveau, une entité est une abstraction d'un record ou une ligne;
- Une clé d'accès est un groupe d'attributs auxquels est associé un mécanisme d'accès; c'est une abstraction d'un mécanisme d'accès basé sur les valeurs tels que les index, les fichiers hash, etc...;
- Un chemin d'accès est un mécanisme d'accès permettant la navigation à travers les types d'association; cela permet de spécifier les *set types* (CODASYL) et les relations *parent-child* (IMS) par exemple;
- Les entités dans un fichier, les entités attachées avec une autre entité à travers un type d'association et les valeurs des attributs multi-valués de type liste, peuvent être ordonnées suivant le moment d'insertion ou une clé de tri;
- Un attribut a une longueur physique dépendant de l'encodage et qui peut être fixe ou variable

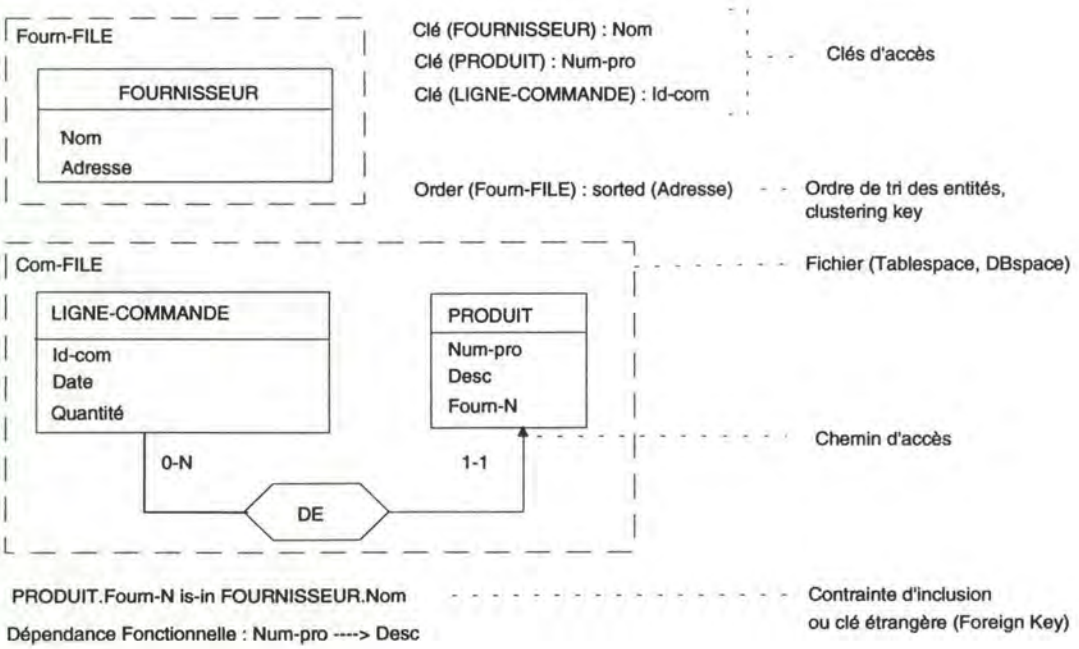


Figure-2 : *Représentation graphique de quelques constructions de la couche technique du modèle.*

2.2. Approche Transformationnelle en tant qu'Outil de Conception

2.2.1. Notions et Concepts de Transformation

La notion de transformation de schéma est à la base de beaucoup d'études concernant principalement la **conception logique** et le **reverse engineering** de base de données.

Dans notre cas, elles seront utilisées pour apporter des optimisations logiques au schéma et ce, lors des quatre sous-processus de la conception logique présentés en 1.2.2.

Plus généralement, on peut dire que la transformation d'un schéma source en un schéma cible, consiste à modifier ce premier selon certains critères pour obtenir un nouveau schéma **sémantiquement équivalent** mais plus spécialisé.

L'équivalence sémantique des schémas est une notion importante. Elle signifie, que les deux schémas représentent le **même réel perçu**.

Une transformation T_1 est dite **réversible** si (informellement), étant donné les instances du premier schéma, après transformation du schéma et des instances, il existe une transformation inverse T_2 qui permette de retrouver le premier schéma et ses instances.
Une transformation T_1 est dite **symétriquement réversible** (SR) si T_1 et sa transformation inverse T_2 sont toutes deux reversibles. Il est souhaitable d'appliquer des transformations symétriquement réversibles mais ceci ne sera pas toujours possible.

On peut trouver dans [HAI91], [HAI93] ou encore dans [ULL88] une présentation rigoureuse de transformations dites de projection-jointure, de dénotation et d'extension ainsi que leur expression formelle.

Donnons cependant tout de suite quelques exemples de transformations très répandues (figure-3, 4 et 5) sous la forme du modèle entité-association étendu que l'on vient de présenter brièvement.

2.2.2. Exemples de Transformations SR

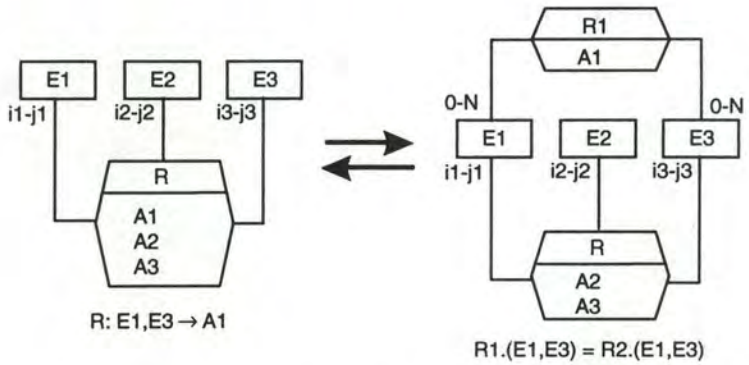


Figure-3 : *Transformation de Projection-Jointure d'un Type d'Association Non-Normalisé.*

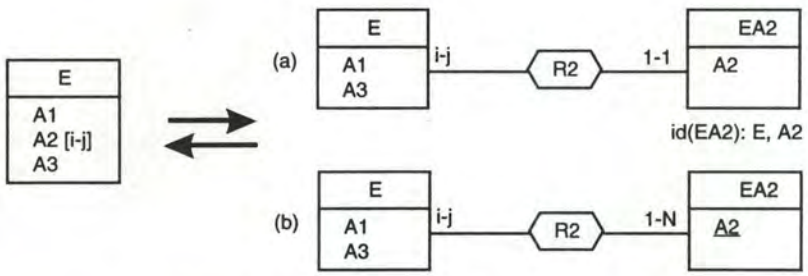


Figure-4 : *Transformation d'Extension d'un type d'entité par représentation des instances d'un de ses attributs (a) et par représentation des valeurs de ce même attribut (b).*

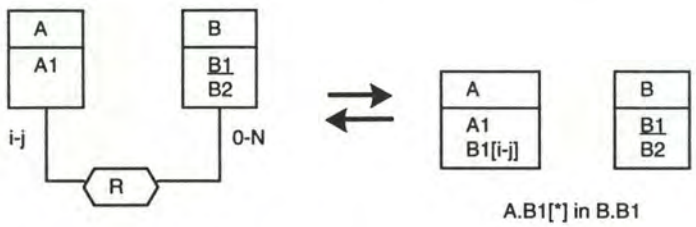


Figure-5 : *Transformation de Dénotation; représentation d'un type d'association par une foreign key multivaluée*

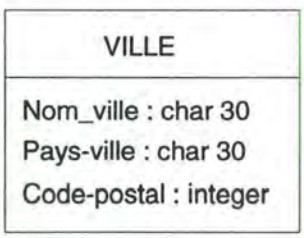
2.3. Système relationnel

Le modèle relationnel n'est pas le premier modèle proposé pour définir le schéma d'une base de données. Cependant, depuis la première proposition de CODD en 1970, sa popularité et son usage effectif dans les prototypes et les systèmes commerciaux, n'ont jamais cessé de croître. On peut dire aujourd'hui que le modèle relationnel et les SGBD-R règnent en maître sur le marché des bases de données.

2.3.1. Le modèle relationnel

2.3.1.1. La notion de relation

Le modèle relationnel se fonde sur le concept mathématique de relation. Une relation est un sous-ensemble du produit cartésien de différents domaines. Pour des raisons évidentes, on ne s'intéresse en fait qu'à des relations finies, même si les domaines sur lesquels elles sont construites sont infinis (les entiers par exemple). La relation de la figure-6 est construite à partir de trois domaines : le domaine des noms de villes (implémenté en char 30), le domaine des entiers et celui des noms de pays (aussi implémenté en char 30).



Id (VILLE) = (Pays-ville, code-postal)

Figure-6 : *Une relation reprenant pour chaque Ville, son Code postal et son Pays.*

Le schéma de relation (type d'entité) est appelé la partie intentionnelle de la relation, tandis que la liste des lignes (entités) de la relation est la partie extensionnelle (ou encore ensemble d'instances).

2.3.1.2. Schéma relationnel

On sait bien évidemment représenter un schéma conforme au modèle relationnel à l'aide du modèle entité-association étendu.

Cependant, tant que l'on reste au niveau des structures de données, le modèle relationnel est particulièrement pauvre (comparé aux modèles CODASYL ou OO).

Le schéma conceptuel contient uniquement des types d'entités, des attributs simples, mono-valués, facultatifs ou non, des identifiants, des contraintes référentielles et des clés d'accès.

Il n'existe donc pas de type d'associations, tous les types d'entités possèdent au moins un attribut, et les seules contraintes d'intégrité reconnues sont les identifiants et les contraintes

référentielles. Les autres contraintes seront supportées par l'application ou par des mécanismes de CHECK et de TRIGER[1] proposés par certains SGBD-R..

Les concepts de la couche technique du modèle E-A étendu supportés par les SGBD-R sont les clés d'accès et les espaces ou fichiers. De plus, tout identifiant sera une clé d'accès. Dans certains rares SGBD, les entités d'un même TE pourront être stockées dans plusieurs espaces ou fichiers (découpe horizontale au niveau physque).

Le vocabulaire particulier des SGBD-R est le suivant :

• Type d'entité	→	table,
• Entité	→	ligne, (tuple),
• Attribut	→	colonne (column),
• Identifiant	→	identifiant primaire (Primary Key), ou index identifiant (unique index), ou clause unique du *creat table* (voir suite),
• Attribut de référence	→	clé étrangère (Foreign Key),
• Clé d'accès	→	index,
• Espace ou fichier	→	Tablespace, DBspace, etc.

2.3.1.3. Algèbre relationnelle

Les domaines et relations étant des ensembles au sens mathématique[2] du terme, on peut leur appliquer la totalité des opérateurs ensemblistes et du calcul des prédicats.

Codd propose une algèbre relationnelle composée d'un jeu d'opérateurs équivalent au calcul des prédicats du premier ordre. Ce jeu d'opérateurs est composé des opérateurs suivants :

- L'**union** de deux ou plusieurs domaines ou relations, notée ∪,
- La **différence** de deux domaines ou relations, notée \,
- Le **produit cartésien** de deux ou plusieurs domaines ou relations, noté X,
- La **projection** d'une relation : relation obtenue en ne conservant de la relation initiale que les valeurs de certains attributs. L'extension d'une relation étant un ensemble, certaines lignes seront amenées à disparaître. On note R[a, b] la projection de R sur les attributs a et b.
- La **sélection** : relation obtenue en ne retenant de la relation initiale que les lignes dont les valeurs vérifient les conditions de sélection. On note A(a='valeur') la sélection des lignes de A ayant pour valeur de l'attribut a, la valeur 'valeur',
- La **jointure** de deux relations : permet de construire une relation en accouplant les lignes de deux relations qui ont une même valeur pour un ou plusieurs attributs compatibles. On note A(a) * B(b) la jointure des tables A et B sur les attributs compatibles a et b. L'opération SQL issue de cet opérateur est le "join".

---

1 On consultera [DAT90] pour plus d'informations sur ces mécamismes.

2 Les domaines et relations peuvent être considérés comme des ensembles au sens mathématique du terme si du moins on ne considère que ces concepts sous un aspect purement statique, c'est-à-dire sans considérer les insertions, suppressions et mises à jour.

A partir de ces opérateurs, il est possible de définir un langage de définition et de manipulation de domaines et de relations. Le langage le plus répandu utilisant ces opérateurs est le langage SQL.

2.3.2. Utilisation d'un système relationnel

Nous ne voulons ici que rappeler très brièvement les opérations proposées par SQL. On se réfèrera par exemple à [IBM81] pour une présentation complète du langage SQL standard.

2.3.2.1. Langage de Définition de Données

La commande principale permettant de créér une relation est la commande `create table` qui prend un nom de relation et un ensemble de définition d'attributs avec les domaines associés, comme dans l'exemple suivant :

```
CREATE TABLE      Client (
num_cli           integer,
nom               char(30),
prenom            char(30),
adresse           char(40),
...                     );
```

Les domaines des attributs en SQL peuvent être des entiers, des nombres en virgule flottante, ou des chaînes de caractères. Certains SGBD-R proposent d'autres domaines spécifiques.

2.3.2.2. Schéma physique

Le langage SQL offre la possibilité de définir des index pour accélérer l'accès à certaines relations. La création d'un index peut par exemple prendre la forme suivante :

```
CREATE INDEX      ind-1
ON                Client (Nom) ;
```

Un index peut être créé sur plusieurs attributs de la relation à la fois. L'utilisateur peut également préciser s'il s'agit d'un clustering index si il veut que les lignes soient triées. La clause `unique` permet enfin de préciser que les attributs de l'index forment un identifiant pour la relation.

```
CREATE UNIQUE INDEX     ind-0
ON                      Client (Num-cli) ;
```

On peut remarquer que le choix de SQL d'associer à la contrainte de clé (identifiant primaire ou secondaire) la création d'un index, va à l'encontre du principe d'indépendance physique prôné par les fondateurs du modèle relationnel. Il est clair que, dans ce cas précis, le choix a été motivé par des considérations d'implémentation et d'efficacité puisqu'il est beaucoup plus simple et performant de vérifier une contrainte de clé lorsqu'un index est présent.

2.3.2.3. Langage de Manipulation de Données

2.3.2.3.1. Sélection

La manipulation de données en SQL est centrée sur la construction select... from... where qui est un filtre très général permettant de construire des relations à partir des relations de la base. On peut par exemple rechercher la liste des clients de nom égal à "DUPONT" :

```
SELECT      *
FROM        Client
WHERE       Nom = 'Dupont';
```

La clause **select** définit les attributs de la relation résultante, la clause **from** définit la (ou les) relation sur laquelle porte la recherche, et la clause **where** définit une condition de sélection.

2.3.2.3.2. Insertion

Pour introduire de nouvelles données dans une relation, on utilise la commande **insert into** :

```
INSERT INTO       client
VALUES            (10, 'dupont', ...);
```

Cette commande ne contient évidemment pas de clause de sélection **where**.

2.3.2.3.3. Suppression

Pour supprimer une ligne d'une relation, on utilise la commande **delete from**, comme dans l'exemple suivant :

```
DELETE
FROM        client
WHERE       num_cli = 10;
```

2.3.2.3.4. Mise à jour

Outre l'adjonction de nouvelles lignes, et la suppression de lignes existantes, SQL offre la possibilité de modifier des valeurs d'attributs de lignes existantes. On utitilise pour cela la commande **update** comme dans l'exemple suivant.

```
UPDATE      client
SET         adresse = 'Moulinsart'
WHERE       num-cli = 10;
```

Chapitre 3

Optimisation et Conception Logique

" D'un autre côté,
nous ne pouvons ignorer l'efficacité "

Jon Bentley

Ayant réalisé les schémas conceptuels des données et des traitements, il faut maintenant transformer ces schémas afin de les implémenter. La première étape dans la chaîne de transformation est la conception logique (voir 1.2).

On peut dire que la conception logique concerne **tout ce qui doit être connu par le programmeur d'application**, et rien d'autre[1]. Il doit par exemple connaître la structure exacte d'un type d'entité (quels sont ses attributs, leur nom, leur domaine de valeurs), ou encore les moyens de recherches mis à sa disposition (dans le cas du relationnel tous les moyens sont mis à sa disposition) mais pas la taille d'une page du SGBD ou le nom des fichiers dans lesquels seront stockées les données.

Cependant, le choix d'un design logique optimal (ou apportant de bonnes performances) de la BD sera parfois influencé par des facteurs comme le facteur de blocage[2], la présence ou l'absence d'un mécanisme d'accès aux données (index relationnel) ou encore l'éventuel tri des données, facteurs qui sont pourtant relatifs au niveau physique (voir notre informelle définition ci-dessus de ce qu'est un concept logique). Ceci nous fait dire que, **du**

---

1 Ceci peut en fait constituer un critère informel mais pratique pour séparer ce qui est logique de ce qui est physique.

2 Le facteur de blocage d'un TE est en fait le nombre d'entités d'un TE particulier contenu dans une page physique du SGBD. Ces pages sont les unités transférées par le SGBD entre les fichiers en mémoire secondaire et son buffer qui lui réside en grande partie en mémoire centrale. Ces pages sont des multiples (1, 2 ou 4) des pages du système de mémoire virtuelle de l'OS.

point de vue de l'optimisation, la frontière entre logique et physique est plus floue et difficilement respectable. C'est sans doute là où réside la première difficulté de l'optimisation "au niveau logique".

D'autre part, la conception logique des traitements se basant sur le schéma logique des données, il est important que ce schéma soit **"juste du premier coup"**. On ne pourra plus penser à des optimisations logiques de la BD une fois que les applications seront réalisées. Des transformations (d'optimisation) du schéma des données entraîneraient en effet de gros coups de maintenance. Ce n'est pas le cas pour les optimisations physiques. C'est en ceci que réside la seconde difficulté de l'optimisation logique.

Nous allons maintenant entamer la simplification du schéma conceptuel des données, premier sous-processus dans la phase de conception logique.

3.1. Transformation de simplification de schéma

Certains concepts de haut niveau et facilitant la conceptualisation ne sont présents dans aucun modèle logique (relationnel, hiérarchique ou réseau). Il faut donc les transformer en des concepts plus simples.

Le lecteur s'étonnera peut être de trouver ici l'élimination des relations many-to-many et de degré supérieur à deux souvent prise en compte dans la littérature ([BAT92] par exemple) au moment de la traduction du schéma dans un modèle spécifique, mais celles-ci n'étant conformes à aucun modèle, nous préférons la faire d'emblée.

3.1.1. La relation IS-A

Si un concept de modélisation de données de plus en plus utilisé et de haut niveau ne se trouve pas dans les modèles logiques comme le relationnel et le hiérarchique, c'est bien le concept de relation de généralisation/spécialisation, ou autrement dit la relation "IS-A".
Cette construction de haut niveau qu'est la relation "IS-A" est présente dans le modèle entité-association étendu, ainsi que dans l'outil de maintenance de base de données DB-MAIN. Elle est aussi présente dans le modèle logique des SGBD futurs et encore mal définis que sont les SGBD-OO.

Il faudra donc enlever cette construction pour se conformer à un modèle logique comme le rationnel.

Supposons que nous ayons le schéma de la figure-1.

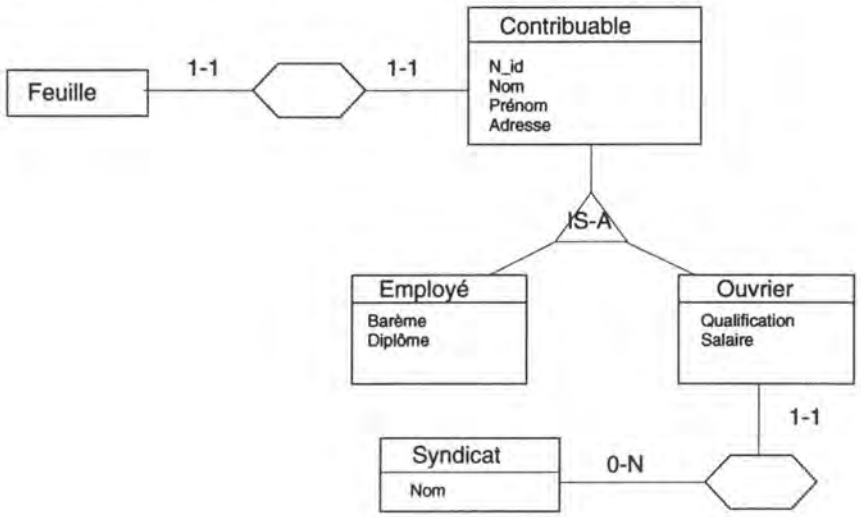


Figure-1 : *Un exemple de schéma de base de données possédant une relation IS-A.*

Si un contribuable est soit un employé ou un ouvrier, la relation de spécialisation est ce que l'on appelle une partition. Si un contribuable peut être à la fois un employé et un ouvrier (à mi-temps par exemple) et est au moins un employé ou un ouvrier, nous aurons ce que l'on appelle une couverture. Si par contre un contribuable n'est pas nécessairement un employé ou un ouvrier, le TE CONTRIBUABLE sera qualifié de TE "concret" (par opposition à "abstrait" c'est-à-dire toute entité de contribuable est une entité d'au moins une des spécialisations).

Il existe trois manières de transformer le schéma de la figure-1 en termes de relations et d'entités, chacune d'entre elles ayant ses avantages et inconvénients par rapport à un certain critère.

La première manière de faire est la plus directe, la solution par défaut dirons-nous. Sa représentation est d'ailleurs graphiquement très proche de celle du schéma comportant la relation IS-A. Tous les TE et TA restent inchangés, seules les relations IS-A entre chaque spécialisation (EMPLOYE et OUVRIER dans notre exemple) et le TE généralisé (CONTRIBUABLE) sont remplacées par des relations one-to-one, obligatoires pour chaque spécialisation, voir figure 2.

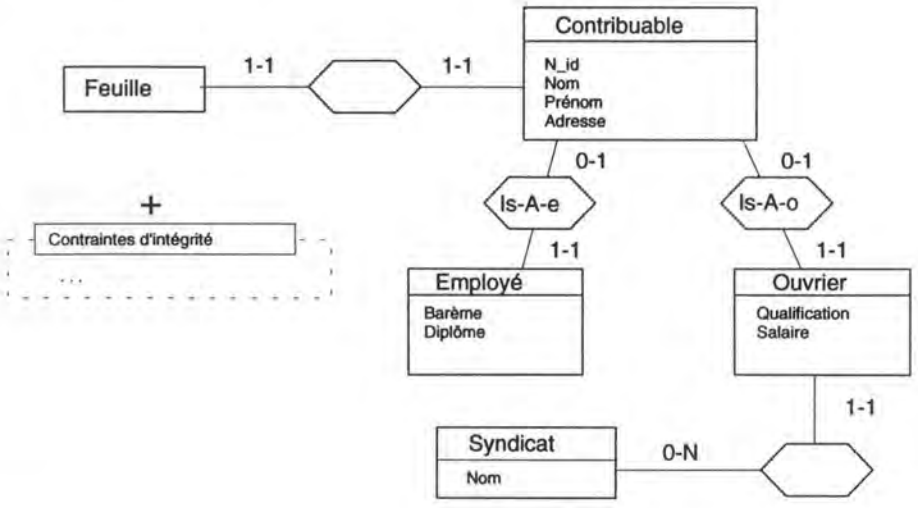


Figure-2 : *Première traduction de la relation de généralisation, la traduction considérée comme la plus directe.*

Des contraintes d'intégrité non gérables par la plupart des SGBD font leur apparition dans le schéma. Ce seront des contraintes sur les rôles joués par le TE généralisé (contribuable). On aura par exemple des contraintes d'exclusion de rôles pour les partitions ou, si le TE généralisé est un TE abstrait, il faudra s'assurer que ce TE joue au moins un rôle dans une des associations matérialisant les relations IS-A, etc...
La gestion de ces contraintes, qu'elle soit effectuée par le SGBD ou non, n'est évidemment pas gratuite et diminuera les performances lors des insertions par exemple.

Cette méthode aura un avantage sur les autres, elle supportera en effet facilement l'héritage multiple.

La seconde manière de traduire la relation de généralisation, comme la troisième d'ailleurs, peut être vue comme une traduction directe du schéma contenant la relation IS-A[3] ou, comme un schéma obtenu après quelques transformations plus élémentaires de la première solution, la solution par défaut (figure-2). Nous proposons cette seconde approche et considèrerons donc les deux autres traductions comme étant obtenues indirectement à l'aide de quelques transformations.
Cette approche (transformationnelle) nous permet de faire l'économie d'une comparaison des avantages et inconvénients de ces différentes traductions du point de vue des performances. On "héritera" en effet, des avantages et inconvénients apportés par les différentes transformations élémentaires combinées pour passer d'un schéma à l'autre. Les avantages et inconvénients de ces transformations élémentaires seront abordés plus tard.

La seconde traduction envisageable pour le schéma de la figure-1 est celle présentée à la figure-3. Elle est souvent reprise dans la littérature sous le nom de **traduction ascendante**.

---

3 C'est la manière d'envisager les choses que l'on rencontre dans la littérature et par exemple dans [BAT92]

Elle peut être obtenue à l'aide d'une transformation élémentaire à sémantique constante appliquée deux fois. Cette transformation est la fusion verticale (transformation inverse de la découpe verticale) présentée en 3.2.2.2. On peut voir à la figure-4 les applications successives de cette transformation.

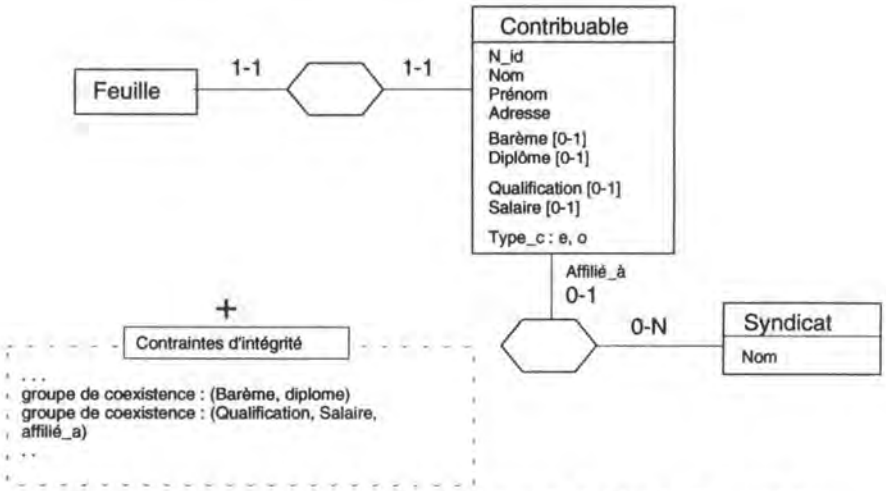


Figure-3 : *Seconde traduction de la relation IS-A à l'aide d'un seul TE.*

On peut évidemment ne faire qu'une seule de ces transformations élémentaires et ainsi profiter des avantages importants de celle-ci sans subir les désavantages de l'autre transformation. On le voit, l'approche utilisant les transformations élémentaires est plus riche. Répétons-le encore, les avantages et inconvénients de ces transformations seront étudiés plus tard.
On peut toutefois remarquer que la traduction proposée à la figure-3 nie l'intention introduite lors du design conceptuel de représenter la relation d'héritage. On ne voit en effet plus aussi distinctement (du moins graphiquement) le concept "IS-A". Ce concept avait déjà évidemment disparu lors de la traduction proposée à la figure-2, mais il était plus aisé de deviner que ces relations one-to-one étaient les vestiges d'une relation IS-A. Rappelons que les schémas des figures 2 et 3 sont sémantiquement équivalents.

Anticipons un peu sur les avantages et inconvénients que peuvent apporter les transformations de fusion en exprimant l'évaluation que l'on peut faire intuitivement de la traduction de la figure-3 :

- elle sera d'autant moins intéressante que le TE CONTRIBUABLE est abstrait (beaucoup d'entités ne sont ni des employés ni des ouvriers) et que la spécialisation s'approche d'une partition (une entité appartient à une et une seule spécialisation). Les entités de la classe contribuable contiendront alors d'autant plus d'attributs dont les valeurs seront égales à NULL avec tous les problèmes que cela comporte (voir 3.4. Tuning des requêtes.).

- Des contraintes appelées "groupe de coexistence" feront leur apparition. Ces groupes d'attributs et de rôles des spécialisations indiquent que tous leurs composants auront une valeur (peut-être NULL) ou tous auront une valeur NULL. Ces groupes de coexistence peuvent être transformés, matérialisés, en un attribut facultatif et composé des attributs de la spécialisation.

Ne nous y trompons, ces contraintes ne seront pas trop lourdes à gérer, et leurs vérifications se feront naturellement par l'application. Par contre, si on accepte les traitements

ad hoc, sur la base de données, il faudra ajouter des checks dont la gestion ne sera elle, pas gratuite.

On ajoutera enfin souvent un attribut "discriminant" indiquant la spécialisation à laquelle appartenait une certaine entité.

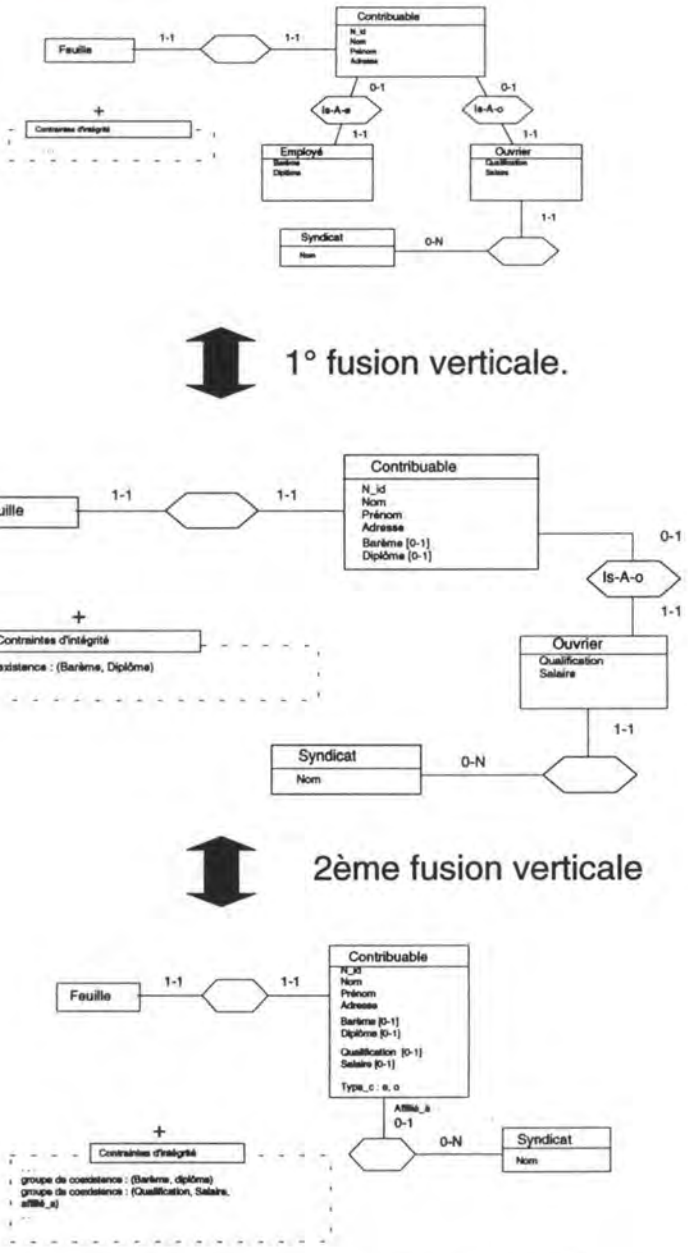


Figure-4 : *Transformation de la première traduction de la relation de spécialisation à l'aide de la fusion verticale appliquée deux fois.*

La troisième traduction est celle présentée à la figure-5. Elle est souvent reprise dans la littérature sous le nom de **traduction descendante**.

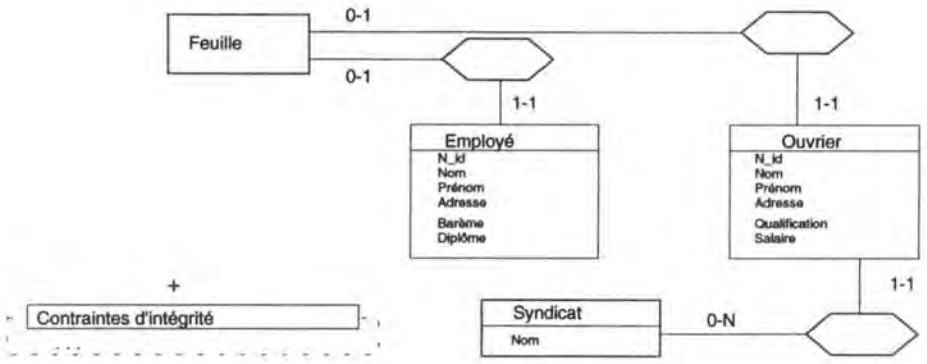


Figure-5 : *Traduction descendante de la relation de généralisation.*

Nous avons supposé pour cette traduction que le TE contribuable était un TE abstrait . Le schéma de la figure-5 ne pourrait en effet contenir sans "astuce" une entité qui ne serait ni un employé ni un ouvrier (un indépendant par exemple).
Une astuce consisterait par exemple à insérer les indépendants dans le TE EMPLOYE et à ne prendre en compte les informations spécifiques à EMPLOYE lorsque l'on a à faire avec un indépendant. Si du point de vue des performances on n'y voit pas d'inconvénients directs, l'interprétation du schéma n'en serait que plus hermétique et la maintenance du code plus difficile.
Une solution appropriée aux indépendants, serait de construire un TE spécifique à ceux-ci.

Cette traduction (figure-5) peut être obtenue à l'aide d'une découpe horizontale (présentée en 3.2.2.1) du TE contribuable suivant la spécialisation et de deux fusions verticales (trois transformations élémentaires). On peut voir l'application successive de ces transformations à la figure-6.

Une traduction dite descendante comme celle présentée à la figure-5, dans le cas d'une relation de spécialisation où les spécialisations ne sont pas disjointes (un contribuable peut être à la fois un employé et un ouvrier), amène de la redondance qu'il faudra gérer. Cette redondance touche les attributs et rôles du TE CONTRIBUABLE qui sont répétés dans les différentes spécialisations. On appelle cette duplication d'attributs et de rôles, de la redondance structurelle. Cette redondance sera étudiée en 3.2.3.. Il n'y aura cependant effectivement de la redondance que lorsque les ensembles d'entités des différentes spécialisations ne seront pas disjoints.

Dans le cas de la traduction présentée à la figure-5, si les spécialisations ne sont pas disjointes, il faudra par exemple lorsque l'on modifie l'adresse d'un ouvrier, modifier l'adresse de l'employé qu'est aussi cet ouvrier.

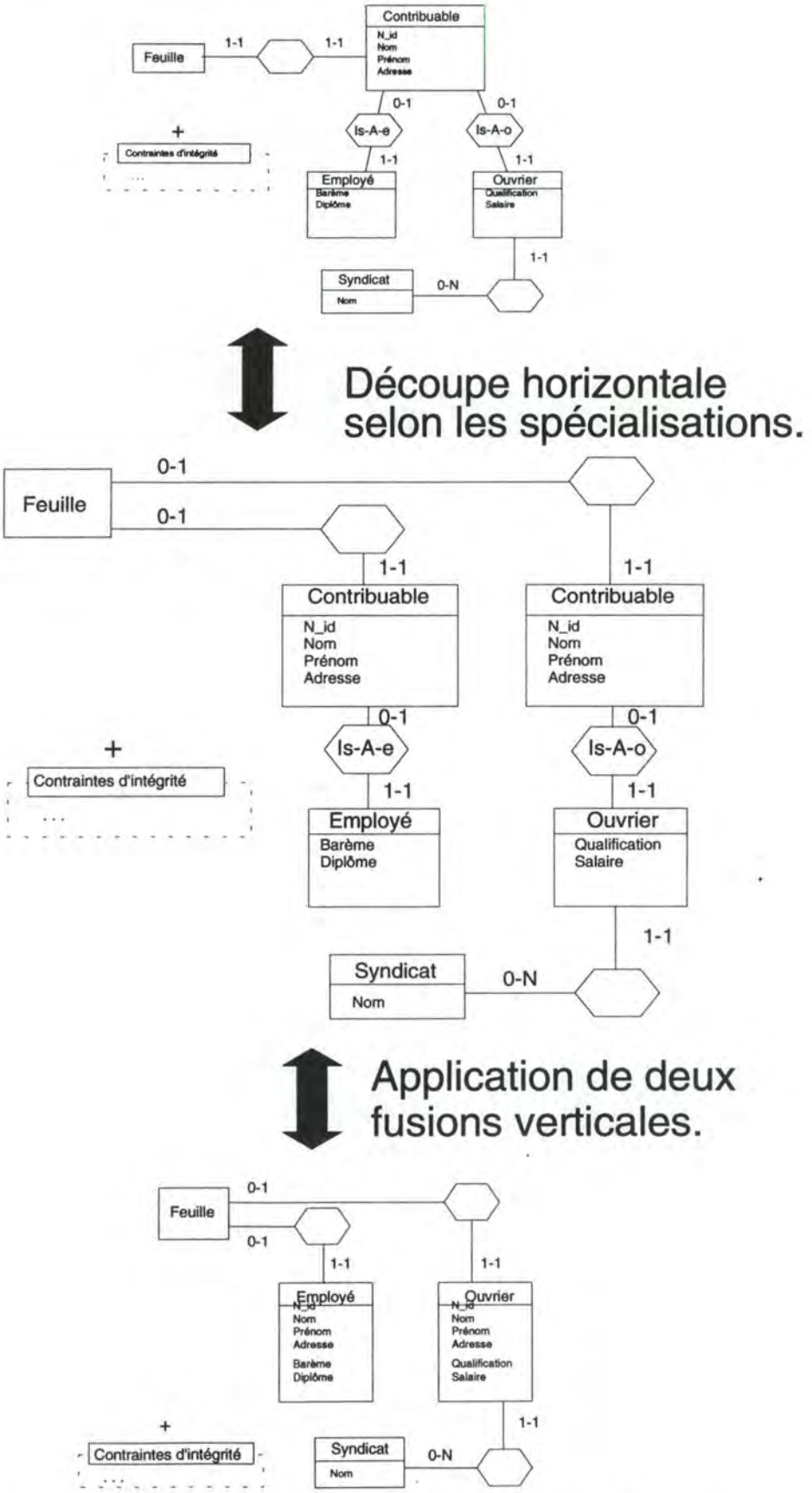


Figure-6 : *Transformation de la première traduction en la traduction descendante.*

3.1.2. Les relations many-to-many, de degré deux avec attributs et de degré supérieur à deux

Les associations many-to-many, de degré deux avec attribut et les associations de degré supérieur à deux doivent être éliminées au début du processus de conception logique.

En effet, de cette manière, on peut après une première élimination de ces concepts, optimiser le schéma de manière indépendante du SGBD.

Si on éliminait seulement ces concepts lors de la conformation à un modèle particulier, on passerait alors peut-être à côté d'optimisations indépendantes (Figure-7) des modèles, surtout **si l'on ne prévoit[4] pas de retour en arrière dans le processus de conception logique** présenté en 1.2.2. On pense par exemple à la dénormalisation ou à la fusion verticale que nous allons bientôt présenter. Voici les transformations d'élimination de ces concepts (Figure-7, et 8).

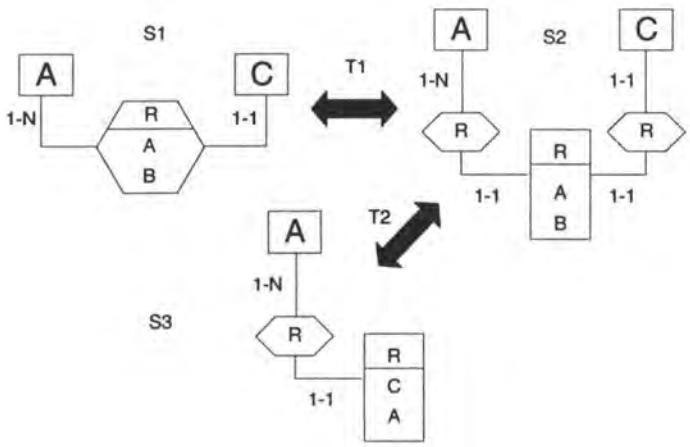


Figure-7 : *Elimination d'une association Many-to-One avec attributs : T1 S1-S2; et fusion verticale (T2 S2-S3) qui est une transformation d'optimisation indépendante.*

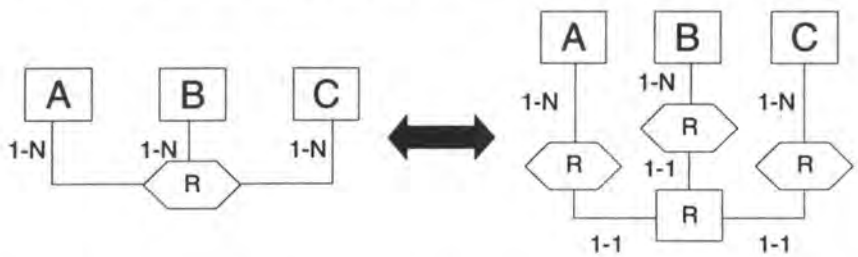


Figure-8 : *Elimination d'une association de degré supérieur à deux. On procèdera de la même manière pour l'élimination des associations binaires Many-to-Many*

4 Nul besoin de prévoir une méthode non linéaire si l'on élimine ces concepts non conformes dès le début.

3.2. Optimisations indépendantes du SGBD

Nous allons aborder dans cette section des techniques de transformation de schémas destinées à apporter une certaine optimisation quel que soit le type de SGBD.
Ces techniques de transformation de schémas sont destinées à augmenter les performances de la BD finale selon des critères aussi divers que le volume des données, le temps d'accès ou de mise à jour, le nombre de transactions qui peuvent s'exécuter simultanément (throughput) **sans toutefois sacrifier la simplicité d'exploitation ou la lisibilité du schéma.** Cependant, afin de pouvoir juger si les transformations présentées apportent une amélioration ou une dégradation par rapport à un certain critère, il nous faudra parfois faire intervenir des concepts simples mais appartenant au niveau physique comme le facteur de blocage.

Nous allons aborder ici les techniques de **dénormalisation**, de **découpes horizontale et verticale** d'un TE et d'**introduction de redondance.**

3.2.1. Dénormalisation

Considérons les deux schémas logiques de la figure-1. Ces schémas sont **sémantiquement équivalents**, on peut passer de l'un à l'autre par une transformation symétriquement réversible.

Le premier schéma est le siège d'une dépendance fonctionnelle[1] (DF) de *Num_fourn* vers *Ville_fourn*, il n'est donc pas sous 3FN. Par contre, le second est sous 3FN (et même sous (FNBC).

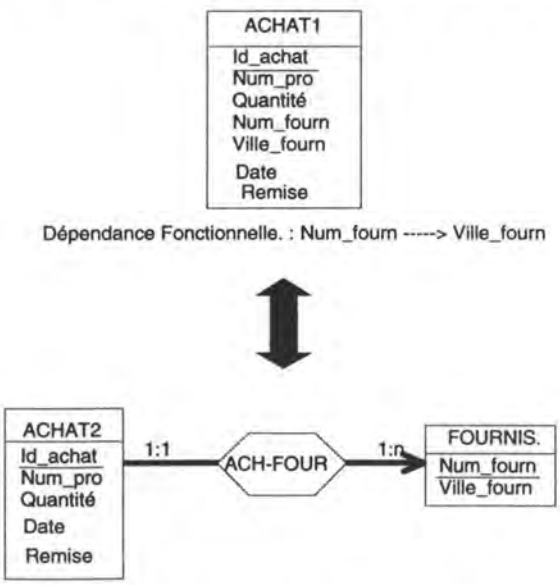


Figure-1 : *Transformation de normalisation/dénormalisation appliquée à un exemple. Le schéma du haut est dénormalisé*

1 De très nombreux articles et livres traitent des dépendances fonctionnelles et des différentes formes normales 3FN, FNBC, ... Le lecteur non familier avec ces notions essentielles consultera par exemple [BAT92] ou [DAT89]. Disons simplement que, dans notre exemple, chaque valeur de *Num_fourn* détermine fonctionnellement la valeur de *Ville_fourn*. Ajoutons que *Num_fourn* est dit être le déterminant et *Ville_fourn* le déterminé de la DF

Une extension possible du schéma dénormalisé serait :

ACHAT1						
Id_achat	**Num_pro**	**Quantité**	**Num_fourn**	**Ville_fourn**	**Remise**	**Date**
1	1	10	**1**	**Namur**	100	10/93
2	1	120	2	Ostende	10	11/93
3	2	15	**1**	**Namur**	0	1/94
4	1	10	3	Tournai	0	1/94
5	3	10	3	Tournai	100	1/94
6	2	100	**1**	**Namur**	1	2/94
7	3	18	**1**	**Namur**	10	2/94

Tab-1 : *Une extension possible du schéma dénormalisé.*

Si on se rapelle qu'une association entre deux TE peut être considérée comme un ensemble de couples [BOD89], on acceptera facilement l'extension suivante sous forme tabulaire[2] du schéma normalisé:

FOURNISSEUR	
Num_fourn	**Ville_fourn**
1	Namur
2	Ostende
3	Tournai

ACHAT2				
Id_achat	**Num_pro**	**Remise**	**Date**	**Quantité**
1	1	100	10/93	10
2	1	10	11/93	120
3	2	0	1/94	15
4	1	0	1/94	10
5	3	100	1/94	10
6	2	1	2/94	100
7	3	10	2/94	18

ACH-FOUR	
Id_achat	**Num_fourn**
1	1
2	2
3	1
4	3
5	3
6	1
7	1

Tab-2 : *Extension du schéma normalisé (3FN) contenant les mêmes informations que celles présentées dans Tab-1.*

On peut constater que dans l'extension du premier schéma, de l'information est répétée. Inutilement? Nous verrons. On retrouve, en effet, plusieurs fois dans l'extension le fait "*le fournisseur 1 habite Namur*" (en gras dans Tab-1).

2 Celle-ci ne préjugeant en rien d'une quelconque traduction dans le modèle relationnel.

Nous allons comparer ces schémas suivant trois critères, l'**espace consommé**, la **préservation de l'intégrité de l'information** et enfin les **temps d'exécution de différents traitements** et tirer les conclusions qui s'imposent. Mais voyons tout d'abord l'expression plus formelle de cette transformation de dénormalisation.

3.2.1.1. La transformation de (dé)normalisation

La dénormalisation consiste à fusionner deux TE entre lesquels existe une association binaire one-to-many. La normalisation consiste quant à elle à extraire une DF présente dans un TE et à faire de celle-ci un autre TE relié au premier par une association one-to-many.

Du point de vue de l'optimisation du schéma logique de la BD, **c'est la transformation de dénormalisation qui va principalement nous intéresser** .

Le schéma conceptuel issu de l'analyse fonctionnelle est en effet déjà très souvent normalisé (3FN).

La normalisation du schéma lors de l'analyse fonctionnelle est très souvent utilisée comme outil de vérification de la complétude et de la cohérence des spécifications et facilite l'intégration et **la compréhenssion du schéma**. Ce dernier point reste une question de goût.

Nous supposerons donc que le schéma sur lequel nous allons travailler est normalisé et que les optimisations éventuelles consistent à dénormaliser le schéma.

Dans le cas contraire (schéma conceptuel non-normalisé), il faudra alors envisager la transformation de normalisation comme transformation d'optimisation. Or, si aucun ordre n'était à respecter dans la prise en compte des dénormalisations, il n'en sera pas de même pour les normalisations. On ne pourra, en effet pas, extraire les dépendances fonctionnelles dans n'importe quel ordre sans voir apparaître des DF qui "s'étendent" sur plusieurs TE. On peut trouver dans [FAG77] ou [BER76] des algorithmes de mise sous troisième forme normale.

La transformation de (dé)normalisation peut alors s'exprimer comme suit :

Soit le type d'entité E1 :

$$\mathbf{E1\ (\ e_1,...,\ e_n,\ a_1,\ ...,\ a_i,\ ...,\ a_j,\ a_k,\ ...,\ a_l,\ ...,\ a_m)\ où}$$

Id (strict) **(E1) = e_1,..., e_n et E1 est le siège d'une DF : a_i, ...,a_j $\rightarrow$ a_k, ..., a_l.**

On peut alors transformer ce schéma en un seul des schémas ci-dessous par une transformation symétriquement réversible, on fait alors une **normalisation**. Nous allons différencier trois cas, celui où ni le déterminant ni le déterminé de la DF n'ont d'intersection avec le groupe d'attributs et/ou rôles composant l'identifiant (cas 1), celui où le déterminant a une intersection avec l'identifiant (cas 2) et enfin celui où le déterminé a une intersection avec l'identifiant (cas 3). Le cas où déterminé et déterminant ont une intersection avec l'identifiant n'est pas pertinent vu que celui-ci nierait l'hypothèse statuant le caractère strict de l'identifiant.

- Cas 1 : *Si* $\{e_1,..., e_n\} \cap \{a_i, ..., a_j, a_k, ..., a_l\} = \varnothing$ (Voir exemple figure-2), on aura :

E11 $(e_1,..., e_n, ..., a_{i-1}, a_{l+1}, ..., a_m)$ **où Id (E11) =** $\{e_1,..., e_n\}$
E12 $(a_i, ..., a_j, a_k, ..., a_l)$ **où Id (E12) =** $\{a_i, ..., a_j\}$
ASS $(r_1$ **1-1 : E11,** r_2 **1-n : E12).**

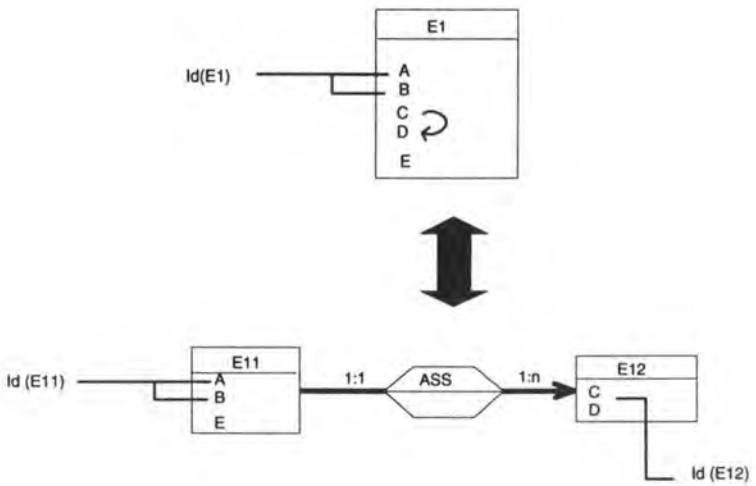


Figure-2 : *Une instanciation de la transformation ci-dessus avec n =2,* $\{e_1, e_2\}$ = $\{A, B\}$, $i = j$, $a_i = C$, $k = l$, $a_k = D$. *Il y a apparition d'un chemin d'accès de E11 vers E12 vu que dans le premier schéma, il était possible étant donné A et B de connaître C ou D.*

- Cas 2 : *Si* $\{e_1,..., e_n\} \cap \{a_i, ..., a_j\} = \{e_u, ..., e_v) =$
$\{a_r, ..., a_s\}$ *et* $\{e_1,..., e_n\} \neq \{a_i, ..., a_j\}$
c'est-à-dire le déterminant de la DF contient une partie seulement de l'identifiant et peut-être des attributs extérieurs à l'identifiant..
et $\{e_1,..., e_n\} \cap \{a_k, ..., a_l\} = \varnothing$ (Voir exemple figure-3), on aura :

E12 $(a_i, ..., a_r, ..., a_s,..., a_j, a_k, ..., a_l)$ **où Id (E12) =** $a_i, ..., a_r, ..., a_s,..., a_j$
E11 $(e_1,...,e_{u-1}, e_{v+1}, e_n, ..., a_{i-1}, a_{l+1}, ..., a_m)$
Id (E11) = $\{e_1, ..., e_{u-1}, e_{v+1}, ... e_n,$ E11.ASS.E12.a_r, ..., E11.ASS.E12.$a_s\}$
ASS $(r_1$ **1-1 : E11,** r_2 **1-n : E12).**

L'identifiant de E11 est composé d'attributs "éloignés", il faut en effet suivre un chemin pour les trouver. Cependant, si $\{a_i, ...,a_j\} \subset \{e_1,..., e_n\}$ on peut alors simplifier cet identifiant en le remplaçant par :

Id (E11) = $\{e_1, ..., e_{u-1}, e_{v+1}, ... e_n, r_2\}$

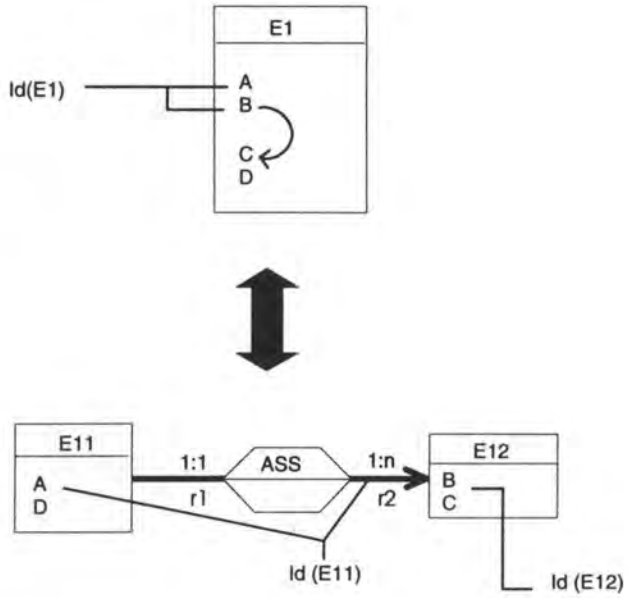


Figure-3 : *Un exemple de (dé)normalisation dans laquelle le déterminant de la DF contient une partie de l'identifiant et rien d'autre.*
$n = 2$, $\{e_1, e_2\} = \{A, B\}$, $i = j$, $a_i = B = e_u$, $k = l$, $a_k = C$.

• Cas 3 : *Si $\{e_1,..., e_n\} \cap \{a_k, ..., a_l\} = \{e_u, ..., e_v) = \{a_r, ..., a_s\}$. (On a $\{e_1,..., e_n\} \cap \{a_i, ..., a_j\} = \varnothing$)*
c'est-à-dire le déterminé de la DF contient une partie seulement de l'identifiant et peut-être des attributs extérieurs à l'identifiant, (voir exemple figure-4) on aura :

E12 (a_i, ..., a_j, a_k, ..., a_r, ..., a_s a_l) où Id (E12) = a_i, ..., a_j
E11 (e_1,...,e_{u-1}, e_{v+1}, e_n, ..., a_{i-1}, a_{l+1}, ..., a_m)
Id (E11) = {e_1, ..., e_{u-1}, e_{v+1}, ... e_n, E11.ASS.E12.a_r, ..., E11.ASS.E12.a_s}
ASS (r_1 1-1 : E11, r_2 1-n : E12).

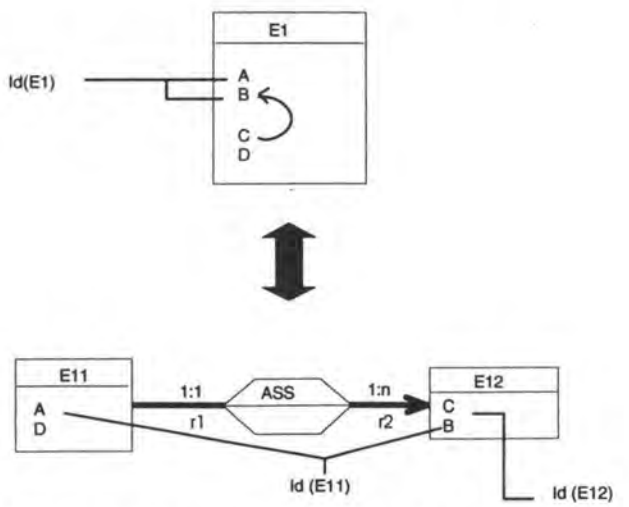


Figure-4 : *Un exemple de (dé)normalisation dans laquelle une partie du déterminé de la DF appartient à l'identifiant. On peut y voir l'identifiant de E11 composé d'attributs "éloignés".*

Remarquons encore que e_1, ..., e_n, a_1, ..., a_m peuvent être aussi bien des attributs du TE siège de la DF, que des rôles joués par des TE dans le cadre de TA auxquels participe notre TE. On peut en voir une illustration à la Figure-4

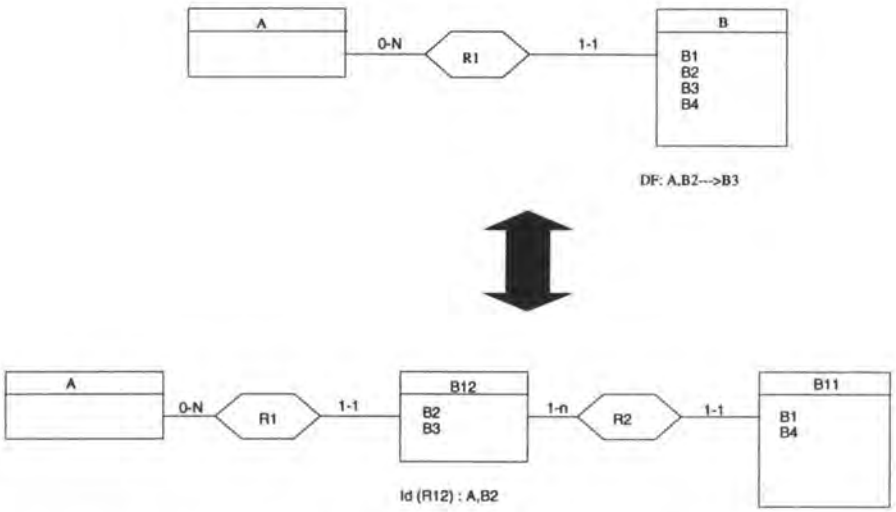


Figure-5 : *Elimination d'une DF dont un des composants du déterminant est un rôle (du haut vers le bas).*

La normalisation entraîne dans le dernier cas (une partie de l'identifiant appartient au déterminé de la DF que l'on essaie d'éliminer) et parfois dans le second (quand on ne peut pas procéder à la simplification de l'identifiant) une complexification de la contrainte d'intégrité que constitue l'identifiant de E1, autrement dit du type d'entité que l'on normalise.

Il faudrait gérer ces identifiants composés d'attributs éloignés à l'aide de l'application lors de chaque opération d'insertion, ceux-ci n'étant pas conformes aux modèles relationnel ou CODASYL. Cette gestion serait très lourde pour l'application et son coût rédhibitoire. On envisagera alors un abandon de sémantique si la perte de cette contrainte d'identification paraît peu importante et si la gestion de cette contrainte entraîne des coûts sans rapport avec le bénéfice de leur conservation. En outre, une vérification de masse permettrait de détecter aisément les anomalies.

Les schémas normalisé et dénormalisé de la figure-4 ne devraient cependant pas se rencontrer trop souvent. Ceux sont en effet des schémas résultats typiques d'une approche bottom-up (par synthèse) [FAG77]. L'approche bottom-up a le mérite de parfois relever des dépendances fonctionnelles que l'analyse fonctionnelle (que nous supposons avoir été utilisée) n'avait pas perçues. Cependant, si l'on veut gérer ces DF cela coûtera très cher lors des insertions et il serait alors plus efficace de dénormaliser même[3] si l'on considère le coût des insertions !

Ce genre d'identifiant se représentera lors des découpes verticales. Nous évaluerons le coût d'insertions lorsqu'il faut gérer ce genre d'identifiant en 3.2.2.

3 Nous verrons la signification de ce "même"

3.2.1.2. Espace consommé

Il est difficile de parler de consommation d'espace au niveau logique, les notions de "byte", "Mbyte", codage de différents types de valeur (un entier est-il codé sur 4 ou 5 bytes ?) appartiennent en effet au niveau physique. Pourtant, les décisions prises ici ((dé)-normalisation) auront des conséquences sur l'espace consommé par la base de données.

On peut toutefois, en prenant quelques hypothèses raisonnables[4], se persuader que **le schéma dénormalisé consomme plus de place** quand il est implanté de manière classique[5] dans des SGBD relationnels ou réseaux, que le schéma normalisé.

Prenons les schémas et leur extension (figure-1 et Tab-1) et supposons qu'un caractère consomme une unité d'espace (U ou byte), qu'un nombre en consomme quatre. Supposons encore que l'on ait décidé lors du design logique que *Ville_fourn* ait une longeur de 25 caractères. L'implantation du schéma dénormalisé coûterait à peu près la même chose sur un SGBD-R ou un SGBD CODASYL, c-est-à-dire 6 x 4U + 25U = 49 U pour chaque entité de ACHAT1, ce qui fera pour l'extension Tab-1 : 7 x 49U = **343** U. Nous parlons ici de la place consommée par les entités sans prendre en compte certains "overhead" dus au SGBD considéré, ni des concepts purement physiques comme le taux de remplissage des pages.

L'implantation du schéma normalisé sur un SGBD-R, se fera *classiquement* à l'aide de deux tables et d'une Foreign Key (FK) qui se substitue à l'association. Nous verrons d'autres implantations à la section 3.3.1. Le lecteur non familier avec la traduction d'une association one-to-many dans les SGBD-R se réfèrera à cette section.
ACHAT2 contient la Foreign Key qui est du même type que l'identifiant de la table FOURNISSEUR. On consommera alors (5+1) x 4U = 24 U pour chaque entité de ACHAT2 et 4U + 25U = 29U pour chaque entité de FOURNISSEUR. Comme il y a 7 entités ACHAT2 et 3 entités FOURNISSEUR (voir Tab-2), on consommera en tout 255U.

Dans un SGBD CODASYL[6], l'implantation classique se fera à l'aide de deux *record-type* (traduction des TE) et d'un *set-type* (traduction de l'association). Si de plus on suppose un double chaînage des *members* du *set-type*, et que l'on évalue la taille d'un pointeur (pour le chaînage) à 4U, chaque entité ACHAT2 consommera (5+3(chaînages)) x 4U = 32U et chaque entité FOURNISSEUR, 4+25+4(chaînage)U = 33U, ce qui fait un total de 7 x 32 + 3 x 33 = 323U.

On voit donc que le schéma non-normalisé consomme d'ores et déjà plus de place quel que soit le type de SGBD considéré.

La différence de consommation d'espace ne ferait que s'amplifier si le nombre d'achat augmentait, si du moins on accepte l'hypothèse réaliste que l'on ne fait pas chaque achat chez

---

4 Nous entendons par *raisonnable* que ces hypothèses ne s'écartent pas trop de ce qu'on peut observer dans la plupart des SGBD commercialisés.

5 *De manière classique*, c'est-à-dire comme le ferait de prime abord un analyste à la vue du schéma sans se poser trop de questions.

6 Le lecteur non familier aux concepts CODASYL peut se référer à *OLLE W The CODASYL approach to data base management Wiley & son 78.* ou encore à *J.L. HAINAUT Introduction aux systèmes de gestion de base de données CODASYL 71, Institut d'informatique. Namur fév, 88.*

un nouveau fournisseur, ce qui aurait pu se représenter[7] par une relation one-to-one entre FOURNISSEUR et ACHAT2. Il n'y aurait alors plus de dépendance fonctionnelle dérangeante vu que *Num_fourn* deviendrait lui aussi un identifiant de ACHAT au même titre que *Id_achat*. Nous tomberions dans le problème abordé à la section suivante[8].

La consommation supplémentaire d'espace est due au fait que certains faits sont représentés plus d'une fois. Il y a redondance. Le fait que le fournisseur 1 habite Namur est, par exemple, nous l'avons déjà dit, représenté quatre fois dans l'extension du premier schéma Tab-1.

La dénormalisation fait donc perdre de la place.

On peut estimer la différence de consommation à :

(F1) $\left(\sum_{\text{déterminant}} (N_{\text{E/déterminant}} - 1) \times T_{DF} \right) - \varepsilon(N_E, \text{SGBD, choix de design})$,

où $\mathbf{N_{E/déterminant}}$ est le nombre d'entités du TE siège de la DF qui ont la même valeur de déterminant,

$\mathbf{T_{DF}}$ est la place nécessaire au stockage du déterminant et du déterminé de la DF, et

ε est une fontion de N_E (nombre d'entités), du SGBD et de différents choix de design physique donnant la place consommée pour l'implantation de l'**association**.

Autrement dit, ayant en face de soi un schéma normalisé, comme le schéma du bas de la figure-2, on peut affirmer, que si l'on dénormalise, le schéma résultant fera consommer d'autant plus de place que les entités de E12 ont une taille importante (T_{E12}) et que le nombre moyen d'associations auquel participe chaque entité de E12 est élevé (noté $\mu_{\mathbf{ASS\ E12}}$).

(F1) peut donc être réécrite ainsi :

$$\mathbf{(\mu_{ASS\ E12} * N_{E12} - 1) * T_{E12} - \varepsilon(N_{E11}, SGBD, choix\ de\ design)}$$

Pour notre exemple, cela donne : ((4-1) + (2 -1) + (1-1)) x (25 + 4) - ε = 116-ε.

Comme on le voit, la différence de consommation de place des schémas dénormalisé et normalisé donnée par la formule (F1) est croissante en $N_{E/déterminant}$, (autrement dit en la sélectivité moyenne de la clé que peut constituer le "déterminant de la DF") et en T_{DF}. Elle est par contre décroissante en la place nécessaire au stockage des associations (une par entité).

Or, comme on le verra, certaines implémentations des associations seront plus coûteuses en place de stockage.

---

7 Lorsque l'on remplace une relation 1-N par une 1-1, la sémantique n'est évidemment pas conservée, même si au niveau des instances de la relation, cela est vérifié pour l'instant.

8 Ce problème de normalisation/dénormalisation est intrinsèquement différent du problème de découpe verticale abordé dans la section suivante. Une des différences est le type de relation unissant les fragments : 1-N pour la normalisation, 1-1 pour la découpe verticale.

3.2.1.3. Préservation de l'intégrité de l'information

Un schéma dénormalisé ou non-normalisé peut entraîner des problèmes connus sous les noms d'**anomalies** de modification, de suppression et d'insertion si on n'y prend garde. Par anomalie on entend état inconsistant de la BD obtenue après un certain type d'opérations. Prendre garde à ces anomalies n'est évidement pas gratuit...

- La modification de *Ville_fourn* dans **une** entité de ACHAT1 entraîne la violation de la dépendance fonctionnelle si il existe une autre entité ayant même valeur pour *Ville_fourn*. Un traitement modifiant l'attribut *Ville_fourn* de l'entité x (nouvelle valeur = α) et respectant la DF aura alors la forme (Figure--5) :

$$\forall y \in \text{ACHAT1 tq } y.\text{Num\_fourn} = x.\text{Num\_fourn} \Rightarrow y.\text{Ville\_fourn} := \alpha$$

Figure-5 : *Formulation "prédicative" d'un traitement modifiant le déterminé (d'une DF) d'une entité x et respectant cette DF.*

Sans cette dépendance fonctionnelle, c'est-à-dire avec le schéma normalisé, la modification est beaucoup plus simple : une seule entité à modifier.

Le code de mise à jour sera parfois plus complexe[9] et l'exécution du traitement de mise à jour sera ralentie. Il faut en effet propager la modification. Cette propagation peut-être prise en charge par certains SGBD ou par le programme d'application. Mais dans les deux cas, il y aura un surcoût à payer.

Lorsque l'on sait que l'opération de mise à jour est la plus coûteuse parmi les opérations simples proposées par SQL[10], on voit qu'il faudra utiliser prudemment la dénormalisation dans un environnement où les mises à jour sont fréquentes.

Il faut toutefois faire remarquer que les mises à jour ne sont particulièrement gênantes dans un schéma dénormalisé, que lorsqu'elles touchent une partie ou la totalité du **déterminé** (comme *Ville_fourn*) de la DF. La mise à jour de *Date* ne posera en effet pas de problèmes dans aucun des schémas de la Figure-1.

On peut dire que le coût d'un traitement modifiant le déterminé d'une DF et respectant la DF, sera d'autant plus élevé que $\mathbf{N_{E/déterminant}}$ est important. On peut tirer cette conclusion de la version généralisée du traitement de la figure-5 proposé à la figure-6.

$$\forall y \in TE \text{ } tq \text{ } y.\text{déterminant} = x.\text{déterminant} \Rightarrow y.\text{déterminé} := \alpha$$

Figure-6 : *Formulation "prédicative" d'un traitement modifiant le déterminé (d'une DF) d'une entité x de TE et respectant cette DF.*

Il faudra en effet modifier $\mathbf{N_{E/déterminant}}$ entités[11] du TE au lieu d'une seule pour une modification touchant un attribut qui n'est pas le déterminé d'une DF ou d'une seule modification dans le cas du schéma normalisé.

---

9 Si le code du traitement effectuant les mises à jour est plus complexe, sa maintenance sera peut-être plus coûteuse. Le code de ce traitement ne sera cependant pas plus complexe dans l'un ou l'autre cas si l'on utilise SQL, seule l'exécution du traitement sera plus lente.

10 Dans DB2 d'IBM, si on évalue le coût d'une insertion à 1, le coût d'une modification est alors de 1.75 .

Autrement dit, ayant en face de soi un schéma normalisé, comme le schéma du bas de la figure-2, on évitera de le dénormaliser si des traitements doivent mettre à jour les valeurs d'attributs de E12 et ce d'autant plus que ces mises à jour seront fréquentes et que le nombre moyen d'associations auquel participe chaque entité de E12 est élevé ($\mu_{ASS\ E12}$).

- Lors des suppressions, il peut y avoir perte d'information dans un schéma dénormalisé. En effet si on supprime l'entité de ACHAT1 dont l'*Id_achat* = 2, on perd toute trace du fournisseur de *Num_fourn* = 2 (voir tab-1). Si la disparition de cette information n'est pas souhaitée, il faudra la stocker dans par exemple, un TE FOURN_SANS_ACHAT séparé.

Par contre, avec le schéma normalisé de la Figure-1, un simple relâchement de la connectivité 1-N remplacé par 0-N, permet de conserver très facilement les fournisseurs sans achat. Conceptuellement, c'est d'ailleurs sans doute le schéma avec une connectivité minimale égale à 0 qui aura été choisi.

Ce relâchement de connectivité allège les contraintes, il y a donc moins de choses à vérifier. Cependant, ce relâchement rend impossible de la transformation de dénormalisation[12]. Les deux schémas ne sont alors plus sémantiquement équivalents. Il est toutefois plus facile de relâcher cette contrainte que de créer un nouveau TE afin de contenir les fournisseurs sans achat.

- Lors des insertions dans le schéma dénormalisé de la Figure-1, il faudra que l'opérateur ou l'application vérifie préalablement la valeur de *Ville_fourn* associée à *Num_fourn* afin qu'elle soit la même que celles qui sont associées aux mêmes *Num_fourn* déjà présents dans la BD. On ne pourra en effet pas insérer un nouvel achat dans ACHAT1 avec comme valeur (..., 2, Tournai, ...) pour (..., *Num_fourn, Ville_fourn,* ...) alors que le fournisseur 2 habite Ostende, et que l'on a toujours notre DF. Cette vérification engendrera des coûts supplémentaires.

Avec le schéma normalisé il suffirait d'associer le nouvel achat à un FOURNISSEUR existant sans obligatoirement connaître sa valeur de *Ville_fourn*.

Le surcoût lors des insertions (vérifications supplémentaires) ne doit pas être négligé, il peut être très important.

En SQL par exemple, il suffit avec le schéma normalisé, d'utiliser l'opération EXIST pour savoir si un fournisseur ayant un certain *Num_fourn* existe.

Cette opération est en effet implémentée de telle sorte que la recherche s'arrête dès que le SGBD est logiquement en mesure de répondre à la question. Il sera déjà logiquement en mesure d'y répondre en consultant par exemple uniquement un index comme nous sommes capables de dire que Namur existe sans savoir où, à la seule lecture de l'index d'un atlas.

Par contre, avec le schéma dénormalisé, il faudra rechercher la valeur de *Ville_fourn* **des** achats ayant une certaine valeur de *Num_fourn*. Pour cela on aura dû sélectionner tous les achats ayant ce *Num_fourn* à l'aide du SELECT, ce qui est beaucoup plus coûteux.

La dénormalisation ou la non-normalisation entraîne donc des surcoûts non-négligeables de maintenance et de vérification lors des insertions, et mises à jour.

Le coût de modification du déterminé d'une DF croît avec $N_{E/déterminant}$.

---

11 On suppose bien entendu que le traitement de modification ne concernait au départ qu'une seule entité même dans un schéma sans DF.

12 Il faudrait pour pouvoir dénormaliser, tout en conservant la sémantique, faire une découpe horizontale (voir 3.2.2.) préalable des entités de TE FOURNISSEUR suivant le prédicat P = "est associé à au moins un achat".

3.2.1.4. Temps d'exécution de différents traitements de sélection

Analysée suivant les deux précédents critères, la dénormalisation ne semble pas être un mécanisme d'optimisation, au contraire, elle consomme plus de place et peut être la source d'anomalies de suppression, de modification et d'insertion qui sont évitées en assumant un surcoût de vérification des DF.

La dénormalisation n'a en fait qu'un seul avantage : les performances des consultations en terme du nombre d'accès.

Même si nous n'allons parler que des traitements de sélection, il faut remarquer qu'il n'y a pas que ces derniers qui effectuent une sélection. Les suppressions et les mises à jour effectuent elles aussi une recherche préalable (voir 2.3.2.3).

3.2.1.4.1. Nombre d'accès logiques

Nous allons comparer les coûts en accès logiques[13] nécessaires à l'exécution de différents traitements s'exécutant sur les schémas normalisé et dénormalisé de la Figure-7.

3.2.1.4.1.1. Un traitement effectuant une "traversée" dans le sens One-to-Many.

Supposons que l'on veuille effectuer le traitement suivant (Trt-1) : *Sélectionner tous les Num_pro des ACHATs faits chez des fournisseurs habitant Namur.*

Supposons encore que l'on ait 100.000 ACHAT1 (ou 100.000 entités dans le TE ACHAT2; ils ont en effet le même nombre d'entités) répartis uniformément[14] chez 1000 FOURNISSEURs répartis eux aussi uniformément sur 20 villes.

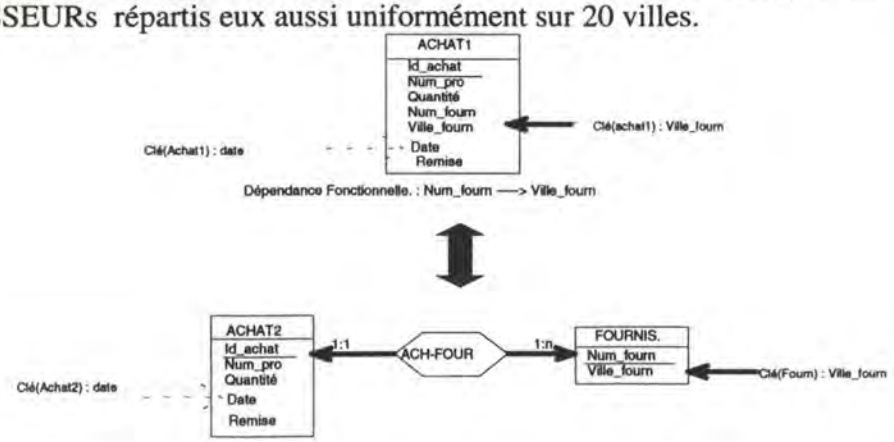


Figure-7 *: Schéma normalisé et dénormalisé repris de la figure-1 et enrichi de différentes clés et chemins d'accès.*

13 Un accès logique est un accès à une entité. Chaque fois que l'on doit lire l'information d'une entité, il y a accès logique.

14 La distribution est supposée uniforme pour faciliter les calculs. On pourrait considérer d'autres distributions mais il est peu probable que l'on puisse obtenir ces informations avant que la BD soit implémentée.

Dans le cas du schéma non-normalisé (ACHAT1), il faudra, 100.000 / 20 = 5000 accès logiques (on suppose que l'**on utilise la clé sur *Ville_fourn*** et que son utilisation est gratuite, niveau logique oblige).

Dans le cas du schéma normalisé (Figure-7 en bas), une des exécutions possibles du traitement est la suivante : d'abord sélectionner les *Num_fourn* des FOURNISSEUR tels que FOURNISSEUR.*Ville_fourn* = 'Namur'. Cela fait 1000 / 20 = 50 accès (utilisation de la clé d'accès *Ville_fourn* supposée). Puis, chercher les *Num_pro* de ACHAT2 ayant un de ces *Num_fourn* (parcours du chemin d'accès dans le sens one-to-many). Comme 50 *Num_fourn* ont été sélectionnés, on accèdera à 50 x 100.000/1000 = 5000 entités de ACHAT2, ce qui nous fait un total de 5050 accès logiques.

En passant du "normalisé" au "dénormalisé", on ne semble donc pas à la vue de cet exemple constater de grandes différences au niveau du nombre d'accès logiques nécessaires à la réalisation de Trt-1.

Cependant, voyons ce qui se passe si nous avons toujours le schéma de la figure-7 mais avec les statistiques suivantes (diminution de $N_{E/valeur\ du\ déterminant}$) :
100.000 ACHAT1 (ou ACHAT2), 10.000 valeurs de *Num_fourn,* et toujours 20 valeurs de *Ville_fourn.*

Pour le schéma dénormalisé, le coût en accès logiques de Trt-1 est toujours de 100.000/20 = 5000.

Pour le schéma normalisé, on comptabilise : 10.000/20 = 500 ajoutés à 500 * (100.000 / 10.000) = 5000, ce qui fait un total de 5500 accès logiques. La différence est déjà plus substantielle.

Avec la diminution du $N_{E/valeur\ du\ déterminant}$, les performances du schéma normalisé se dégradent donc de plus en plus.

En effet, le coût logique de Trt-1 sur le schéma normalisé de la figure-7 est plus formellement exprimé par :

$$C_{Norm}^{Trt-1} = \frac{N_{FOURNISSEUR}}{N_{Ville\_fourn}} + \frac{N_{FOURNISSEUR}}{N_{Ville\_fourn}} \times \frac{N_{ACAHT2}}{N_{FOURNISSEUR}}$$

Le coût logique de Trt-1 sur le schéma dénormalisé est lui de :

$$C_{Denor}^{Trt-1} = \frac{N_{ACAHT1}}{N_{Ville\_fourn}}$$

Comme $N_{ACHAT1} = N_{ACHAHT2}$, la différence de coût entre les deux schémas est de $N_{FOURNISSEUR} / N_{Ville\_fourn}$.

Or, $N_{FOURNISSEUR} = N_{ACHAT1} / \mu_{ACH\text{-}FOUR\ FOURNISSEUR}$, donc la différence de coût peut être exprimée comme suit :

$$\Delta_{Denor-norm}^{Trt-1} = \frac{N_{ACHATI}}{\mu_{ACH\text{-}FOUR\ FOURNISSEUR} \times N_{Ville\_fourn}}$$

Si l'on considère le coût d'exécution de Trt-1 avec le schéma dénormalisé comme l'étalon, le surcoût à payer pour effectuer Trt-1 avec le schéma normalisé est donc de :

$$\overline{\Delta}^{\text{Trt-1}}_{\text{Denor-norm}} = \frac{1}{\mu_{\text{ACH-FOUR FOURNISSEUR}}}$$

$\mu_{\text{ACH-FOUR FOURNISSEUR}}$[15] étant égal à $N_{\text{ACHAT1/Num\_fourn}}$[16] qui est le nombre d'entités par valeur de déterminant de la DF, une diminution de ce dernier fait augmenter le surcoût.

$\mu_{\text{ACH-FOUR FOURNISSEUR}}$ étant borné inférieurement par la cardinalité minimale du rôle joué par FOURNISSEUR dans ACH-FOUR et égale à 1, on aura $\overline{\Delta}^{\text{Trt-1}}_{\text{Denor-norm}} \leq 1$. On aura un coût logique double avec le schéma dénormalisé si et seulement si $\overline{\Delta}^{\text{Trt-1}}_{\text{Denor-norm}} = 1$ c'est-à-dire si la relation one-to-many est du point de vue des instances, une relation one-to-one.

Remarquons encore trois choses :

- Avec ce genre de traitement, le schéma normalisé sera plus coûteux,
- Le surcoût est calculé en moyenne, mais il est de toute façon supérieur à zéro (connectivité maximale du rôle < ∞), et
- Si la cardinalité minimale du rôle joué par FOURNISSEUR n'est pas 1 mais 0 (comme suggéré par exemple pour la sauvegarde des FOURNISSEUR sans ACHAT), le surcoût pourra dépasser une fois le coût occasionné par Trt-1 avec le schéma dénormalisé ! Ce coût logique pourrait encore être amplifié au niveau physique. Pour en effet se rendre compte que le FOURNISSEUR sélectionné n'est associé à aucun achat, on consommera encore parfois plusieurs accès physiques[17].

3.2.1.4.1.2. Un traitement effectuant une "traversée" dans le sens Many-to-One

Supposons que l'on veuille maintenant effectuer le traitement suivant : *Sélectionner tous les Num_pro des achats faits chez des fournisseurs habitant Namur à une date précise* (Trt-2).

Supposons aussi que l'on ajoute une clé d'accès sur *Date* (figure-7 clé en pointillé) dont on imaginerait raisonnablement la sélectivité[18] à 100/100.000.

On comptabiliserait alors 100 accès logiques dans le cas du schéma non-normalisé. On accède en effet d'abord aux entités de ACHAT1 à l'aide de la clé *date* et on ne retient ensuite que les entités de ACHAT1 vérifiant ACHAT1.*Ville_fourn* = 'Namur'. Un mauvais plan d'exécution de Trt-2 serait de se servir de la clé *Ville_fourn* et ce même avec le schéma normalisé.

---

15 Le nombre moyen d'associations du TA ACH-FOUR auquel participe chaque entité du TE FOURNISSEUR.

16 Nombre moyen d'entités du TE ACHAT1 ayant même valeur pour *Ville_fourn.*

17 Ces accès physiques inutiles sont caractéristiques des SGBD relationnels. Ils sont absents des SGBD CODASYL ou OO. Nous reparlerons de ceci à la section 3.3.4.

18 Rappel : la sélectivité est le pourcentage d'entités qui vérifient un certain critère de sélection.

Comme on peut s'y attendre, avec le schéma normalisé, le coût logique d'exécution de Trt-2 sera beaucoup plus grand, le chemin d'accès étant traversé ici dans le sens Many-to-One. On accèdera en effet toujours à 100 entités de ACHAT2 en utilisant la clé *date,* mais on devra encore, **pour chacune de ces entités**, rechercher l'entité de FOURNISSEUR qui lui est associée afin de vérifier que cette dernière ait 'Namur' comme valeur de *Ville_fourn.* La recherche de l'entité associée à chaque entité de ACHAT2 sélectionnée se fait via le chemin d'accès les unissant.

Cela fait passer le coût logique d'exécution du simple au double.

3.2.1.4.1.3. Un traitement n'accédant qu'au TE du côté Many

Dans ce cas, le calcul du coût du traitement en accès logiques donne le même résultat, qu'il soit appliqué au schéma normalisé ou dénormalisé.

Soit Tr-3 : *donner le Num_pro, la quantité de toutes les entités du TE ACHAT1.* Ce traitement séquentiel coûtera $N_{ACHAT1} = N_{ACHAT2}$ accès logiques.

3.2.1.4.1.4. Accès logiques : synthèse

La dénormalisation améliore les performances des traitements de sélection en terme d'accès (logiques), c'est un résultat bien connu.

Soit le schéma dénormalisé suivant :

E1 (e_1,..., e_n, a_1, ..., a_i, ..., a_j, a_k, ..., a_l, ..., a_m) où
E1 est le siège d'une DF : a_i, ...,a_j $\rightarrow$ a_k, ..., a_l.

et son équivalent normalisé :

E11 (e_1,..., e_n, ..., a_{i-1}, a_{l+1}, ..., a_m)
E12 (a_i, ..., a_j, a_k, ..., a_l) où Id (E12) = a_i, ..., a_j
ASS (r_1 1-1 : E11, r_2 1-n : E12).

- Lorsqu'une traversée de l'association ASS dans le sens one-to-many **via le chemin d'accès** allant de E12 vers E11 doit être envisagée pour réaliser un traitement, le coût en accès logiques de ce traitement pourrait être diminué en adoptant le schéma dénormalisé, et ce, d'autant plus que $\mu_{ASS\ E12}$ (borné inférieurement par 1) est petit. Si $\mu_{ASS\ E12}$ =1, le coût en accès logiques sera alors même réduit de moitié. Si $\mu_{ASS\ E12}$ devient grand, le surcoût n'est pas proportionnellement important.

- Par contre, lorsqu'une traversée de l'association ASS dans le sens many-to-one via le chemin d'accès allant de E11 vers E12 doit être envisagée pour réaliser un traitement, le coût en accès logiques de ce traitement pourrait être diminué de moitié en adoptant le schéma dénormalisé.

- Les traitements ne devant accéder qu'à E11 ne profiteraient pas de la dénormalisation, le coût logique étant identique dans les deux cas.

Pour savoir si, **un** traitement donné est plus performant avec un schéma normalisé ou avec un schéma dénormalisé, autrement dit, pour savoir si on doit dénormaliser le schéma pour rendre plus performant un certain traitement, il suffirait donc logiquement de

- Savoir choisir les plans d'exécution[19] optimaux pour chaque schéma et ensuite de
- Comparer ces coûts en accès logiques pour trancher.

Pour savoir quel schéma est le meilleur **pour l'ensemble des traitements** (sélection insertion,...), il "suffirait" donc de minimiser la somme des coûts logiques des différents traitements pondérés par leurs fréquences d'activation et/ou leur relative priorité pour les différents schémas logiques.

En ne considérant que les accès logiques, le coût d'un schéma normalisé augmente uniquement avec les insertions et les mises à jour du déterminé de la DF. En leur absence et si la place de stockage n'est pas limitée, on dénormalisera. On le fera sans hésitation d'autant plus que pour les traitements de sélection, le schéma dénormalisé n'est pas plus difficilement interprétable pour le programmeur que ne l'est le schéma normalisé.

Cependant, les choses se compliquent très vite lorsque l'on considère le coût non plus en accès logiques mais en accès physiques, il faut par exemple tenir compte des coûts d'utilisation et de maintenance des clés d'accès.

Il faut pourtant bien considérer les coûts en accès physiques, car en pratique, ceux sont eux qui sanctionnent les performances d'une application.

Un facteur physique que nous prendrons très souvent en considération (nous l'avons dit au début de la section 3.2) est le facteur de blocage (FB).

Il est maintenant temps de donner des approximations des coûts physiques d'utilisation d'une clé d'accès, d'une clé d'accès lorsque les entités sont triées sur les valeurs de cette clé (accès semi-séquentiel) et de l'accès séquentiel pour retrouver k entités parmi N groupées par page de FB entités.

Accès séquentiel k =1	Accès séquentiel k > 1	Accès par clé k >=1	Accès par clé triée K >=1 (semi- séquentiel)
$\left\lceil \frac{\frac{N}{FB}+1}{2} \right\rceil$	$\left\lceil \frac{N}{FB} \right\rceil$	Idxsecond + k	Idxprim + $\left\lceil \frac{k}{FB} \right\rceil$

Tab-3 : *Approximations des coûts physiques des accès séquentiels, par clé, et semi-séquentiels, sans tenir compte de la concurrence, des buffers, mais uniquement du Facteur de Blocage.*

Nous pourrons en tirer quelques enseignements intéressants.

19 On consultera [HAI86] pour une étude du choix de la meilleure stratégie d'exécution d'un traitement (basée les coûts en accès logiques) appliqué à un schéma logique. Dans [HAI86] on envisage la remise en cause du choix de stratégie lors du design physique.

3.2.1.4.2. Nombre d'accès physique.

Il n'existe malheureusement pas de relation simple entre le nombre d'accès physiques et le nombre d'accès logiques nécessaires à la réalisation d'un traitement.

D'un côté, les accès logiques supplémentaires nécessaires à la réalisation de Trt-1 ou Trt-2 avec le schéma normalisé de la figure-7 (le double du nombre d'accès logiques nécessaires à sa réalisation avec le schéma dénormalisé pour Trt-2) ne se traduiront par exemple pas toujours chacun par un accès physique.

Le SGBD saura en effet peut-être profiter du **facteur de blocage[20] en diminution et supérieur à 1**, comme il était supposé pour les accès logiques, du **buffer**[21], ou encore de **l'ordre des entités** dans le fichier, etc...

D'un autre côté, un accès logique nécessitera souvent plusieurs accès physiques dus à l'utilisation de mécanismes d'accès tels que les index ou encore dus à la concurrence de divers processus..

Sans vouloir rentrer dans les détails, mettons en évidence certains aspects à côté desquels on passe lorsque l'on ne considère que les accès logiques pour prendre des décisions à propos du design logique, et en particulier lors de la décision de dénormalisation.

3.2.1.4.2.1. Accès séquentiel au TE du côté Many

Soit Tr-4 : *donner le Num_pro, la quantité de **toutes** les entités du TE ACHAT*, ce traitement ne nécessite pas l'utilisation de clés ou chemins d'accès lorsqu'il est effectué sur le schéma normalisé de la figure-7, c'est en effet un traitement séquentiel typique (sélectivité = 1). Son exécution sera alors plus rapide si l'on dispose du schéma normalisé.

Il y a en effet, autant d'entités dans ACHAT1 ou dans ACHAT2, **mais la taille d'une entité de ACHAT1 est supérieure à celle d'une entité de ACHAT2.** T_{ACHAT1} est égale à $T_{ACHAT2} + T_{FOURNISSEUR}$.

Toute autre chose égale par ailleurs (taille de page, taux de remplissage, ...), il faudra plus de pages pour contenir toutes les entités de ACHAT1. Il y aura en effet plus d'entités de ACHAT2 par page physique (facteur de blocage plus élevé) que d'entités de ACHAT1.

Or, toutes les entités doivent être lues donc toutes les pages accédées. Dans ces conditions, la lecture séquentielle[22] de tout le TE ACHAT1 consommera donc plus d'accès physiques que la lecture séquentielle de ACHAT2.

En un seul accès physique, "on ramènera en effet en MC plus d'entités de ACHAT2 que d'entités de ACHAT1".

---

20 Son augmentation fait diminuer le nombre d'accès physiques nécessaires à l'utilisation d'une clé et surtout des lectures séquentielles (voir formule coût physique de l'accès séquentiel Tab-3).

21 Le buffer est une zone de la MC sous le contrôle du SGBD dans lequel ce dernier stocke des informations récemment utilisées en espérant, à la manière du système de pagination de l'OS, économiser un accès physique au support secondaire de stockage, qui est beaucoup plus coûteux.

22 C'est-à-dire lire toutes les entités les unes après les autres dans l'ordre physique et donc, une page après l'autre.

Ce phénomène sera d'autant plus important que $T_{DF} = T_{FOURNISSEUR}$ sera grande. Le rapport des coûts est en effet à peu près de :

$$\frac{C_{Trt-4\ Denor}}{C_{Trt-4\ Norma}} = \frac{T_{ACHAT2} + T_{FOURNISSEUR}}{T_{ACHAT2}}$$

Cette augmentation du facteur de blocage influençant les performances des accès séquentiels, interviendra aussi dans le choix des découpes verticales éventuelles mais de manière plus importante si du moins on ne considère que les performances et adopte une approche que nous qualifierons de brutale.

On peut en effet dire que l'augmentation du facteur de blocage n'est qu'un "effet de bord" (parfois très important) de la normalisation, son but premier étant d'éliminer la redondance et d'éviter les anomalies.

Si un traitement ne nécessite[23] ni l'utilisation d'une clé d'accès ni l'utilisation de chemin d'accès **lorsqu'il est effectué avec le schéma normalisé**[24], alors il restera plus efficace avec le schéma normalisé si du moins on ne décide pas de faire une recherche semi-séquentielle sur le schéma dénormalisé alors qu'elle n'était que séquentielle, sur le schéma normalisé.

Si la sélectivité du traitement est égale à 1, alors il sera, sans condition, plus efficace avec le schéma normalisé.

3.2.1.4.2.2. Le TE du côté One ne contient que peu d'entités

Si le TE du côté One (FOURNISSEUR dans notre exemple) est tel qu'il a beaucoup de chance de rester constamment dans le buffer, donc si le nombre d'entités ainsi que leur taille sont restreints ou plus exactement si le nombre de pages contenant le TE est petit, alors le coût physique d'exécution des traitements avec les schémas normalisé ou dénormalisé seront sensiblement les mêmes.

Dans le cas du schéma normalisé, la politique de remplacement des pages (par exemple last recently used) du buffer fera peut-être en sorte que le TE FOURNISSEUR reste, lors de l'exécution de Trt-2, constamment dans le buffer du SGBD. De ce fait, seul un accès logique au TE FOURNISSEUR (un sur le nombre d'entité de ACHAT2 sélectionnées) se traduira par un accès physique au disque : le premier. Dès que la page contenant le TE FOURNISSEUR sera ramenée dans le buffer, seuls les accès au TE ACHAT2 occasionneront des accès physiques au support secondaire.

---

23 On ne peut pas juger de la nécessité d'utilisation d'une clé d'accès à la seule lumière des coûts en accès logiques. Le coût logique d'un accès séquentiel est en effet toujours supérieur au coût logique d'un accès par clé. Si par contre on connaît les coûts exacts (pour un SGBD particulier dans un environnement particulier) d'un accès séquentiel et d'un accès par clé, on pourra déterminer le "moment" où l'accès par clé devient plus cher. Ce moment sera déterminé principalement en fonction de la sélectivité et du FB. On peut lire dans les manuels du Tuning des SGBD que la sélectivité autour de laquelle se situe le breack-even-point est à peu près de 15 %

24 Il se peut qu'un traitement ne nécessitant pas l'utilisation d'une clé d'accès lorsqu'il est exécuté sur le schéma normalisé, nécessite l'utilisation d'une clé lorsque l'on dénormalise le schéma, le break-even-point étant dépassé .

La différence en nombre d'accès physiques entre les exécutions de Trt-2 sur les schémas normalisé et dénormalisé sera donc très faible et ce d'autant plus que le buffer sera grand et que la probabilité P que la page de FOURNISSEUR que l'on vient de ramener contienne l'entité de FOURNISSEUR qui est associée à l'entité suivante de ACHAT2 est proche de 1.

$$P = \frac{FB_{FOURNISSEUR} \times \mu_{ACH\text{-}FOURN\ FOURNISSEUR}}{N_{ACHAT2}}$$

Rappelons qu'avec le schéma normalisé, l'exécution de Trt-2 coûtait le double en accès logiques de ce qu'elle coûtait avec le schéma dénormalisé !

De plus, comme nous l'avons dit au point précédent, le facteur de blocage du TE ACHAT2 sera supérieur à celui du TE ACHAT1. Si l'on ne prend en compte que l'approximation du coût physique des accès par clé (non triée) donnée dans Tab-3, on ne constatera pas de différence de coût des accès par clé quel que soit le facteur de blocage (FB).

Cependant, si l'on examine la formule[25] donnant la probabilité de devoir accéder à une page du fichier contenant un TE, pour un FB donné et une sélectivité S fixée lorsque la distribution des entités suivant leur valeur de clé dans les pages ne peut être supposée qu'uniforme.

$1-(1-S)^{FB}$,

on voit que le nombre minimum de pages (du fichier contenant le TE) auxquelles il faut accéder augmente (mais faiblement) quand FB diminue. Ceci laisse présager que les performances des accès par clé pourraient aussi se détériorer avec la diminution du facteur de blocage.

Ce facteur de blocage a cependant largement moins d'influence sur les coûts des accès par clé (nombre d'accès physiques) que sur les coûts des accès séquentiels surtout si la sélectivité est faible. Cela dit, on voit toutes les "nuances" que la considération des coûts en accès physiques apporte, et ce n'est pas fini.

3.2.1.4.2.3. Un traitement n'accédant qu'au TE du côté One

Supposons que l'on ait le traitement suivant (Trt-5) : *trouver la ville du fournisseur de numéro égal à 10.*

Si on a le schéma normalisé de la figure-7, enrichi d'une clé d'accès sur *Num_fourn*[26], l'exécution de Trt-5 sera très peu onéreuse.

---

25 S est la probabilité d'accéder à une ligne, $(1-S)^{FB}$ est donc la probabilité de ne pas devoir accéder à une page. Quand FB tend vers 1, on obtient l'approximation donnée dans tab-3.

26 En relationnel, si l'on veut faire gérer la contrainte d'intégrité d'identification par le SGBD, il y aura une clé d'accès sur *Num_fourn* : PRIMARY KEY ou UNIQUE.

Par contre, avec le schéma dénormalisé, à moins que l'on choisisse de prendre *Num_fourn* comme clé de tri (ce qui paraît assez improbable et plus coûteux en maintenance[27]), le coût d'exécution de Trt-5 sera plus élevé.

En effet, si on a le schéma dénormalisé, il faudrait écrire en SQL la requête suivante pour implémenter Trt-5 :

```
SELECT DISTINST   Ville_fourn
FROM              ACHAT1
WHERE             Num_fourn = 10 ;
```

Or, comme on le verra dans la partie consacrée au tuning des requêtes 3.4), l'utilisation de DISTINCT (ici nécessaire) se paie cher. De plus, pour effectuer cette requête, on a dû au moins lire toutes[28] les entités vérifiant *Num_fourn* = 10, alors que l'on pouvait se contenter de lire une seule entité de FOURNISSEUR lorsque le traitement était effectué sur le schéma dénormalisé.

Le coût sera d'autant plus important avec le schéma dénormalisé pour réaliser Trt-5 que $\mu_{ASS\ FOURNISSEUR}$ est élevé.

Si ce genre de traitement est fréquent, mais que pour l'ensemble des autres traitements l'on préfèrerait le schéma dénormalisé, alors, on envisagera le schéma logique présenté à la figure-8 :

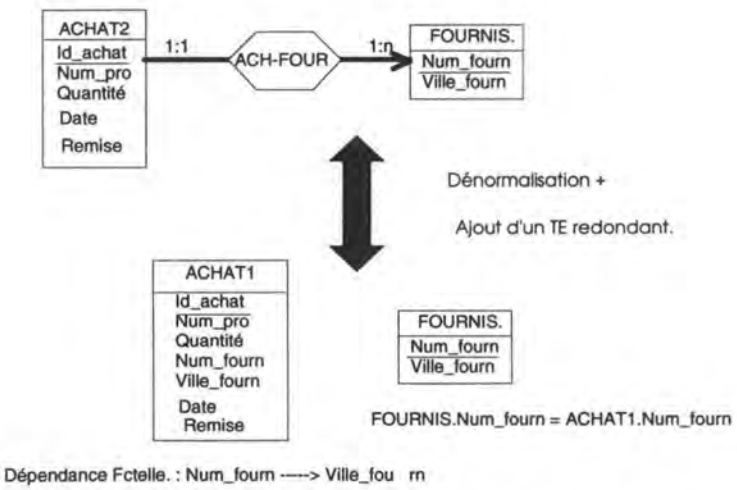


Figure-8 : *Après avoir dénormalisé, on peut ajouter un TE redondant pour optimiser les traitements utilisant uniquement des composants de la DF.*

On a ajouté au schéma, après avoir dénormalisé, un TE structurellement redondant[29].

---

27 On se rend intuitivement compte que de garder les entités dans un certain ordre est plus coûteux lors des insertions ou des mises à jour par exemple. On peut trouver dans [SCH85] une étude mathématique des coûts de maintenance lors des insertions et mises à jour dans les SGBR-R

28 L'optimiseur ne peut s'arrêter après avoir trouvé la première occurrence vérifiant le prédicat, il ne sait en effet pas qu'il existe une DF, celle-ci n'étant pas supportée par les SGBD-R.

29 Nous étudierons la redondance structurelle en 3.2.3.

Il faut cependant remarquer que ce problème est plus grave pour les SGBD-R que pour les SGBD CODASYL. Avec ces derniers, on pourra en effet s'arrêter après avoir trouvé dans le schéma dénormalisé la première entité ayant un *Num_fourn* = 10.

La recherche de cette première entité sera cependant plus lente que si l'on ne devait accéder qu'au TE FOURNISSEUR, et ce de part l'absence probable d'un index sur *Num_fourn* et le nombre d'entité de ACHAT1 plus important que celui de FOURNISSEUR.

3 .2.1.4.2.4.Lecture séquentielle et en parallèle de deux TE

Nous savons que le coût logique de réalisation de Trt-2 est plus important si il est effectué sur le schéma normalisé de la figure-7 et que celui de Trt-1 est à peu près identique avec les deux schémas. Nous savons aussi que si le TE FOURNISSEUR peut résider dans le buffer durant l'exécution du traitement, les coûts physiques sont sensiblement les mêmes.

Mais quels seront les résultats des courses pour l'exécution de Trt-1 si l'on a les schémas[30] normalisés et dénormalisés de la figure-9, traductions immédiates des schémas de la figure-1 et enrichis de clés d'accès et de tri apportant les meilleures performances possibles pour l'exécution de Trt-1 avec chaque schéma.

Nous supposons en fait que les optimisations physiques (choix des clés, de l'ordre de tri des articles, ...) ont été réalisées pour chacune des propositions de schéma logique de la figure-9 pour effectuer le traitement Trt-1. Ce genre de design physique est typique d'une situation où le nombre d'entités de ACHAT2 auxquelles il faut accéder est très important, la sélectivité du critère de sélection "ACHAT2.*Num-fourn = FOURNISSEUR.Num_Fourn tel que FOURNISSEUR.Ville_fourn = 'Namur'*" n'étant pas très importante mais plutôt proche de 1 (comme par exemple si il n'y avait que deux valeurs de *Ville_fourn* différentes).

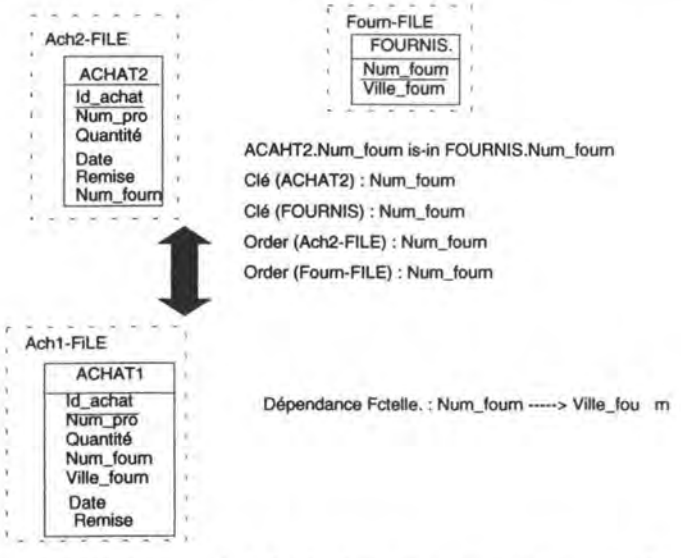


Figure-9 *: Schémas optimisés physiquement pour exécuter Trt-2.*

Si le schéma dénormalisé a été optimisé physiquement de la sorte, c'est que la lecture séquentielle de ACHAT1 s'avérait au moins aussi performante que l'utilisation d'un index sur

30 Ces schémas sont conformes aux SGBD relationnel, CODASYL et aux Systèmes de Gestion de Fichiers COBOL.

Ville_*fourn*. De même, si le schéma normalisé a été optimisé physiquement comme représenté à la figure-9 c'est sans doute que le programmeur CODASYL ou COBOL ou l'optimiseur du SGBD-R avaient l'intention de lire les deux fichiers séquentiellement et en parallèle. Ils s'étaient sans doute rendu compte que le nombre de pages nécessaires à la réalisation de Trt-1 avec le schéma normalisé de la figure-9 était inférieur au nombre de pages nécessaires à la réalisation de ce même traitement avec le schéma dénormalisé.

En effet, même[31] si on ajoutait une clé de tri *Ville-Fourn* sur ACHAT1, le nombre de pages N1 à lire dans Ach1-FILE serait de :

$$N1 = \left\lceil \frac{S \times N_{ACHAT1}}{FB_{ACHAT1}} \right\rceil \text{ ou S est la sélectivité du critère "}\textit{Ville\_fourn}\text{ = 'Namur' ",}$$

alors que le nombre de pages N2 à lire dans Ach2-FILE et Fourn-FILE pour effectuer Trt-1 serait égal à :

$$N2 = \left\lceil \frac{N_{ACHAT2}}{FB_{ACHAT2}} \right\rceil + \left\lceil \frac{N_{ACHAT2}}{m_{ACH-FOUR\ FOUNISSEUR} \times FB_{FOURNISSEUR}} \right\rceil$$

Or, N1 sera supérieur à N2 quand S sera assez faible. Ceci s'explique par le fait que lorsque l'on lit une nouvelle entité de ACHAT2, la probabilité que l'on ait déjà en MC l'information concernant le fournisseur qui lui est associée est proche de 1, vu que les deux fichiers lus séquentiellement sont triés sur la même clé. C'est-à-peu près le même phénomène qui se produisait quand le TE était de petite taille (voir 3.2.1.4.2.2.).

On peut s'en persuader en observant le schéma de la figure-10. Le volume A + B est le volume (convertible en nombre de pages) lu pour l'exécution de trt-1 avec le schéma normalisé de la figure-9 (les deux TE sont triés sur la même vakeur de clé). Le volume C = A + B représente le volume d'entités à partir duquel il est certainement[32] plus performant d'utiliser le schéma normalisé de la figure-9.

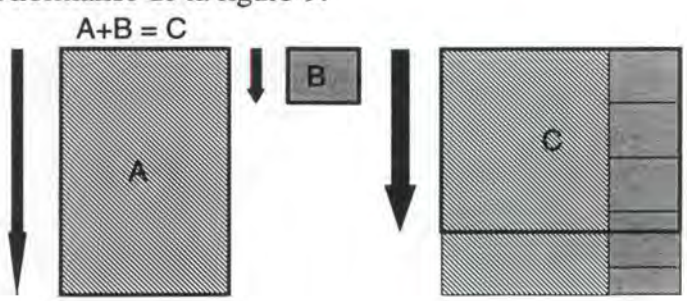


Figure-10 : *Volume de données transféré plus important (à droite) dans le cas de l'exécution de Trt-2 avec le schéma dénormalisé de la figure-9. Volume tranféré lors de la mise en oeuvre de la lecture séquentielle et parallèle = A + B.*

31 Insistons sur le fait que l'ajout d'une clé de tri sur *Num_fourn* durant l'optimisation physique ne se justifierait pas si la sélectivité du critère de Trt-1 était proche de 1. Or nous avons supposé que l'on avait optimisé physiquement le schéma logique normalisé de la figure-9, et cet optimisation physique ne se justifie pour l'exécution de Trt-1 que si le critère a une sélectivité proche de 1.

32 Car nous avons supposé ici que toutes les entités de ACHAT1 étaient dans des pages contiguës, ne nécessitant pas la lecture séquentielle de tout le TE ACHAT1.

3.2.1.4.2.5. Coût de compilation des requêtes

Nous ferons une dernière remarque au sujet de l'exécution des traitements :
dans le cas particulier des SGBD-R, la navigation inter-tables engendrée par la normalisation (et par d'autres techniques d'ailleurs), augmente la consommation des ressources CPU et Mémoire Centrale. La compilation de ces requêtes peut ainsi nécessiter des millions d'instructions.

De plus, les requêtes multi-tables sont souvent plus difficilement compréhensibles par les programmeurs.

On ne tiendra cependant plus compte de ces coûts pour des requêtes dont le temps d'exécution est proportionnellement beaucoup plus important ou lorsque le résultat de la compilation a été sauvé[33] (save query plan) comme cela peut être le cas pour ORACLE..

33 Il faut alors faire attention au fait que l'optimiseur n'entre plus en fonction même lorsque l'évolution des extensions des relations justifierait une autre stratégie d'exécution.

3.2.1.5. Comment savoir si il faut dénormaliser[34]

Si du point de vue des accès logiques, le schéma dénormalisé était toujours plus performant pour les traitements de sélection, nous avons vu qu'il ne l'était plus dans certains cas de figure si l'on considérait les accès physiques.

Pour les comparaisons des avantages et inconvénients (place de stockage, anomalies, efficacité des traitements de sélection) des schémas normalisés et dénormalisé, plusieurs facteurs entrent en ligne de compte :

- T_{DF}, la taille de la DF ou autrement dit la taille des entités de E12 et
- $N_{Entité\ /\ valeur\ du\ déterminant}$, c'est-à-dire le nombre d'entités du TE comprenant la DF ayant en moyenne la même valeur pour le déterminant de la DF, soit $\mu_{ASS\ E12}$.
- Les sélectivités des critères de sélection des traitements.

Après tout ce que nous avons dit, on serait donc tenter de :

Préconiser la **normalisation du schéma comme solution par défaut**[35] et de dénormaliser uniquement si les conditions suivantes tiennent :

1. Le volume des données n'est pas à minimiser[36],
2. Peu d'insertions[37] et de mises à jour[38] sont envisagées,
3. Peu de traitements doivent accéder uniquement à E12 (séquentiellement ou par clé),
4. Beaucoup de traitements de sélection **doivent franchir** l'association dans le sens many-to-one.
   La condition sera plus forte si ils doivent le faire pour vérifier une condition[39] que si ils doivent le faire pour uniquement récupérer de l'information.

---

34 Nous reprenons ici les notations de l'encadré page 3.23.

35 Il n'y a donc rien à faire vu que nous supposons le schéma déjà normalisé (voir début de section).

36 D'autant plus que $N_{Entité\ /\ valeur\ du\ déterminant}$ et T_{DF} (autrement dit $\mu_{ASS\ E12}$ et T_{E12}) sont grands

37 A moins que le schéma normalisé soit le siège d'un identifiant composé d'attributs "éloignés", ce qui nécessite alors de très lourdes vérifications lors des insertions.

38 Uniquement si ces mises à jour touchent des attributs de E12 et ce d'autant plus que $N_{Entité\ /\ valeur\ du\ déterminant}$ est grand.

39 Trt-2 devait encore vérifier si l'achat était fait chez un fournisseur habitant Namur. Dans ce cas, pour chaque entité de ACHAT2 on était obligé d'accèder à une entité de FOURNISSEUR. Si il fallait uniquement récupérer de l'information, comme par exemple si on avait le traitement : *sélectionner les Num_pro, Num_fourn et Vill_fourn des achats faits à une certaine date*, il ne fallait alors accéder à une entité de FOURNISSEUR que si la condition sur *date* (la clé d'accès) était vérifiée et si les autres éventuels critères de sélection évaluables étaient vérifiés. Dans le cas où la traversée est entreprise pour uniquement faire une vérifiaction, on envisagera alors d'introduire de la redondance structurelle dans le schéma normalisé au lieu de dénormaliser (voir 3.2.4)

La condition sera moins forte si E12 ne "risque" de séjourner totalement ou presque dans le buffer.

5. Peu de lectures en parallèle des deux TE triés sur la même clé sont envisagées.

6. Peu de franchissements dans le sens one-to-many quand $\mu_{ASS\ E12}$ s'approche de 1 sont envisagés. En effet, dans ce cas, le gain apporté par la dénormalisation est semblable à celui apporté pour les traversées dans le sens Many-to-One.

7. Peu de traitements séquentiels ou semi-séquentiels n'accédant qu'à E11 lorsqu'ils sont exécutés sur le schéma normalisé sont envisagés.

Seules les conditions 1 à 3 de l'encadré ci-dessus peuvent être évaluées facilement et ne dépendent pas particulièrement du niveau physique. Cependant, le nombre d'insertions, de suppressions et de mises à jour influencera tout particulièrement[40] le design physique qui à son tour influencera l'évaluation des conditions 4 à 7.

La condition 4 est uniquement et précisément évaluable lorsque l'on connaît les stratégies d'exécution (choisies par le programmeur CODASYL ou l'optimiseur du SGBD-R) et les mécanismes physiques d'accès aux données choisis (clés, clés de tris, chemins d'accès, ...). Il en est de même pour les conditions 5 à 7.

Or, le choix des mécanismes d'accès (niveau physique), si l'on veut qu'il soit optimal (ou plutôt qu'il apporte de bonnes performances[41]) pour le schéma logique choisi, dépend des traitements mais aussi de l'environnement : vitesse des disques, du réseau, taux de remplissage choisi pour ne pas trop pénaliser les insertions ni les sélections, etc...

Il faut donc faire preuve d'un "certain feeling" pour pouvoir trancher. Il faudrait idéalement pouvoir choisir le schéma logique (avec ou sans dénormalisation) qui, optimisé physiquement, est globalement le moins coûteux.

Il faudrait par exemple savoir si un traitement accédant uniquement à E11 sera réalisé séquentiellement ou à l'aide d'une clé d'accès. Car dans le cas de l'accès séquentiel, le schéma normalisé sera plus performant, alors que les coûts seront semblables pour les deux schémas si l'on décide de faire des accès par clé. Il faudrait aussi par exemple dans le cas des SGBD-R connaître les réactions de l'optimiseur pour un traitement écrit d'une certaine manière (voir 3.4). Or, un traitement ne saura évidemment utiliser une clé d'accès que si elle est présente car implémentée lors du design physique.

On en est cependant convaincu, certains schémas logiques optimisés physiquement seront plus performants que d'autres aussi optimisés physiquement. Il peut donc être intéressant de penser à l'optimisation au niveau logique plutôt que de refuser de telles

40 Si les traitements de sélection bénéficient souvent de l'ajout d'une clé d'accès, les insertions, suppressions et mises à jour subissent elles souvent les modifications supplémentaires qu'il faut répercuter dans ces mécanismes d'accès à la manière des réorganisations des fichiers séquentiels indexés de COBOL.

41 Le choix de la meilleure configuration physique est en effet selon Comer [CORN78] un problème NP-complet.

optimisations sous prétexte qu'elles diminuent la lisibilité du schéma, ce qui est à notre avis inexacte, mais ceci est une question de goût.

Nous aurions besoin pour essayer de choisir le meilleur schéma **logico-physique** du :

- Schéma logique normalisé,
- Des **quantifications du schéma** (nombre d'entités par TE, nombre moyen de participation d'un TE à une association, nombre moyen de valeurs d'un attribut multivalué, distribution des valeurs d'un domaine, etc...),
- De l'énoncé des traitements, ainsi que leurs fréquences d'activation,
- De l'environnement physique.

On peut alors se prendre à rêver quand on s'aperçoit qu'il existe sur le marché la majorité des outils nécessaires au choix du meilleur design physico-logique (du moins pour ce qui est du modèle relationnel et certains SGBD-R).

DEC **RdbExpert** de Digital Equipment Corporation [RDBEX] est en effet à même de proposer un bon (si pas le meilleur) design physique pour une base de données existantes ou non sur base du workload, de la structure de données, des statistiques statiques et de l'environnement qu'on lui spécifie (éditeur à l'appui). Si la BD est opérationnelle, il peut bien évidemment rassembler les informations nécessaires via le système. Si la BD n'existe pas encore, il faut lui fournir le code SQL des traitements et diverses quantifications.

Un autre outil fort semblable est **RDT** d'IBM [FINK88]. Ce produit semble ne prendre en compte que le choix des index et clés de tri.

Ils peuvent tous deux utiliser le comportement de l'optimiseur. Il sera cependant important de "bien écrire" les requêtes si l'on ne veut pas les induire en erreur (voir 3.4).

DB-MAIN peut faciliter le passage d'un schéma normalisé à un autre schéma normalisé ainsi que peut-être prochainement la conversion des quantifications du schéma. Des études portant sur la conversion de ces quantifications ont déjà été réalisées.[HAI92].

Il ne resterait alors que la conversion des traitements qui ne soit supportée par un outil, bien que cela soit sans doute réalisable si ceux-ci ont été construits en suivant une certaine méthodologie. J-L Hainaut proposait dans [HAI86] un essai sur les transformations des traitements pour qu'ils "suivent" les transformations du schéma logique.

On pourrait alors sans doute procéder comme schématisé à la figure-11.

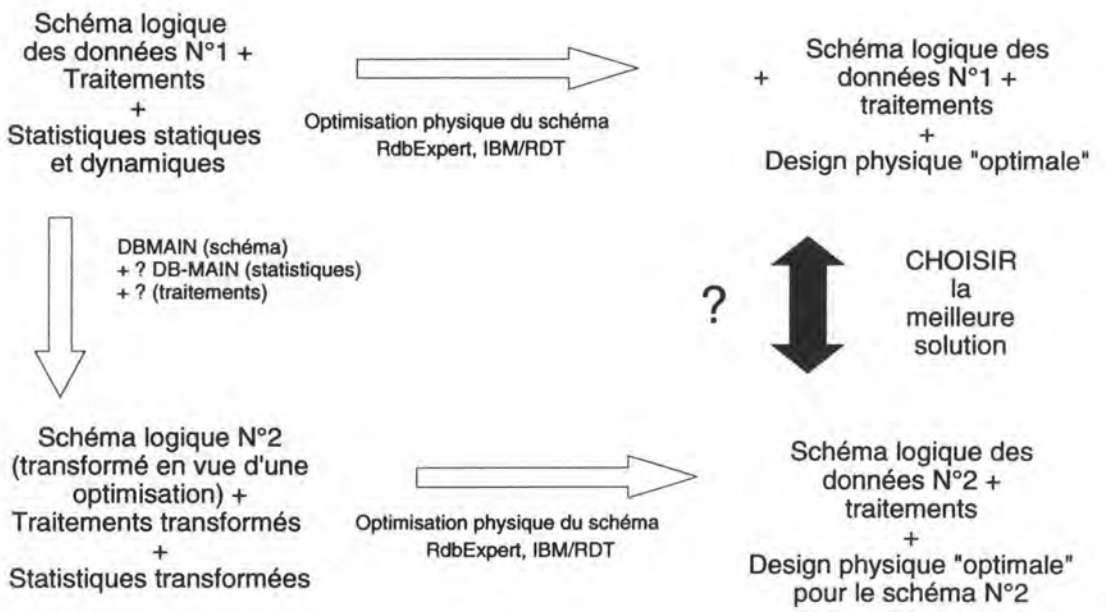


Figure-11 : *Ayant une proposition de schéma logique N°1 , on l'optimise physiquement (de manière globale) , on transforme ensuite le schéma logique en un schéma logique N°2 qui nous semble pouvoir améliorer les performances et on l'optimise physiquement. Il ne reste alors qu'à choisir le schéma physico-logique le plus performant*

En attendant une telle intégration d'outil (sans doute à trop long terme), il faut à notre avis différencier deux cas, celui où :

- Le schéma logique normalisé est composé de beaucoup de TE (dont le nombre d'entités est relativement petit) et où la navigation est très importante (situation caractéristique des environnement de CAO) et le cas où,
- Le schéma logique (est composé de peu de TE et) est peu traversé de part en part .

Dans le premier cas, le choix d'un design logico-physique est très complexe. Si les insertions ne sont pas trop fréquentes et que les recherches se font essentielllement par clés d'accès (donc forte sélectivité), on profitera pleinement de la dénormalisation. Sinon, il faudra compter sur la taille des buffers et laisser le schéma normalisé.

Deux autres solutions nous semblent cependant devoir être envisagées : l'utilisation d'un SGBD-OO très performant pour les traversées de graphes [SHA92], [DEL91] et l'implémentaton de sa propre structure de données en MC à l'aide d'un langage de programmation comme C++. C'est la décision qui a été prise pour implémenter l'outil DB-MAIN. Il est alors évident que l'on perd les facilités de gestion de reprise après incident et de concurrence proposées par les SGBD commercialisés.

Dans le second cas, il est possible de tenir compte de toutes les conditions faites dans l'encadré page 3.32. On essaiera alors d'obtenir le meilleur design logico-physique possible en se fiant au bon sens et en calculant les coûts physiques pour chaque opération. Les applications de type batch utilisant principalement l'accès séquentiel (paralèllement ou non sur deux fichiers), seront sans doute plus efficaces avec un schéma dénormalisé.

Nous allons maintenant passer à un autre sujet, les découpes verticales et horizontales.

3.2.2. Restructuration : découpe/fusion horizontale et verticale

Considérons respectivement $E=\{e_i\}$, $A=\{a_i\}$ l'ensemble des entités d'un TE et l'ensemble des attributs de ce même TE. Etant donné ces deux ensembles E et A, considérons $\Sigma=\{S_i\}$ où S_i est défini par une paire (A_i, E_i) où $A_i \subseteq A$ et $E_i \subseteq E$. Σ est appelé un découpage du type d'entité TE. S_i est un **fragment**.

Un découpage dans lequel $E_i = E$ pour tout i, et où $\cup A_i = A$ est appelé découpe verticale. Par contre, une découpe où $A_i = A$ pour tout i, et $\cup E_i = E$ est appelé découpe horizontale. Les unions peuvent être disjointes ou non.

Voyons un exemple de découpe horizontale suivi d'un exemple de découpe verticale.

Le TE ACHAT du schéma A de la Figure-12 est découpé horizontalement (schéma B) suivant le critère "Date postérieure au 1-1-94". Tous les achats vérifiant ce critère appartiennent alors à ACHAT-2 et les autres à ACHAT-1. Dans notre exemple, une entité est donc dans ACHAT-1 ou ACHAT-2, mais pas dans les deux. Une entité de FOURNISSEUR ne peut donc apparaître que dans une seule association soit dans CHEZ-1 ou dans CHEZ-2, d'où l'introduction d'une contrainte d'exclusion d'associations.

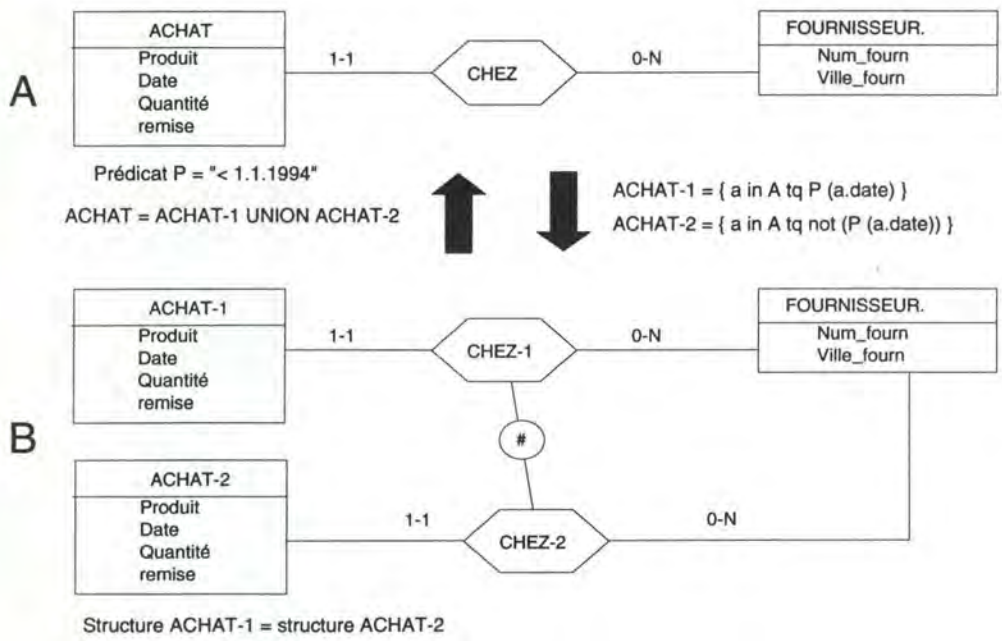


Figure-12 : *Découpe horizontale de ACHAT en ACHAT1 et ACHAT2. Le critère de découpe (partition) est "x ∈ ACHAT-1 ⇔ x.date < 1.1.1994"*

Pour la découpe verticale (Figure-13), nous avons séparé *pack-1, -2 et -3* de leur TE d'origine. On peut toutefois se demander quelle est la signification de cette découpe. C'est cette notion de **signification de la découpe** qui est centrale à notre avis. Mais voyons d'abord les transformations un peu plus formellement.

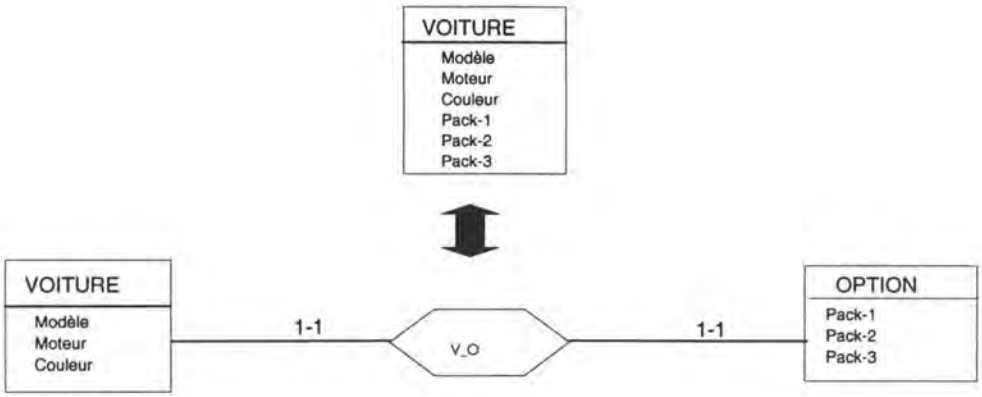


Figure-13 : *Découpe verticale de VOITURE, pack-1, -2, -3 ont été découpés du TE originel. C'est une transformation (d'extension) à sémantique constante.*

3.2.2.1. La transformation de découpe/fusion horizontale

Soit un TE E1 :

$$\mathbf{E1\ (e_1, ..., e_i, a_1, ..., a_m)\ où\ Id\ (E1) = \{e_1, ..., e_i\}}$$
et un prédicat de découpe.

On peut alors passer de ce schéma au schéma suivant par une transformation[1] symétriquement réversible, on fait alors une découpe horizontale :

$$\mathbf{E11\ (e_1, ..., e_i, a_1, ..., a_m)}$$
$$\mathbf{E12\ (e_1, ..., e_i, a_1, ..., a_m)}$$
$$\mathbf{où\ Id\ (E11 \cup E12) = \{e_1, ..., e_i\}}$$

Nous avons ici affaire avec un identifiant global aux deux TE.

Il faut insister sur le fait que la découpe horizontale n'est une transformation symétriquement réversible[2] qu'à la condition que le prédicat fasse partie intégrante du schéma contenant E1.

---

1 Cette transformation n'est pas formellement bien étudiée même si elle est intuitivement évidente. Il nous semble que les préconditions à la découpe soient les suivantes : E1 est un type d'entité, et il existe un prédicat P applicable aux instances de E1. La transformation t (de découpe) peut alors s'exprimer comme suit : E11 = E1 | P, E12 = E1 | P. La précondition à la transformation de fusion est : E11 et E12 sont des TE et E11 et E12 ont la même structure. La transformation t^{-1} peut alors s'exprimer comme suit E1 = E11 ∪ E12.

2 Partant du schéma non fragmenté, on peut partitionner l'ensemble des entités en deux sous-ensembles suivant un prédicat, celles qui le vérifient et les autres. Ayant ces deux ensembles, on peut faire leur union et retrouver l'ensemble de départ sans connaître le prédicat qui a conduit au partitionnement. La découpe est donc réversible. Par contre, partant du schéma partitionné, on peut faire l'union des deux ensembles mais, ne connaissant pas le prédicat, on sera bien incapable de reconstituer la partition. La transformation de découpe verticale n'est donc symétriquement réversible que si le prédicat fait partie intégrante du schéma non-partitionné.

3.2.2.2. La transformation de découpe/fusion verticale

Soit un TE E1 :

$$\mathbf{E1\ (\ e_1, ..., e_i, a_1, ..., a_m)\ où\ Id\ (E1) = \{e_1, ..., e_n\}}$$

On peut passer de ce schéma au schéma suivant par une transformation sysmétriquement réversible, on fait alors une découpe verticale :

$$\mathbf{E11\ (\ u_1, ..., u_r),\ E12\ (\ v_1, ..., v_s),\ ASS\ (\ r_1\ 1\text{-}1 : E11,\ r_2\ 1\text{-}1 : E12)}$$
$$\mathbf{où\ \{u_1, ..., u_r\} \cup \{v_1, ..., v_s\} = \{e_1, ..., e_i, a_1, ..., a_m\}}$$
$$\mathbf{et\ \{u_1, ..., u_r\} \cap \{v_1, ..., v_s\} = \varnothing.}$$

Remarquons que :

$e_1, e_i, a_1, ..., a_m$ peuvent être des rôles joués par E1 dans des associations,

On peut particulariser quelque peu cette transformation :

- Si $\{v_1, ..., v_s\}$formait un groupe de coexistence[3] dans E1, alors la connectivité du rôle r_1 pourra être ramenée à 0-1 au lieu de 1-1. On peut en voir un exemple à la Figure-14. Si de même, un des rôles de ASS a une connectivité minimale égale à 0, lors de la fusion on, verra apparaître un groupe de coexistence et c'est l'identifiant du TE jouant ce rôle qui sera choisi comme identifiant du TE résultat de la fusion.
- Si des éléments facultatifs mais ne formant pas un groupe de coexistence font l'objet d'une découpe, les rôles seront de connectivité 1-1 et les éléments resteront facultatifs.
- Si les deux rôles de ASS ont une connectivité minimale égale à 0, on ne fera pas la fusion[4].
- Si E1' et E1Compl ont tous deux un identifiant simple, un d'entre-eux devra être choisi comme identifiant primaire après la fusion. Du point de vue des optimisations, on choisira l'dentifiant ayant la plus petite taille[5].
- Si $\{e_1, ..., e_n\} \subset \{u_1, ..., u_r\}$, alors Id (E11) = $\{e_1, ..., e_n\}$, sinon, Id(E11) est composé de la partie de $\{e_1, ..., e_n\}$ incluse dans $\{u_1, ..., u_r\}$ et des attributs de E12 appartenant à $\{e_1, ..., e_n\}$.

---

3 Rappelons qu'un groupe de coexistence est un groupe d'attributs et/ou de rôles d'un TE tels que les valeurs des attributs et/ou les jeux de ces rôles par le TE existent tous ou qu'aucun n'existe. Ils ont chacun une connectivité 0-1.

4 On aurait, en effet, un identifiant composé de l'un ou l'autre identifiant des TE d'origine pour chaque entité, suivant que ce soit l'un ou l'autre qui soit NULL. L'intérêt d'une telle fusion est inexistant.

5 Cette considération n'a rien de "logique" mais est plutôt relative au niveau physique et à la facilité de gestion par le SGBD de clé de taille restreinte. On consultera pour cela le manuel de tuning des SGBD pour les conseils sur ce point particulier.

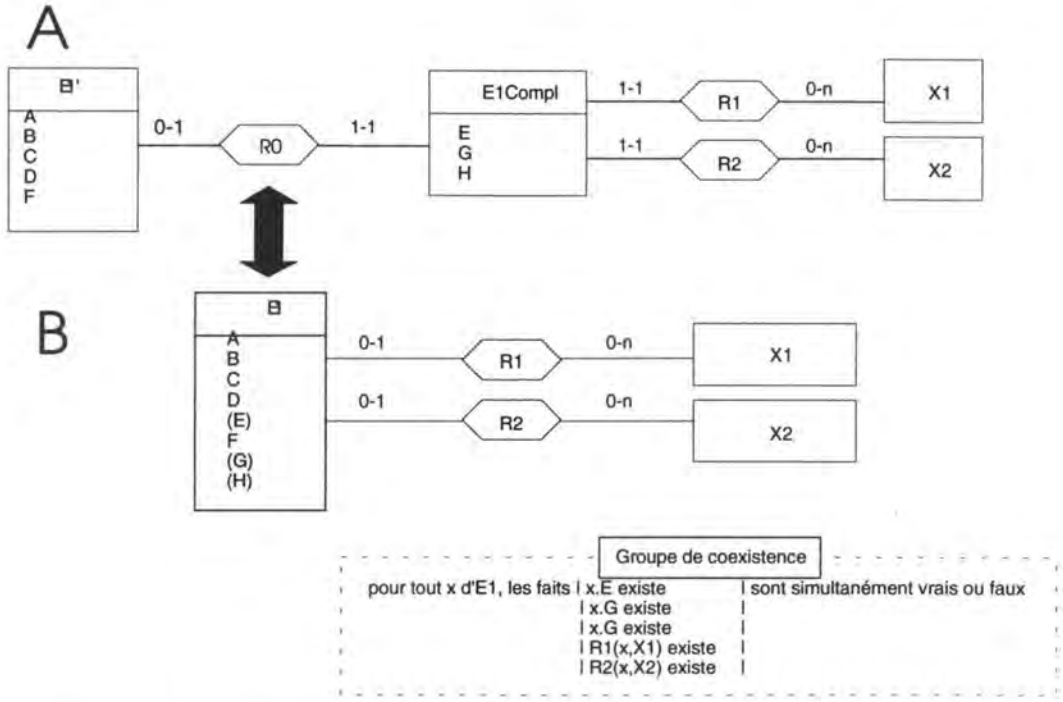


Figure-14 : *Découpe/fusion verticale d'un groupe de coexistence.. Le schéma A contient une relation de type one-to-one reliant E1' et E1Compl. On peut fusionner E1 et E1Compl pour former un TE unique E1. Comme le rôle joué par E1' est facultatif, on créera un groupe de coexistence.*

Si comme dans le dernier cas particulier présenté ci-dessus, on fait une découpe verticale qui scinde l'identifiant en deux (identifiant composé d'attributs éloignés), la gestion de cet identifiant (qui est une contrainte d'intégrité) deviendra très lourde. Nous avions déjà rencontré ce problème avec certains schémas normalisés (voir 3.2).

Le schéma B de la figure-15, résultant de la découpe verticale du schéma A, est "plus ou moins" conforme à CODASYL : on peut garder l'association mais il faudra relâcher la cardinalité du rôle joué par E1 (de 1-1 à 0-N). L'identifiant ne saura par contre être pris en charge par les SGBD CODASYL. C'est précisément la lourdeur de cette non prise en charge de l'identifiant que nous voulons souligner.

En effet, si on veut insérer un nouveau couple[6] d'entités (e1, e2) des TE E1 et E2, il faudra vérifier via l'application que la CI d'identification est respectée. Pour cela, l'application recherchera toutes les entités de E1 ayant même valeur d'attribut e1.A, et pour chacune de celles-ci, on recherchera l'entité de E2 qui lui est associée. On vérifiera enfin que cette dernière n'a pas même valeur pour l'attribut B que l'entité de E2 que l'on désire insérer (e2.B). Si aucun couple ne présente déjà de telles valeurs pour A et B, on pourra faire l'insertion. Il faudra donc effectuer pour la vérification, deux accès logiques multipliés par le nombre d'entités de E1 qui présentent la même valeur de A.

Si l'identifiant n'était pas scindé, la vérification (par le SGBD de surcroît) coûterait uniquement un accès logique. On le voit, la gestion de la contrainte devient très lourde, trop lourde.

---

6 Même si pour des raisons de conformité avec le SGBD, la relation est implémentée en tant que one-to-many, l'application doit conserver la sémantique one-to-one. On insèrera donc des couples et jamais uniquement une entité dans E1.

Même sans avoir encore décrit les avantages de la découpe verticale, il apparaît évident qu'il ne faille pas permettre de scinder un identifiant lors d'une découpe verticale, surtout si beaucoup d'insertions sont prévues.

Le schéma normalisé de la figure-4 (dénormalisation) perd lui aussi son intérêt lors des insertions si il est le siège d'identifiant composé. Seule une consommation inférieure de place et l'optimisation des updates resteraient à l'avantage de ce dernier.

De la même manière, toute coupure (lors d'une normalisation ou d'une découpe verticale) dans un ensemble d'attributs et/ou de rôles que constituent :

- un identifiant,
- les déterminants et déterminés d'une DF ou d'une DM,
- un groupe de coexistence,

compliquera fortement la gestion des CI associées à ces ensembles. De telles coupures ne devraient pas, dans la mesure du possible[7], être envisagées si ce n'est dans un environnement où les accès en lecture sont plus que prédominants.

De tels ensembles d'attributs devraient être considérés comme des atomes, dès lors non fissibles.

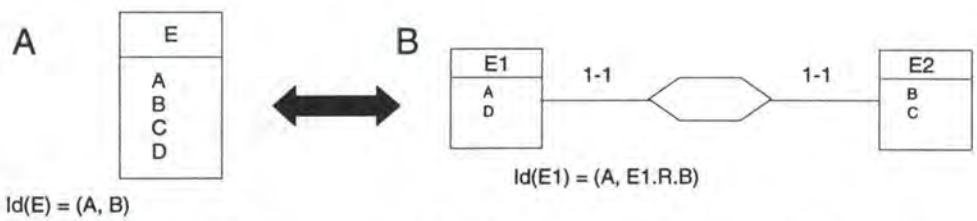


Figure-15 : *Découpe verticale du TE E. Le schéma B "presque" conforme à CODASYL nécessiterait une lourde vérification de la contrainte d'identification de E2.*

3.2.2.3. Avantages de la découpe horizontale

- **Si on ne doit accéder pour effectuer un traitement qu'à un seul des fragments** de taille (en nombre d'entités) n_i nécessairement inférieure à la taille N du TE original, et **si on sait auquel (importance de la sémantique)** alors,

- Une recherche séquentielle[8] d'une entité prendra évidemment beaucoup moins de temps vu le plus petit nombre d'entités : $n_i < N \Rightarrow (n_i + 1)/2 < (N + 1)/2$.

- Si de plus un mécanisme d'accès comme les SEARCH-KEYs, un INDEXED SET-TYPE de CODASYL ou les index (secondaires ou primaires) du relationnel a été jugé nécessaire lors du design "classique"[9] de la BD, **sa gestion et son utilisation seront allégées.**

7 Si les avantages du schéma scindant une contrainte d'intégrité sont très intéressants, on envisagera la perte de sémantique

8 Pour rappel, rechercher séquentiellement une entité parmi N entités rangées de manière quelconque, coûte en moyenne (1+N)/2 accès logiques.

9 Nous entendons par classique une méthode telle que celle présentée dans [HAI86] où la détermination des clés et chemins d'accès nécessaires se fait sur base des traitements et des quantifications en accès logiques avec une possibilité de revoir les décisions prises au niveau logique lors de la phase d'implémentation. On se

En effet, plus le nombre d'entités est faible, moins le nombre de niveaux[10] d'index sera élevé, moins le nombre d'accès physiques sera grand lors de l'utilisation de ce mécanisme et donc plus le temps nécessaire à son utilisation sera court. La gestion de ces mécanismes d'accès deviendra peut-être même inutile, l'accès séquentiel devenant plus performant.

On ne sait clairement constater cela qu'en considérant non pas les accès logiques, mais les accès physiques et donc en tenant compte entre autres du nombre d'entités par page.

Encore une fois, des transformations logiques ne sont parfois clairement justifiables en tant qu'optimisations qu'en termes d'accès physiques.

Supposons par exemple que nous ayons 10.000 achats groupés par bloc de 10, dont 2.000 sont des achats vérifiant le prédicat P de la figure-12, 400 de ces derniers ayant en plus été faits chez le fournisseur 1. Supposons aussi que l'on veuille effectuer le traitement suivant *"retrouver les achats de cette année et faits chez le fournisseur 1"* (Trt-6)

En ayant le schéma A de la Figure-12 enrichi d'une clé d'accès composée[11] sur (*Num_fourn, Date*), le nombre d'accès logiques nécessaires à la recherche de ces entités sera de 400. Nous avons cependant déjà dit (à la section précédente) que ces accès logiques ne se traduiront pas tous en accès physiques.

Par contre, on peut estimer que le nombre d'accès physiques pour retrouver ces 400 achats parmi les 10.000 regroupés toujours par bloc de 10 entités et avec une organisation physique "en tas[12]", si l'on accédait séquentiellement au TE, serait[13] égal à 1.000. L'utilisation de la clé d'accès est donc très intéressante. On aurait d'ailleurs tirer cette conclusion en ne considérant que les accès logiques comme dans [HAI86].

Si on découpe horizontalement comme proposé plus haut (figure-12 schéma B), le TE ACHAT-1 contiendra 2.000 entités dont les 400 achats faits chez le fournisseur 1. Une clé d'accès sur *Num_four* aurait alors une sélectivité de 0.2 (400/2000). Le coût logique de recherche de ces 400 entités via la clé d'accès serait donc de 400. Une lecture séquentielle de la table coûterait seulement quant à elle 200 accès physiques. Si il est vrai que le nombre minimum d'accès physiques nécessaires[14] à la réalisation de trt-4 en utilisant la clé d'accès sur *Num_fourn* est d'approximativement 180 accès physiques, il est aussi vrai que ces 180 accès suffiront très rarement et que le gain de 20 accès (200-180) ne justifira pas les coûts de maintenance physique de cette clé.

---

réfèrera plus particulièrement dans l'ouvrage sus-cité à la détermination du Schéma des Accès Nécessaires (SAN).

10 Faisant l'hypothèse de l'utilisation d'un mécanisme d'accès de type B-tree dense [DAT90], on peut estimer que le nombre de niveaux d'index est proportionnel à Log N où N est le nombre de lignes à indexer. Le nombre d'accès physiques nécessaires pour accèder à une ligne via un index de ce type est approximativement égal au nombre de niveaux d'index.

11 Une clé composée est utilisable pour retrouver des entités vérifiant deux ou plusieurs conditions à la fois.

12 Cette organisation (ou plutôt non organisation des entités dans les pages physiques) implique une distribution uniforme des valeurs des entités de Fournisseur parmi les pages. Cette organistion est l'organisation par défaut des tables des SGBD-R. On consultera les manuels de références des différents SGBD ou [DAT90].

13 Une estimation plus fine peut être obtenue en utilisant la formule proposée dans [YAO77] mais la précision extrême apportée par cette formule, n'a pas grand intérêt en pratique.

14 Ce nombre est le résultat de l'application de la formule suivante : $\mathbf{(1-(1-S)^{FB})}$ **x Nombre_de_page**.

On voit donc que l'utilisation d'un mécanisme d'accès peut être rendue moins intéressante par la découpe horizontale, celle-ci jouant en quelque sorte le rôle de ce mécanisme. **Mais il faut savoir quel TE accéder. Il faut donc que la découpe supporte une certaine sémantique.**

Dans notre exemple, nous sommes passés, pour Trt-6, de 400 accès en utilisant une clé d'accès que le SGBD doit gérer, à 200 accès en n'utilisant aucune clé d'accès.

Insistons sur le fait que cette découpe horizontale est idéale pour Trt-6 qui **sait** quel TE accéder et n'accède que celui-là.

- La gestion de la sécurité peut aussi être facilitée par une découpe horizontale

On peut par exemple découper horizontalement un TE DOSSIERS en DOSSIERS_PUBLICS et DOSSIERS_CONFIDENTIELS. Ici aussi, la découpe supporte une certaine sémantique qui peut être déduite des traitements ou des privilèges différents de certains utilisateurs. Cette découpe n'était peut-être pas significative lors du design conceptuel vu que DOSSIERS_PUBLICS et DOSSIERS_CONFIDENTIELS sont en fait tous deux des DOSSIERS.

Cette découpe apportant des optimisations au même titre que la découpe précédente, ne complique pas la tâche du programmeur si du moins aucun traitement n'accède aux deux TE du part la sémantique qu'elle supporte.

Les découpes que nous allons présenter maintenant peuvent apporter des gains de performances mais ne supportent pas une sémantique pertinente pour l'application et le programmeur d'application.

- Si on partitionne **suivant la probabilité d'accès** aux entités du TE, -un fragment contient les entités les plus accédées-, mais sans pouvoir être sûr que l'on accèdera qu'à ce seul fragment, on **améliorera le temps d'accès moyen à une entité.**

Supposons par exemple que 80% des traitements accèdent à 20% des entités. Si on admet que tous les traitements d'accès ont la même importance, c'est-à-dire qu'aucun d'entre eux n'est prioritaire, on essaiera d'améliorer les performances du maximum de traitements.

Pour ce faire, on créera un TE, "hot TE", regroupant les entités les plus accédées de l'ensemble total des entités (20%). On optimisera (physiquement) les accès à ce "hot TE". Le second fragment reccueillera les 80% restants, les entités les moins accédées.

Voyons un exemple chiffré. Supposons que le TE à partitionner contienne 10.000 entités. Si nous sommes en relationnel, une clé d'accès identifiante, secondaire aux 10.000 entités se traduira approximativement par 3 niveaux d'index. Pour le fragment (table) ayant 20 % des lignes de la table originale, l'index aura 2 niveaux et la table regroupant les 80 % restants aura un index avec 3 niveaux.

Si on accédait à une ligne de la table originale via la clé d'accès, on fera 3 + 1 = 4 accès physiques pour tout traitement.

Avec la solution "multi-tables", on fera dans 80 % des cas 2 + 1 = 3 accès et 3 +1 dans les autres cas, ce qui fait en moyenne 0.8 x 3 + 0.2 x 4 = 3.2 accès physiques. Cela fait une diminution des temps d'accès de 20%.

Cependant, si on ne sait pas quelle table accéder à coup sûr, on devra d'abord chercher dans la première table (la plus "probable") et, en cas d'échec, il faudra encore accéder à la seconde. Cela coûterait 0.8 x 3 + 0.2 x (3 + 4) = 3.8 accès physiques, ce qui constitue une amélioration sensible.

Sans clé d'accès aucune (lecture séquentielle), l'amélioration est plus spectaculaire encore, même si on ne connaît pas le fragment à accéder. Supposons que le facteur de blocage soit égal à 10. Le coût en accès physiques avec le schéma partitionné est alors de 0.8 x (200 + 1)/2 + 0.2 x (201/2 + 801/2) = 181, ce qui est largement inférieur à (10000/10+1)/2 = 500. Il suffit en effet souvent (dans 80% des cas) de lire séquentiellement la première table, ce qui nous coûtera en moyenne (200+1)/2. Dans les autres cas, on devra en plus lire séquentiellement l'autre table.

- Le partage des données dans l'hypothèse d'un design Top-Down d'une base de données distribuée passe aussi par la découpe horizontale [BOU92].

Vu l'infériorité actuelle du taux de transfert des réseaux face à celui des disques, on préfèrera avoir ses propres données dans un fragment "en local". Chaque succursale d'une banque aura par exemple les données de ses clients chez elle et pourra accéder moins efficacement aux fragments localisés dans d'autres succursales, chaque fragment résultant de la découpe horizontale d'un même TE, disons CLIENT.

Cependant, il n'est pas pensable que le programmeur doive savoir dans quelle ville habite le client, pour effectuer une recherche d'informations le concernant et stockées dans un TE regroupant tous les clients habitant la même ville.

- L'administration des données peut aussi être facilitée par la découpe horizontale.

L'un des fragments peut par exemple faire l'objet d'un back-up tout en laissant l'accès aux autres entités possibles. Dans le cas de ORACLE V7.0, un TABLESPACE (une unité spécifique regroupant une ou plusieurs tables[15] et pouvant recevoir une certaine paramétrisation physique) pourrait ne contenir qu'un seul des fragments. Un TABLESPACE peut alors être mis "on-line" ou "off-line" afin d'effectuer un tâche administrative sur celui-ci sans perturber l'exécution des traitements n'accédant pas à ce fragment. Un TABLESPACE ou plus exactement les fichiers qui lui sont associés peut aussi être stocké sur un disque plus ou moins performant selon les fréquences d'accès au fragment qu'il contient, ce qui diminue le coût de stockage. On peut donc penser au "storage management[16]" au niveau logique !

On peut aussi partitionner suivant le taux de mises à jour. Si certaines entités sont fortement touchées par les mises à jour, on les mettra dans un même fragment afin que les réorganisations internes effectuées par le SGBD puissent se faire en parallèle avec les accès aux entités "stables". On pourra même peut-être fixer certains paramètres physiques pour une meilleure gestion des entités souvent mises à jour.

---

15 Une TABLESPACE peut plus exactement regrouper non seulement des tables mais aussi des index et d'autres éléments spécifiques à ORACLE.

16 Le storage management est l'art de mettre les données les plus souvent utilisées sur des supports de stockage rapides (mais chers) et inversément pour les données très rarement accédées. Le but est donc de minimiser une fonction du coût de stockage et de la disponibilité des informations.

- Le partitionnement peut aussi **augmenter la concurrence** et la politique de reprise après incident.

Si une transaction accède essentiellement aux entités d'un seul des fragments, elle ne dérangera pas d'autres opérations en ayant un "lock" sur certaines pages d'index ou sur le TE lui-même. Une insertion, ou une suppression appliquée à un des fragments pourrait même s'effectuer indépendamment d'un autre traitement utilisant un autre fragment, ce ne sera pas le cas pour la découpe verticale.

La concurrence des requêtes accédant les achats postérieurs au 1-1-94 d'une part et les achats antérieurs à cette date d'autre part, serait très fortement améliorée.

Elle le serait d'autant plus que ces fragments seraient localisés sur des disques différents, ce qui ne causerait évidemment aucune contention au niveau de ceux-ci.

Cette amélioration de la concurrence est valable pour tout partitionnement, qu'il supporte ou non une sémantique.

Si la charge en nombre de traitements est bien "balancée" sur les différents fragments et si ses fragments sont stockés sur des supports différents, en cas de chute d'un seul des supports, seuls les traitements accédant le fragment inutilisable seront stoppés.

3.2.2.4. Désavantages de la découpe horizontale

Tout d'abord, les **contraintes** d'intégrité sur le TE d'origine seront **dédoublées.** Si le TE d'origine était le siège d'une DF qu'il fallait respecter, les deux fragments résultant seront, eux aussi, le siège de la même DF.

Deuxièmement, **les associations** dans lesquelles le TE d'origine joue un rôle **sont dédoublées. Des contraintes** sur les rôles joués par le TE associé au TE d'origine ou sur les associations complèteront[17] le schéma dans le cas d'association one-to-many. Ce seront des contraintes d'exclusion, (d'inclusion ou d'égalité de rôles) ou d'association qu'il faudra gérer.

Troisièmement, il sera souvent très **difficile de garantir** le fait qu'un groupe d'attributs et/ou de rôles soit un **identifiant** non seulement pour chaque TE, mais aussi pour l'union des entités des différents TE résultants (voir 3.2.2.1).

Dans le même ordre d'idées, il sera difficile d'avoir des **clés d'accès à l'union des fragments**. Les SGBD-R commercialisés ne supportent en tout cas pas ce genre de clé. Il en est donc de même pour la Foreign Key.

De ce fait, le code de certains traitements sera plus lourd pour tous les SGBD et plus coûteux pour les SGBD-R[18].

---

17 Soit R une association one-to-many entre A et B, A devant être partitionné horizontalement. Si A est associé à plusieurs B, on pourrait avoir une CI sur les rôles joués par B dans les nouvelles associations. Si par contre B est associé à plusieurs A comme dans l'exemple à la Figure-12, on aura une CI entre les associations.

18 Pour les SGBD de type CODASYL, le code des traitements sera plus complexe mais les performances ne devraient pas trop en pâtir. Les accès physiques inutiles pour s'apercevoir qu'une entité n'est pas associée à une autre entitée ne sont pas le lot des SGBD CODASYL mais relationnels. De plus, si le TE partitionné est du côté "many" de l'association, les deux associations résultant de la découpe seront efficacement implémentées en CODASYL par un seul *Set Type* avec des *member* appartenant aux différents fragments.

Une jointure des entités de A et des entités de B vérifiant un certain critère (B est donc la table "extérieure" de la stratégie Nested-Loop d'exécution de la jointure) du schéma A de la figure-16 est très simple à réaliser. Par contre, avec le schéma B, il faudra non seulement connaître l'identifiant de A auquel chaque B sélectionné est associé mais aussi le TE dans lequel elle se trouve (A1 ou A2). On ne saura donc pas faire prendre en charge la jointure par les SGBD-R vu que la référence que l'on trouve dans B donne un identifiant *et* le nom d'un TE.

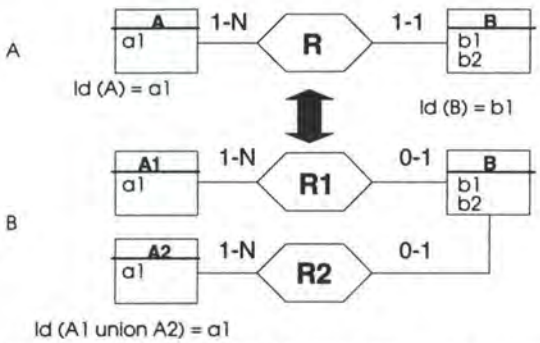


Figure-16 : *Découpe horizontale d'un TE participant à une association du côté One*

Il faudra alors écrire pour réaliser le traitement :

```
SELECT *
FROM A, B
WHERE B.b2 = :x
AND A.a1 = B.a1;
```

valable sur le schéma A de la figure-16 conformé au relationnel (B.a1 étant la Foreign Key remplaçant l'association), la requête suivante et faire l'union de deux jointures :

```
SELECT      *
FROM        A1, B
WHERE       B.b2 = :x
AND         A1.a1 = B.a1

UNION

SELECT      *
FROM        A2, B
WHERE       B.b2 = :x
AND         A2.a1 = B.a1;
```

Or, on imagine très bien que la seconde requête sera beaucoup moins performante : elle impliquera en effet très souvent un tri des deux résultas [LON92], et engendrera beaucoup d'accès physiques "inutiles". En effet, pour chaque entité de B, une seule des sous-requêtes rendra un résultat. Or, pour chaque B, dans chacune des jointures, l'optimiseur du SGBD-R essaiera de trouver le A associé et consommera pour cela des accès physiques. On verra que l'on peut éviter cette perte de performance[19] en ajoutant de l'information technique (3.3.4).

Plus généralement, un traitement qui ne peut tirer profit de la partition comme : *"lire toutes les entités de A1 et A2"*, sera pénalisé d'une manière ou d'une autre. On ne constatera

19 Encourue uniquement par les SGBD-R.

évidemment aucune différence au niveau du nombre d'accès logiques, ou même au niveau du nombre d'accès physiques si on parcourt séquentiellement les fragments, à moins que l'on ne prenne en compte la mise en oeuvre du traitement : temps CPU, union des résultats des recherches sur chacun des fragments, etc...

Cependant, pour des accès par clé, si il n'y a aucune corrélation entre la distribution des valeurs de clé et le partitionnement, la découpe pénalisera très probablement les traitements utilisant la clé. La relation suivante peut en effet faire sentir la contre-performance des traitements faisant des accès par clé dans de telles conditions : $\lceil Log(N) \rceil < \lceil Log(N1) \rceil + \lceil Log(N2) \rceil$ où N=N1+N2 pour de grandes valeurs de N. N, N1 et N2 sont respectivement le nombre d'entités du TE original et des deux fragments.

Cette relation montre que le nombre de niveaux d'index du TE original est inférieur à la somme des niveaux d'index des fragments.

En admettant qu'un traitement "n'ait aucune idée" dans quel fragment chercher l'information, qu'il y ait deux fragments et que ceux-ci contiennent tous deux 50.000 entités, un accès par clé sur le TE non partitionné, coûterait pour récupérer la première entité à peu près $\lceil Log(100.000) \rceil = 5$. Pour retrouver les premières entités dans chacun des deux fragments, il en coûtera $2 \times \lceil Log(50.000) \rceil = 10$.

3.2.2.5. Avantages de la découpe verticale

Tout d'abord, elle peut, au même titre que la découpe horizontale, **améliorer fortement le parallélisme** de traitements accèdant des ensembles disjoints d'attributs d'un TE, la distribution des données entre différents sites[20], ou encore la politique de reprise après incident.

Les Trt-a et Trt-b utilisant des ensembles disjoints d'attributs (voir tab-4) pourraient s'exécuter en parallèle.

	Attributs d'un TE						
	a	**b**	**c**	**d**	**e**	**f**	**g**
Trt-a	1	0	1	0	0	1	0
Trt-b	0	1	0	1	1	0	0

Tab-4 : *Des ensembles disjoints d'attributs utilisés par le Trt-a et Trt-b.*

Cependant, alors qu'avec la découpe horizontale une insertion ou une suppression pouvait ne toucher qu'un seul des fragments, ici les insertions et suppressions toucheront les deux fragments. Remarquons toutefois que le parallélisme des traitements effectuant des sélections (lectures) ne sera véritablement effectif que, si il n'y a pas de contention à niveau quel qu'il soit et en particulier pas au niveau des disques. On essaiera donc de localiser les fragments sur des disques différents.

20 Si toutefois ceux-ci n'ont pas les mêmes intérêts, autrement dit, n'accèdent pas le même sous-ensemble d'attributs

Elle peut améliorer également le "storage management". On mettra alors dans un fragment les attributs les moins accédés durant une journée ou les attributs accédés uniquement le lundi, etc...

Mais un de ces avantages principaux est sans doute **l'augmentation du facteur de blocage**[21] qu'elle entraîne, avec toutes les conséquences dont nous avons déjà parlé pour les traitements ne devant accéder qu'à un seul des fragments (voir 3.2.1).

Enfin, on peut mettre à l'abri des informations qui doivent rester inaccessibles pour certains utilisateurs. On pense ici par exemple, aux données à caractère personnel protégées par les lois. En mettant ces attributs dans un fragment séparé, on pourra faire gérer la limitation des accès à ces attributs par le SGBD lui-même. ORACLE V7.0 peut par exemple limiter l'accès de certains utilisateurs à une table du schéma.

Gérée par le SGBD, la sécurité n'en sera que plus efficace, une requête ad-hoc ne saura en effet pas accéder à l'information alors que si la gestion de la sécurité est effectuée par une application, une requête ad-hoc saura outrepasser cette sécurité.

3.2.2.6. Désavantages de la découpe verticale

Elle peut diminuer la lisibilité du schéma et en augmenter la complexité (augmentation de la taille du catalogue des SGBD-R par exemple).

Elle augmente la place nécessaire au stockage de l'information (place nécessaire au stockage de l'association).

Mais surtout, **elle ralentit l'exécution de traitements qui "utilise" des attributs situés dans plusieurs fragments du TE d'origine** comme le schéma normalisé pénalisait les traitements devant traverser l'association dans les sens Many-to-One (même coût).

Les insertions et suppressions, utilisant tous les attributs, seront donc ralenties.

21 C'est le même phénomène physique que celui observé lors de la normalisation.

3.2.2.7. Avantages des découpes verticales et horizontales : synthèse

Nous l'avons vu sur des exemples dans les pages qui précèdent, on peut améliorer à l'aide des découpes verticales et horizontales les performances de la base de données suivant des critères aussi divers que :

- la facilité de reprise après incident,
- le storage management,
- la facilité d'administration,
- la confidentialité,
- la distribution des données sur plusieurs sites (SGBD distribués),
- l'augmentation du parallélisme,
- l'amélioration des performances des traitements qui "se contentent" d'accèder à un seul des fragments.

Certains de ces critères peuvent être contradictoires. Ce sera par exemple peut-être le cas des critères de confidentialité et d'amélioration des performances de traitements ayant besoin d'accéder à tous les fragments.

On peut encore remarquer que le parallélisme ne saurait diminuer lorsque l'on partitionne.

On peut dire que l'amélioration des performances apportée par la découpe verticale provient de deux phénomènes :

- L'augmentation du blocking factor,
- L'augmentation du parallélisme.

L'amélioration des performances apportée par la découpe horizontale provient elle, de :

- La diminution de la sélectivité du filtre des traitements ne devant accéder qu'à un seul des fragments,
- La diminution du nombre d'entités d'un fragment,
- L'augmentation probable du parallélisme.

3.2.2.8. Comment partitionner verticalement ?

Contrairement à la découpe horizontale, à la normalisation ou aux optimisations propres aux SGBD-R que nous verrons par la suite, la découpe horizontale a été le sujet de diverses études [HOFF75], [HAM79], [NAVA84], [CORN87]. Nous allons présenter très brièvement les idées à la base de ces études et les critiquer.

En fait, un TE donné sera "touché" par différents traitements de types divers (insertion, suppression, sélection...) qui requerront différents sous-ensembles d'attributs $\{a_i\}$.

En supposant connus les traitements[22] et leurs fréquences d'activation d'une part et la longueur des attributs d'autre part, il serait possible de faire un partitionnement qui aurait de fortes chances d'être plus efficace en terme du nombre de traitements absorbés par le système par unité de temps (better throughput), c'est en tout cas ce que pensent Hoffer & Severance [HOFF75].

A partir d'une matrice (T x A) où T est l'ensemble des traitements et A l'ensemble des attributs (voir Tab-5), Hoffer & Severance proposent une **méthode analytique** de découpe de l'ensemble des attributs. Leur méthode est basée sur la notion d'**affinité de deux attributs** (Aff) : deux attributs i et j devraient rester dans le même fragment si ils sont souvent accédés ensemble par beaucoup de traitements.

	Type-trait.	*Importance*	Att-1	...	Att-m
Trt-1			$TA_{1,1}$		
...	...	...		...	
Trt-n					$TA_{n,m}$
Taille des attributs				...	

Tab-5 : *Matrices des informations utilisées en entrée par la méthode de Hoffer & Severance; la matrice (T x A)..*

Dans tab-5, à un élément de T x A (t,a) noté $TA_{t,a}$, on a fait correspondre 1 si le traitement t utilise l'attribut a et 0 sinon. On a en plus, pondéré les traitements par leur importance, comme leur nombre d'occurrence par jour et/ou par leurs relatives priorités. On y ajoute encore le type du traitement (insertion sélection) et la taille des attributs

Ils définissent une première approximation de l'affinité : $Aff_{ij} = \Sigma_t \, freq_t \, TA_{t,i} \, TA_{t,j}$. Hoffer & Severance incorporent également dans leur fonction d'affinité, une fonction de la taille des attributs.

Après l'application d'un algorithme utilisant la notion d'affinité dont on peut trouver la description dans leur article, on obtient une matrice d'affinité (AxA) carrée, diagonale et dans laquelle les grandes valeurs sont entourées de grandes valeurs et les petites de petites.

Leur méthode ne produit rien d'autre.

22 On espère avoir le moins possible de traitements had-hoc.

Cette méthode nous a laissés perplexes quant à son adéquation et sa pertinence.

Tout d'abord, il faut pour obtenir la découpe, faire un **effort d'interprétation** de la matrice d'affinité qui ne nous semble pas "direct". Et les auteurs de reconnaître :

"Further experimentation is necessary to develop a procedure for interpreting the output matrix...".

Deuxièmement, elle ne tient **pas compte du type de recherche utilisé** par un traitement : par clé ou séquentiel.

Or, comme nous l'avons dit, ce sont les traitements séquentiels accédant à un seul des fragments qui profiteront le plus de l'augmentation du facteur de blocage et verront leur temps d'exécution diminuer fortement. On ne constatera en effet pas d'amélioration aussi spectaculaire pour des traitements faisant des accès par clé.

De plus la méthode de Hoffer & Severance ne pénalise pas (du moins explicitement) une découpe qui oblige des traitements à accéder à plusieurs fragments. On sait pourtant toute la lourdeur des opérations de jointure dans les SGBD-R par exemple. Le problème provient du fait qu'ils ne considèrent en aucune manière le nombre des accès physiques dont le calcul diffère pourtant fortement suivant le type d'accès utilisé.

Enfin, leur méthode ne tient pas compte d'éventuels groupes (identifiant, DF, coexistence,...) qui devraient être considérés tels des atomes, nous en avons déjà parlé. Ceci pourrait évidemment constituer une simple extension de leur méthode.

On peut qualifier cette approche de "brutale", nous reviendrons sur la signification que l'on donne à cet adjectif.

Navathe [NAV84] poursuit dans la voie tracée par Hoffer & Severance en donnant des méthodes d'interprétation de la matrice d'affinité résultante. Ces méthodes d'interprétation restent assez vagues et traduisent une certaine intuition que l'on peut avoir, et cela sans jamais se référer aux accès physiques. Dans la méthode proposée par Navathe, la matrice d'affinité produite par la méthode de Hoffer & Severance joue un rôle de première sélection parmi toutes les découpes possibles[23].

J.L Guigou [GUIG87] a repris toutes les techniques de groupement et de séparation de données semblables à celles utilisant la notion d'affinité mais sans s'intéresser particulièrement au problème de la découpe verticale dans les bases de données.

[23] Hammer a montré que le nombre des découpes possibles d'un TE composé de m attributs est égal au nombre de Bell [HAM79]. Le nombre de Bell tend vers m^m lorsque m tend vers l'infini. Pour m = 15, Bell (15) $\approx 10^9$.

Pour pallier à la critique que nous avons faite (ne tient pas compte des accès physiques), on peut imaginer un modèle mathématique d'optimisation d'une fonction du coût physique de tous les traitements accédant le TE que l'on désire partitionner verticalement. Ce modèle pourrait alors être résolu par un algorithme dit de *branch and bound* [FICH82].

Nous aurions les variables de décisions booléennes suivantes : $X_{i,j}$;

i = 1, ..., N étant l'indice donnant le numéro de l'attribut.
j = 1, 2 étant l'indice donnant le numéro du fragment; et

$X_{i,j}$ *serait égal à 1 si l'attribut i appartenait au fragment j et égal 0 sinon.*

Soit a_t, *l'attribut "filtre" pour le traitement t (l'attribut avec lequel on effectue la sélection). C'est à l'aide de cet attribut que la sélection des entités requises par le traitement t est effectuée. C'est donc l'attribut le plus sélectif apparaissant dans la condition de sélection du traitement t.*

On dira que le segment contenant l'attribut a_t *est le segment "principal" pour le traitement t, autrement dit le segment accédé en premier par ce traitement (table extérieure de la stratégie Nested-Loop).*

La fonction objective à minimiser serait :

$$Z = \min \sum_{t \in T} \text{Freq}_t \times (C^t_{\text{Princ}} + C^t_{\text{Sec}}),$$

où C^t_{Princ} et C^t_{Sec} *sont respectivement les coûts d'accès pour le traitement t aux fragments principal et secondaire (au besoin).*

Si un attribut k est dans le même segment que a_t, *c'est-à-dire le segment principal pour le traitement t, alors* $I(k,t)=1$, *et*

$$I(k,t) = X_{a_t,1}\ X_{k,1} + X_{a_t,2}\ X_{k,2}$$

La longueur d'une entité du fragment principal pour le traitement t (fragment 1 ou 2), est alors égale à :

$$T_t^{\text{Princ}} = \sum_k^N (I(k,t)\ \theta_k)$$

où θ_k *est la taille de l'attribut k.*

Le nombre d'accès physico-logiques nécessaire pour récupérer toute l'information requise par le traitement t dans le segment principal ou autrement dit C^t_{Princ} *est égal à (nous avons déjà proposé ces approximations du coût physique des traitements dans le Tab-3) :*

- $T_t^{\text{Princ}} \times N_r\ /\ TP$

où N_r est le nombre d'entités du type d'entité que l'on veut fragmenter (c'est également le nombre d'entités de chaque fragment) et TP la taille physique d'un bloc de la base de données ***si et seulement si le traitement t est séquentiel et k >1.***

- $T_t^{Princ} \times N_r \times S_{a_t} / TP$
  où S_{a_t} est la sélectivité de a_t, ***si et seulement si le traitement t est semi-séquentiel,*** *c'est-à-dire que a_t est une clé primaire au sens où elle impose l'ordre des entités dans le fragment principal du traitement t.*

- $N_r \times S_{a_t}$
  si et seulement si le traitement t est un traitement par clé, *(il utilise donc une clé d'accès : a_t).*

Le traitement t devra accéder au second fragment si il ne trouve pas toute l'information nécessaire dans le fragment principal.
Comme une relation one-to-one unit les deux fragments, on suppose qu'un et un seul accès physico-logique sera nécessaire pour chaque entité sélectionnée du fragment principal afin de récupérer l'information présente dans le second fragment. On ne prend donc pas en compte le coût d'utilisation de la clé d'accès qu'est la primary key du second fragment.

Soit $a_t, a_t^1, a_t^2, \ldots, a_t^{N(t)}$, *les attributs utilisés par le traitement t, on devra alors accéder au second fragment (après avoir accédé au premier) si et seulement si A2 = 0 avec :*

$$A2 = X_{a_t,1} X_{a_t^1,1} \ldots X_{a_t^{N(t)},1} + X_{a_t,2} X_{a_t^1,2} \ldots X_{a_t^{N(t)},2} .$$

Le coût d'accès à ce second fragment, C_{Sec}^t est donc égal à $N_r \times S_{a_t} \times (1 - A2)$.

Cette méthode quoique maintenant prenant en compte[24] le nombre d'accès physiques est tout aussi brutale et ne saura tout prendre en compte.

Par "méthode brutale" nous entendons méthode ne prenant pas en compte le contenu sémantique du schéma. C'est de l'optimisation logique à tout prix.

Le modèle de Hoffer & Séverance et le modèle que nous venons de présenter risquent en effet de conduire à un schéma optimal (dans la limite des phénomènes modélisés) au point de vue du coût mais dont l'interprétation en terme significatif pour le programmeur serait difficile. Ces méthodes ne se soucient en effet pas si les traitements (introduit par exemple dans la matrice (TxA) ont une sémantique. On pourrait à la limite entrer des 0 et 1 aléatoirement, ces méthodes trouveraient encore sans doute une découpe optimale.

---

24 Le nombre d'accès physiques considéré dans le modèle que nous avons présenté n'est qu'une approximation très grossière, il ne tient pas compte du coût d'utilisation des mécanismes d'accès ni la possible utilisation de stratégies plus intelligentes mises en oeuvre par certains SGBD, ni les buffer. Il ne tient pas non plus compte du fait qu'après avoir partitionné, un traitement qui faisait un accès par clé sur le schéma non partitionné pourrait être plus performant en utilisant un accès séquentiel, comme nous l'avons vu pour la normalisation en 3.2.1.

Or, les traitements ont une signification et ne prennent souvent en compte qu'une certaine **facette** d'un TE.

Nous sommes d'ailleurs d'avis que les partitionnements horizontaux comme la partition 80-20, ou améliorant le storage management, le design des BD distribuées ainsi que les partitionnements verticaux essayant d'atteindre les mêmes objectifs ne devraient pas trouver leur place au niveau logique mais au niveau physique car ils n'ont pas de sens pour le programmeur d'application. Ces découpes sont faites pour le SGBD; elles découpent le TE en différentes facettes significatives pour le SGBD mais pas pour le programmeur.

Cependant, tant qu'aucun SGBD ne proposera ce genre de partitionnement, il faudra, si on veut absolument optimiser, reporter ces découpes au niveau logique et contraindre le programmeur à les utiliser.

Les découpes verticales (et horizontales) respectant une certaine sémantique peuvent par contre être envisagées sans restriction particulière.

Ceci étant dit, il nous semble plus opportun d'utiliser une approche basée sur la sémantique des traitements pour rechercher une possible partition et ainsi mettre en évidence différentes facettes du TE, pour ensuite, de la même manière que proposé pour la dénormalisation, optimiser physiquement les deux propositions et enfin choisir la meilleure (voir figure-11).

On pourra par exemple choisir une découpe (à tester par la suite) à l'aide d'un petit algorithme non déterministe.

Ces facettes du TE peuvent être considérées comme des **objets composants** du TE (dans le cas de la découpe verticale) ou **spécialisés** (dans le cas de découpe horizontale).

Il n'a cependant pas été jugé utile ou même remarqué au moment de l'analyse fonctionnelle que ce TE était composé d'objets plus petits. On peut dire qu'une agrégation (souvent inconsciente) ou une généralisation a été réalisée .

Donnons un exemple de TE **composé** de différents petits objets qui seront accédés par des traitements différents.

Soit le TE *Ligne_Com* reprenant les lignes des commandes passées chez une firme de vente de vêtements par correspondance (figure-17) :

Figure-17 : *Un TE ligne de commande semblable à celui que l'on pourrait trouver dans la BD d'une entreprise de vente par correspondance.*

Supposons que l'on doive effectuer les traitements suivants sur le schéma de la figure-17.

1. Après l'encodage des bons de commandes, faire un tri des lignes de commandes afin de rejeter les lignes comportant des incohérences,

2. Préparation journalière des factures,

3. Collecte d'informations pour provoquer le réapprovisionnement,

Les attributs utilisés par les traitements sont repris dans tab-6.

Même si ces trois traitements accèdent "conceptuellement" au TE *Ligne_Com*, ils n'ont pas besoin de la même information et accèdent tous trois à une **facette** particulière du TE.

	Id_L_C	Id_catalogue	Page	Description	Id_article	Taille	Quantité	Prix_u	Prix_Tot	Id_client
Cohérence	0	**1**	**1**	**1**	**1**	0	0	0	0	0
Facturation	**1**	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**
Réapprovision-	0	0	0	0	**1**	**1**	**1**	0	0	0

Tab-6 : *Attributs du TE Ligne_Com utilisés par les traitements.*

Si on partitionne verticalement le TE *Ligne_Com* comme indiqué à la figure-18, les trois traitements accédant essentiellement le TE *Ligne_Com* séquentiellement, verront leur performance améliorée. Le traitement vérifiant la cohérence ne devra accéder qu'au TE Description, les deux autres au TE *Ligne_Com* (nouveau *Ligne_Com*).

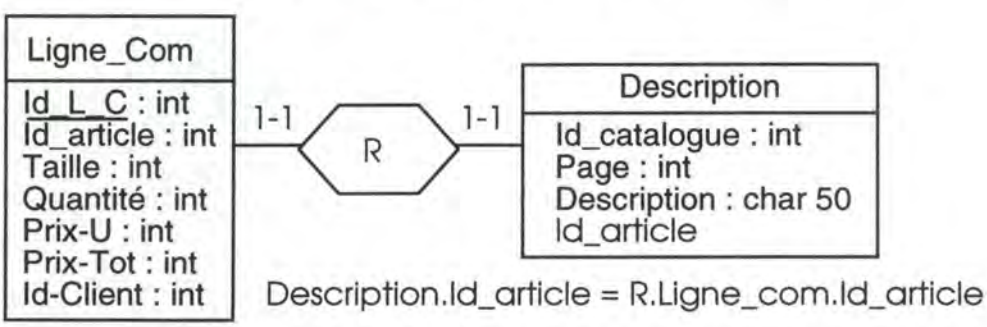


Figure-18 : *Découpe verticale du TE ligne_Com en suivant la sémantique des traitements.*

Il faut toutefois remarquer que nous avons introduit, afin que les traitements ne doivent pas accéder aux deux TE, une redondance structurelle que nous étudierons en 3.2.4. Nous avons en effet dupliqué l'attribut *Id_article* dans le TE Description.

Nous allons proposer un petit algorithme non-déterministe qui pourrait rechercher ce partitionnement qui semble intuitivement adéquat. Après avoir trouvé un partitionnement, il faudrait comme proposé à la figure-11, procéder au design physique des deux propositions et choisir celle qui apporte les meilleures performances.

Dans le cas particulier de l'exemple figure-18, il est sûr que les performances seront améliorées pour les traitements considérés, ceux-ci étant sans doute des traitements "batch" de gros fichiers et n'utilisant pas de clé d'accès. Il ne faudra donc pas passer par l'étape de design physique des solutions pour choisir le meilleur schéma (figure-11).

Un algorithme non déterministe de recherche du partitionnement pourrait être le suivant :

Soit TE l'ensemble des attributs du type d'entité que l'on veut partitionner, A(t) l'ensemble des attributs de TE utilisés par le traitement t et t(A), les traitements utilisant au moins un attribut de l'ensemble d'attributs A.

1. $E := \varnothing$
2. Choisir un traitement t
3. $E := A(t) \cup t(E)$
4. Si $E \neq TE$ alors on à une découpe possible : E. STOP.
5. Sinon, il faudrait de la redondance structurelle. Si pas redondance STOP.
6. Choisir l'attribut de TE le plus utilisé : r
7. $R := R \cup r$, $E := \varnothing$, $TE := TE / R$, GOTO 3

Après ces découpes de TE, nous allons maintenant aborder la découpe d'un attribut.

3.2.3. Découpe d'un attribut

Il existe sans doute un continuum entre la transformation de découpe d'un attribut et la transformation de partitionnement verticale. Cependant, nous l'avons abordée séparément car le but poursuivi n'est pas le même . En effet, rarement un seul attribut suffira à satisfaire un ou plusieurs traitements comme c'était le cas pour les fragments issus de la découpe verticale (voir exemple figure-18).

On découpera un attribut rarement accédé mais surtout de grande taille et/ou de taille pouvant fortement varier et sur lequel ne porte pas de recherches.

L'objectif de la découpe d'un attribut est en effet souvent :

- **L'augmentation du facteur de blocage**. La découpe d'un attribut, comme la découpe verticale augmente le facteur de blocage du TE. Comme nous le savons, l'augmentation du facteur de blocage est favorable aux lectures séquentielles. Pour des attributs des très grande taille, la découpe peut même éliminer le besoin de clé d'accès sur le TE, la lecture séquentielle du TE devenant moins chère que l'accès par clé (voir approximation des coûts physiques des traitements dans tab-3).
- **La gestion appropriée d'attribut souvent mis à jour et dont la taille peut varier fortement**. Ce sera, par exemple, le cas d'un attribut de pseudo-type *text*[1] qui peut être vide ou avoir une longueur de 1000 caractères. La mise à jour de tels attributs (quand les valeurs diffèrent grandement en taille) nécessite souvent pour les SGBD, de grands coûts de réorganisation interne qui ralentissent tous les accès. Il doit par exemple réorganiser la page qui ne peut plus contenir autant d'entités (vu l'augmentation en taille d'une des entités). Il faut alors peut-être réquisitionner une nouvelle extension du fichier contenant le TE, un appel au gestionnaire de fichier sera peut-être nécessaire, etc... La mise à l'écart de ces attributs permet une gestion appropriée[2] et séparée de ceux-ci.
- La **diminution de la place nécessaire au stockage** des informations. Si le domaine effectif[3] de l'attribut est restreint et n'évolue pas dans le temps, on pourra découper l'attribut "par les valeurs". La place nécessaire au stockage de ces valeurs est alors minimisée (figure-19).

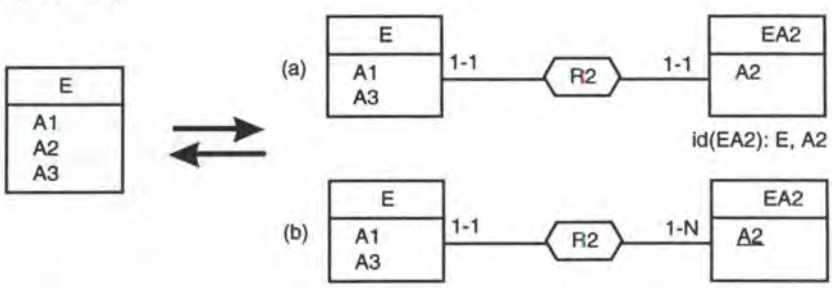


Figure-19 : *Transformation d'Extension d'un type d'entités par représentation des instances d'un de ses attributs (a) et par représentation des valeurs de ce même attribut (b).*

---

1 On pense par exemple aux types de données LONG ou VARCHAR2 (non standard) proposés par ORACLE. La place utilisée pour le stockage d'une valeur de ces types est variable.

2 On choisira par exemple pour les TE de tels attributs des paramètres physiques comme un PCTFREE élevé dans ORACLE V7.0.

3 Le domaine CHAR(30) est pratiquement infini mais le domaine des chaînes de caractères représentant les villes de Belgique est lui assez restreint.

3.2.4. Redondance structurelle

Une BD peut contenir des redondances, c'est-à-dire une duplication d'informations. Nous avons déjà rencontré des cas de redondance dans les deux paragraphes précédents. La DF de *Num_four* vers *Ville_four* dans le schéma de la Figure-1 apportait de la redondance et lorsque nous avons fait notre découpe verticale de Ligne_Com (voir figure-17 et 18), nous avons dupliqué l'attribut *Id_article* pour permettre à tous les traitements de trouver toute l'information qui leur était nécessaire dans un seul des fragments.

L'objectif principale de l'introduction de la redondance est la diminution du temps d'exécution de traitements effectuant une recherche. D'autres objectifs sont l'obtention d'une meilleure sécurité vis-à-vis des incidents (une information stockée à plusieurs endroits a moins de chances d'être perdue), le partitionnement de la base de en modules réorganisables de manière indépendante ou encore d'augmenter le parallélisme..

Les critères à l'origine de l'introduction de la redondance étant essentiellement de nature technique, il est normal que l'on introduise la redondance à ce niveau et non au niveau conceptuel. L'introduction de redondance est toutefois valable pour tous les types de SGBD.

La **duplication d'attribut**, la **composition d'association**, l'**aggrégation de données** (calcul d'une information à partir d'autres) sont quelques exemples de transformations introduisant de la redondance ***structurelle***. Par opposition, la dénormalisation introduit de la redondance non structurelle. La redondance se retrouve seulement ici au niveau des données. Ces transformations sont trivialement symétriquement réversibles.

Une redondance doit être explicitement décrite par une contrainte d'intégrité. Elle s'exprimera par une contrainte d'inclusion ou plus souvent par une contrainte d'égalité d'ensembles de données. Il faudra donc supporter cette contrainte d'intégrité supplémentaire. Cela se fera par l'intermédiaire des applications qui accèdent la BD ou de mécanismes proposés par certains SGBD. CODASYL propose par exemple des mécanismes spéciaux destinés à gérer la duplication d'attributs, la dérivation d'une valeur d'attribut à partir d'autres valeurs, etc...

Il va sans dire que ces transformations aboutissent à un schéma qui consommera plus de place de stockage et que la gestion de la contrainte d'intégrité supplémentaire ralentira les insertions et les suppressions.

Voyons quelques scénari pour fixer les idées.

- Soit le traitement Trt-6 : *Donner les Editeurs des OUVRAGEs dont un EXEMPLAIRE est repris dans les EMPRUNTs dont la Date_retour est antérieure à aujourd'hui,* et le schéma A de la Figure-20. L'exécution de Trt-6 nécessite l'utilisation des chemins d'accès unissant EMPRUNT à EXEMPLAIRE et EXEMPLAIRE à OUVRAGE pour enfin (et seulement) accèder à *Editeur*. La **duplication de l'attribut** *Editeur* de OUVRAGE dans EMPRUNT (Schéma B) diminuerait fortement le nombre[1] d'accès logiques à la BD, donc sans doute le nombre d'accès physiques et le temps d'exécution. Cela diminuera la "navigation" inter-table.

[1] Pour chaque entité EMPRUNT de *Date_fin* antérieure à aujourd'hui, on effectura un accès à EXEMPLAIRE et un dernier accès à OUVRAGE. En dupliquant Editeur dans EMPRUNT, aucun accès supplémentaire n'est nécessaire.

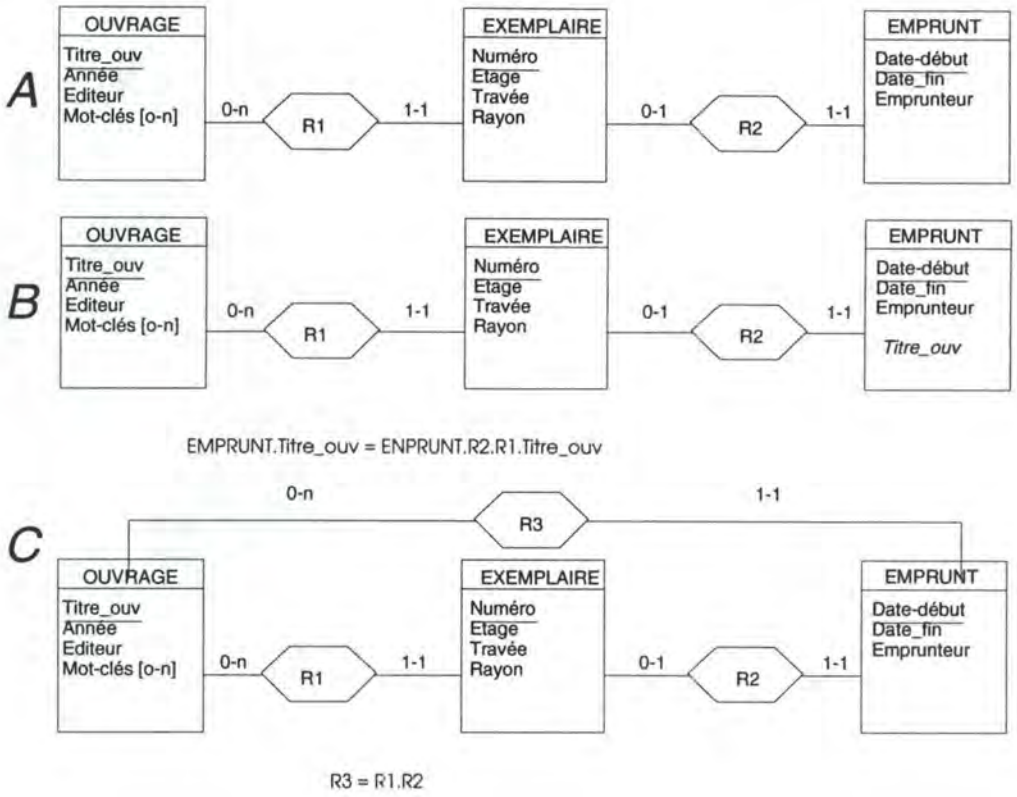


Figure-20 : *Schéma A, pas de redondance structurelle, schéma B, l'attribut éditeur de EMPRUNT est redondant, schéma C, R3 est redondante. Ces schémas sont sémantiquement équivalents.*

Si il s'avère que beaucoup de traitements doivent suivre la même suite de chemins d'accès pour finalement accéder à différents attributs de OUVRAGE, il est alors peut-être plus raisonnable au lieu de dupliquer dans EMPRUNT tous les attributs de OUVRAGE accédés, d'ajouter une association R3 redondante avec la **composition d'association** R2.R1. comme on peut le voir sur le schéma C.

Si un traitement a besoin d'une information dans un TE et donc de suivre un chemin d'accès pour l'obtenir, on hésitera d'autant moins à alouter de la redondance si cette information n'est nécessaire que pour la vérification d'une condition comme c'était le cas pour Trt-2.

- Il pourra parfois même être intéressant de dupliquer non pas un ou plusieurs attributs mais un TE tout entier.

A la base de cette duplication on peut bien sûr trouver des SGBD distribués[2], la volonté d'améliorer le parallélisme mais aussi le désir d'avoir **différentes organisations physiques des données**. On ne peut en effet qu'imposer un seul ordre de rangement des entités d'un TE. Pourtant, parfois différents rangements pourraient être intéressants.

Prenons l'exemple des botins téléphoniques (les pages jaunes et le traditionnel). Dans les pages jaunes, les entités sont groupées, ordonnées, par secteur d'activité. Nous avons une

2 Chacun aimerait avoir la totalité des informations chez lui et ne pas dépendre des autres sites, on peut choisir de dupliquer l'information dans différents sites.

clustering key[3] : *Activité*. On recherche le numéro de téléphone d'un abonné appartenant à un secteur d'activité.

Dans le botin traditionnel, les entités sont rangées par ordre alphabétique sur *Nom*. Les abonnés des pages jaunes sont pourtant repris (en partie) dans le botin traditionnel, il y a donc redondance. Cependant, les modes d'accès sont, de part leur rangement respectif, optimisés.

Quelle sera la différence si on veut retrouver les abonnés nommés Dupont ? On ne peut évidemment pas faire la différence entre les performances en terme d'accès logiques, mais bien en terme d'accès **physiques**.

Comme les abonnés ayant des noms semblables seront rangés de manière contiguë, avec un accès physique on récupèrera plusieurs entités ayant le même nom vu qu'une page contient plusieurs entités qui nous intéressent. Par contre, si l'ordre était celui des secteurs d'activité, il faudrait presque autant d'accès physiques que d'accès logiques. Nous avons schématisé sur la Figure-21, la plus grande efficacité physique d'un accès sur la clé *Activité* que sur la clé *Nom*.

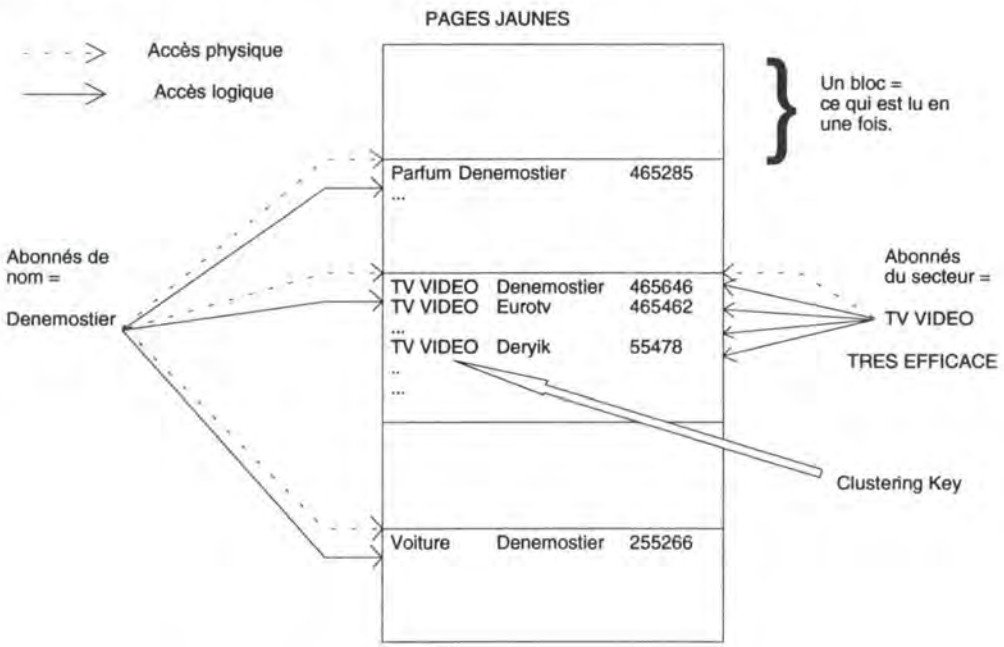


Figure-21 : *Les entités sont rangées par secteur, l'utilisation de la clé Nom n'est alors pas très efficace.*

Remarquons que les pages jaunes ne sont pas dans la réalité une simple réorganisation du botin traditionnel mais plutôt la fusion d'un fragment (d'une découpe horizontale) du botins traditionnel et d'un TE comprenant des entités ne figurant pas dans le botin traditionnel.

• Voyons enfin un exemple d'agrégation de données. Supposons que le département comptable d'une chaîne de distribution *demande toutes les 20 minutes le montant total des ventes réalisées par chaque vendeur* (Trt-7) et *par centre regroupant ces vendeurs* (Trt-8). Une vente entraîne *l'insertion d'une entité dans un TE VENTES* (Trt-9) qui contient les ventes de la journée. Le premier schéma de la Figure-22 est le schéma sur lequel portent ces différents traitements.

---

3 Clustering key = "clé regroupante". Les numéros de téléphone des abonnés ayant même valeur de clé (même activité) sont groupés sur des pages consécutives.

Trt-7 et Trt-8 vont prendre beaucoup de temps. Il faudra par exemple pour Trt-7 accéder à chaque vente d'un vendeur, et accéder à PRODUIT pour connaître le prix du produit vendu et ceci pour chaque vendeur. On le voit, ces traitements sont très longs et en plus vont perturber le cours normal des insertions. Ces traitements utilisent en effet les mêmes ressources.

Dénormaliser (mettre PRODUIT dans VENTES) diminuerait les accès logiques réalisés par Trt-7 et Trt-8 mais la modification d'un prix et les insertions deviendraient très coûteuse. Par contre, ajouter un attribut *Total_ventes* dans VENDEURS et dans MAGASINS, qui serait le total des ventes réalisées par un vendeur ou un magasin et qui serait mis à jour lors de chaque insertion, réduirait Trt-7 et 8 à la simple consultation de cet attribut. On *distribue* ici le travail de Trt-7 et 8 sur Trt-9.

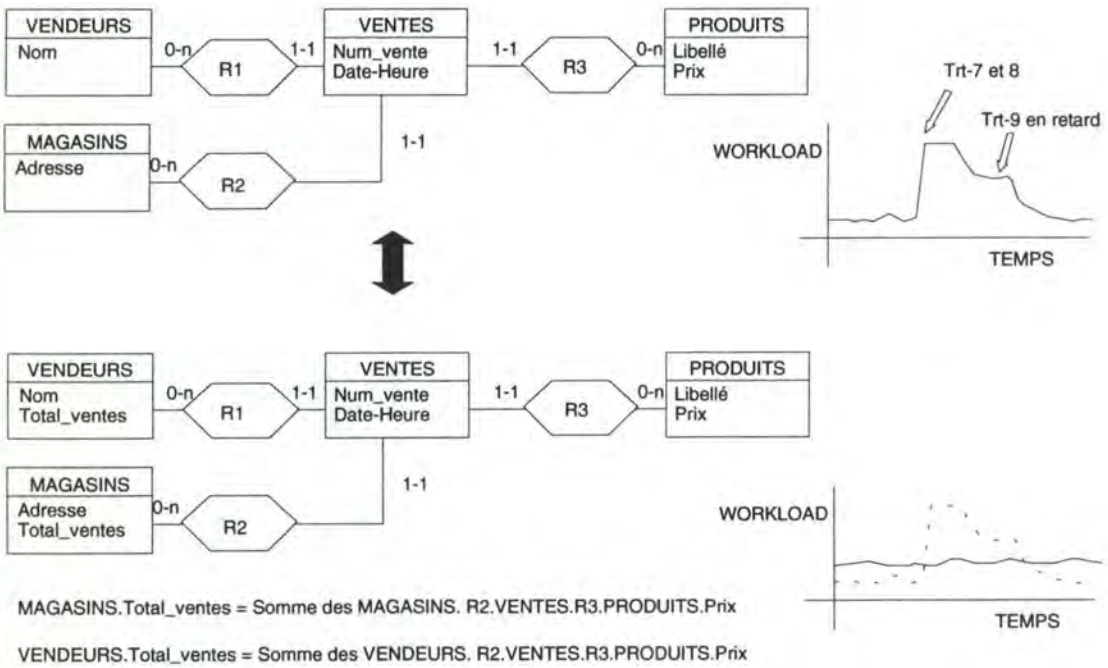


Figure-10 : *Agrégation de données dans le second schéma. Schématisation d'une prévision intuitive du workload pour les deux schémas.*

Les temps d'exécution de Trt-7 et 8 seraient plus courts et ne gêneraient que très faiblement le flot des Trt-9, chaque Trt-9 devenant lui, plus coûteux.

3.3. Conformation au modèle relationnel et optimisation

Comme nous l'avons dit en 2.3, les structures de données du modèle relationnel sont particulièrement restreintes.

En plus des associations many-to-many, des associations de degré supérieur à deux et des associations avec attributs supportées par aucun SGBD, et de ce fait enlevées lors de la simplification du schéma conceptuel (voir 3.1), il nous faut maintenant transformer le schéma logique afin de supprimer les **attributs multivalués** et/ou **décomposables** et les **types d'associations one-to-many** non conformes au modèle relationnel. On pensera aussi à l'ajout d'informations techniques.

Certaines transformations de ces concepts non conformes apporteront plus de performances que d'autres comme nous le verrons. Nous n'avons pas repris toutes les traductions possibles et imaginables, mais seulement les plus fréquentes et celles qui ont un certain intérêt pour la recherche de l'efficacité de la base de données.

3.3.1. Association one-to-many (one-to-one)

Il n'y a pas une grande variété de façon d'éliminer un type d'association one-to-many. La première transformation est la plus classique.

On remplacera simplement le type d'association par un attribut de référence ajouté dans le type d'entité du côté Many et en ajoutera au schéma une contrainte référentielle[1] (Figure-1 I).

Nous n'envisageons pas la seconde transformation présentée à la figure-1 II, elle engendre en effet un attribut multivalué qu'il faudrait par la suite supprimer.

Si aucun des types d'entité n'est tenu de jouer un rôle dans une association, on verra apparaître un attribut de référence facultatif. Or, qui dit facultatif dit valeur NULL. Il faudra, dans ce cas, faire attention à la formulation de certaines requêtes ou bien utiliser la déclaration "Foreign Key" si l'on ne veut pas avoir de dégradations "insoupçonnables" des performances (voir 3.4).

Par exemple avec le traitement suivant appliqué au schéma de droite de la figure-1 III :

```
SELECT      B.b2
FROM        A, B
WHERE       A.a1 = w
AND         A.b1 = B.b1
```

et si la stratégie de l'optimiseur est la suivante (nested loop), il pourrait en effet y avoir des accès physiques inutiles :

---

[1] Cette contrainte se traduira pratiquement par une Foreign Key, (un trigger, ou un check) Si aucun de ces mécanismes n'est utilisé, c'est dans la dynamique de l'application qu'elle sera enfouie.

- D'abord sélectionner les couples (a1, b1) de A qui vérifient le prédicat (a1=w) (sélection S1),
- ensuite, pour chaque couple retenu (b1 pouvant être NULL), **sélectionner** (sélection S2) **l'élément de B** (un couple) **vérifiant le prédicat (B.b1 = A.b1) où A.b1 est la valeur d'un couple retenu par la sélection S**, etc...

Le critère de sélection S2 (**en gras**) a en effet une sélectivité égale à 0 lorsque b1 est NULL (B.b1 étant un identifiant, il ne peut avoir la valeur NULL[2]). La seconde sélection entraînerait donc une lecture séquentielle de tout[3] le type d'entité B comme on le mentionnera en 3.4, mais une lecture qui ne sert à rien.

Si par contre, l'attribut de référence est bel et bien déclaré comme une Foreign Key référençant une Primary Key (qui est déclarée implicitement NOT NULL), l'optimiseur (de la plupart) des SGBD-R est alors assez intelligent pour ne pas faire cette recherche. On aurait tout aussi bien pu écrire le traitement de la manière suivante afin d'éviter les pertes de performances :

```
SELECT      B.b2
FROM        A, B
WHERE       A.a1 = w
AND         B.b1 = A.b1
AND         A.b1 not NULL
```

Il n'y avait alors aucun problème.

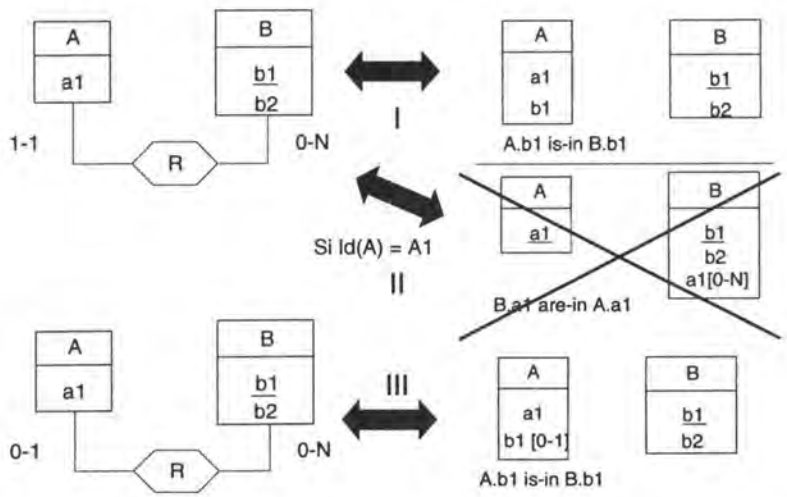


Figure-1 *: Transformation d'une relation one-to-many en un attribut de référence et une contrainte référentielle (I), en un attribut de référence multivalué (II), en un attribut de référence facultatif (III).*

Pour la traduction des relations **one-to-one**, on peut faire un choix qui pourra influencer très légèrement les performances. Si les deux types d'entités unis par le type d'association one-to-one sont munis de leur propre identifiant (pas identifié par l'autre type

2 Les composants d'un index primaire PRIMARY KEY ne peuvent être NULL [DAT83].

3 Une lecture séquentielle de **tout** le TE car, soit le TE n'est pas muni d'une clé d'accès sur B.b1 et dans ce cas, il faudra lire jusqu'à la dernière entité pour se rendre compte que cette valeur (NULL) n'y figure pas, soit le TE possède une clé d'accès mais l'optimiseur refusera de l'employer vu la comparaison avec NULL (voir 3.4).

d'entité), on peut choisir dans quel type d'entité localiser l'attribut de référence (voir figure-2 I et II).

Le choix pourra être guidé par la considération suivante : l'attribut de référence prenant de la place, le facteur de blocage diminuera pour le type d'entité qui supportera cet attribut et une lecture séquentielle de ce type d'entité deviendra donc plus coûteuse. On ajoutera donc cet attribut au type d'entité sur lequel on fera le moins de lectures séquentielles.

Il faut cependant remarquer que cet attribut de référence sera du même type que l'identifiant primaire de l'autre type d'entité qui comme la plupart des identifiants primaires n'est pas très coûteux en place. Un identifiant est en effet souvent un numéro (entier) et rarement un string[40].

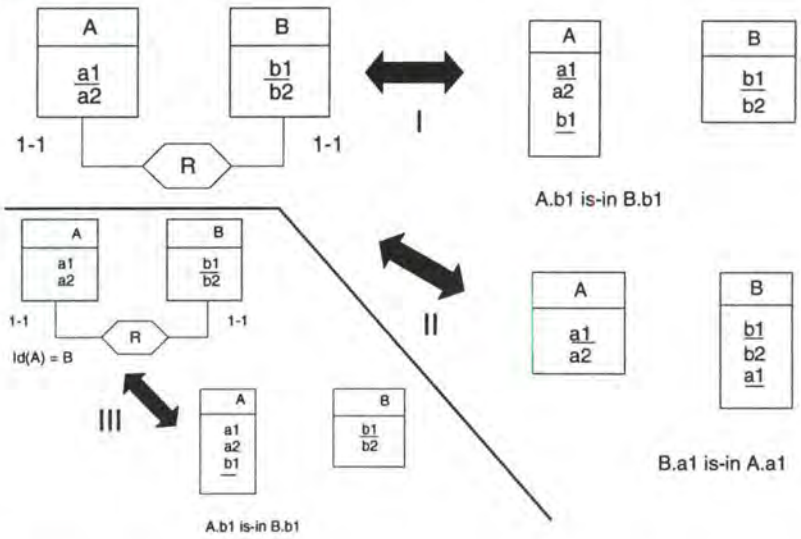


Figure-2 : *Elimination d'une association one-to-one en la remplaçant par un attribut de référence d'un côté ou de l'autre (I et II) ou du côté sans identifiant si les types d'entités n'en possèdent pas chacun un (III).*

Si un des rôles de la relation one-to-one est facultatif, on préfèrera la traduction présentée à la figure-3 I. Cette traduction évite en effet des problèmes rencontrés pour la traduction d'une relation one-to-many dont nous venons de parler. De plus, l'attribut de référence facultatif généré par la traduction de la figure-3 II sera implémenté par certain SGBD comme un attribut en longueur variable (comme tout attribut facultatif). Or, les attributs en longueur variable peuvent être la cause d'autoréorganisation très coûteuse.

Cependant, un traitement accédant B et voulant uniquement savoir si B est associé à une entité A serait bien évidemment avantagé par le schéma II de la figure-3, il ne devrait en effet pas faire de recherche dans A[4]. Il en serait de même d'un traitement accédant A et se "contentant" de connaître l'identifiant de l'entité du type d'entité B qui lui est associé ave ke schéma I de la figure-3.

4 Cet avantage serait d'ailleurs non négligeable même si le type d'entité A possède un mécanisme d'accès sur A.b1.

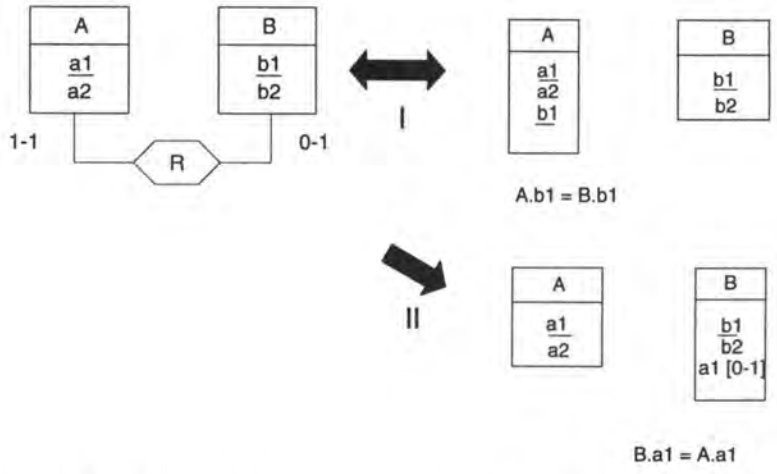


Figure-3 : *Elimination d'une relation one-to-one dont un rôle est facultatif. La solution II génère des valeurs NULL qui peuvent être la source de problèmes.*

Il existe enfin une dernière manière de traduire un type d'association one-to-many ou one-to-one : la traduction utilisée pour éliminer les associations many-to-many.

Cette traduction est sans doute **la plus mauvaise à court terme**, elle nécessite en effet un *join* supplémentaire lorsque l'on veut récupérer de l'information contenue dans les deux types d'entités d'origine (figure-).

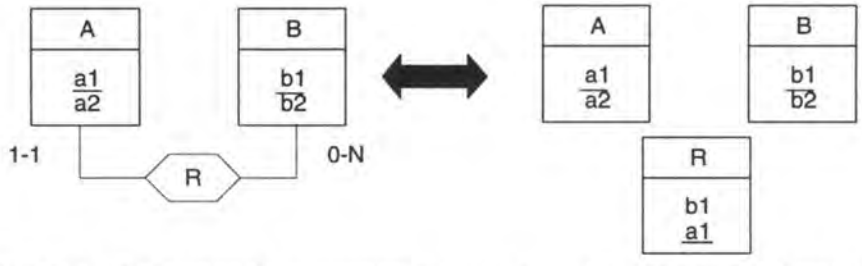


Figure-4 : *Elimination d'une association one-to-many à l'aide d'un type d'entité.*

Le seul avantage de cette traduction est la facilité de maintenance de l'application. Si par la suite le type d'association gênant et initialement perçu comme une association one-to-many se métamorphose en une association many-to-many, cette dernière traduction acceptera facilement cette évolution des spécifications, c'est-à-dire sans modification des données ou du schéma de la base.

3.3.2. Attributs multivalués

La première manière d'éliminer un attribut multivalué d'un type d'entité TE est d'isoler cet attribut dans un type d'entité distinct et de relier ce type d'entité à TE par une relation one-to-many comme illustrée à la figure-5. On supprimera ensuite la relation one-to-many comme expliquée au point précédent. Toutes les valeurs de l'attribut multivalué de toutes les entités de TE sont alors reprises dans le nouveau type d'entité. On appellera cette représentation la **représentation par instanciation**.

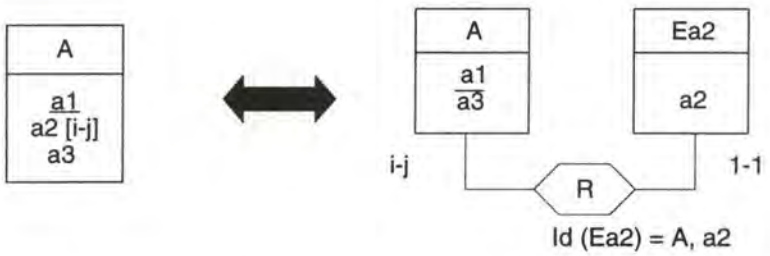


Figure-5 : *Elimination d'un attribut multivalué par représentation des instances.*

Une seconde manière de faire est la **représentation par les valeurs** (figure-6). Seules les valeurs distinctes des attributs multivalués de toutes les entités de TE seront représentées.

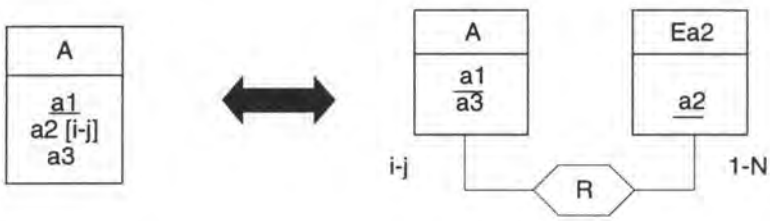


Figure-6 : *Elimination d'un attribut multivalué par représentation des valeurs.*

Une troisième manière de faire est de remplacer l'attribut multivalué par plusieurs attributs simples mais de domaine identique (figure-7), c'est-à-dire **par juxtaposition** d'attributs.

Figure-7 : *Elimination d'un attribut multivalué par juxtaposition d'attributs simples.*

Une dernière manière de faire est de concaténer les valeurs de l'attribut multivalué pour ne plus former qu'une seule valeur d'un attribut portant sur un autre domaine (Figure-8).

Figure-8 : *Elimination d'un attribut multivalué par concaténation.*

Chaque traduction a ses propres avantages et inconvénients. Le choix entre l'une ou l'autre solution sera guidé par les cardinalités minimale et maximale `[min-max]` de l'attribut multivalué ou même par la distribution des nombres de valeurs de l'attribut multivalué (a) pour une entité (E), c'est-à-dire $N_{a/E}$. On peut voir un exemple de distribution des nombres de valeurs à la figure-9.

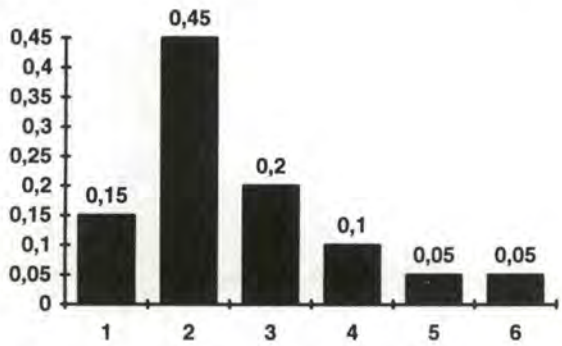


Figure-9 : *Une distribution du nombre de valeurs d'un attribut multivalué ayant [min-max] = [1-6] pour les entités d'un type d'entité. On voit par exemple que 45% des entités ont deux valeurs pour l'attribut multivalué.*

On pourrait aussi sans aucun doute faire un modèle d'optimisation donnant la meilleure traduction de l'attribut multivalué basée sur ces statistiques mais la collecte de la distribution des $N_{a/E}$ (pendant l'analyse d'opportunité) n'est sans doute pas très réaliste, c'est pourquoi nous restons au niveau de l'intuition.

Si `max` est très grand, la troisième solution entraînera de très **mauvaises performances pour les lectures séquentielles**[5] du type d'article.

Si le nombre moyen de valeurs est plus proche de `min` que de `max`, il y aura alors "**gaspillage de place**", certains attributs simples n'ayant pas de valeur (ou une valeur NULL).

---

5 Ce phénomène est maintenant un refrain bien connu, il y a en effet diminution du facteur de blocage due à l'augmentation de la taille des entités et donc augmentation du coût d'une lecture séquentielle. On pourrait alors penser que si l'on s'inquiète du fait que la somme des tailles des attributs simples se substituant à l'attribut multivalué, fait diminuer le facteur de blocage et donc détériore les performances des lectures séquentielles, on aurait, de même, dû s'inquiéter de ce phénomène pour l'attribut multivalué et, donc peut-être, le découper verticalement de son type d'entité.

Cependant, la taille de l'attribut multivalué ne peut être estimée qu'en moyenne et cette moyenne pouvait être assez basse que pour ne pas nous inquiéter. Par contre, la taille de la "juxtaposition" des attributs simples est, elle, bien constante et est égale à `max` fois la taille (peut-être de surcroît variable) d'une valeur de l'attribut multivalué, cette taille pouvant être fortement au-dessus de la moyenne et ainsi nous inquiéter.

Cette solution est par contre idéale si `min` et `max` sont proches et peu élevés comme `[2-2]`. Dans ce cas, le facteur de blocage ne diminuera pas trop et le gaspillage de place ne sera pas trop élevé.

De plus, lorsqu'on a accédé à une entité du type d'entité, **aucune navigation supplémentaire** ne sera nécessaire pour connaître les valeurs de l'ancien attribut multivalué.

Ce ne serait pas le cas pour les deux premières solutions.

Cependant, une condition sine qua non à l'utilisation de cette solution sera la connaissance de `max`, il ne peut être inconnu (= N).

Un autre avantage de cette troisième solution est qu'elle peut conserver un éventuel **ordre sur les valeurs** de l'attribut multivalué.

La dernière solution (concaténation) permet elle aussi de garder **un ordre** sur les valeurs, et sera bien adaptée à une situation où la distribution des probabilités $P(N_{A/E}=x)$ est uniforme (ou si la variance de la distribution est forte) mais aussi où le nombre de valeurs différentes pour cet attribut ne varie pas fortement dans le temps. Si il y avait évolution dans le temps du nombre de valeurs de l'attribut multivalué, la taille de l'attribut qui le remplacerait dans le schéma conforme au relationnel (la concaténation des valeurs) varierait-elle aussi[6], ce qui nécessiterait des réorganisations[7] très coûteuses.

Elle sera cependant évitée si on doit faire une recherche basée sur les valeurs de l'attribut multivalué. La troisième solution (juxtaposition) permet elle, les recherches sur les valeurs d'attributs mais, à moins que l'on accepte de faire une recherche séquentielle, les performances seront très mauvaises si l'on ne sait pas sur quel attribut effectuer la recherche et donc en quelque sorte profiter de l'ordre de ces attributs.

Ni la première solution ni la seconde ne peuvent être envisagées si l'ordre des valeurs de l'attribut multivalué est significatif. Elles entraîneront en outre une (des) navigations supplémentaire(s) pour connaître les valeurs des attributs du type d'entité, ce qui les rendra peu efficaces si `min` et `max` sont très petits, et surtout si les valeurs de ces attributs sont souvent requises par des traitements accédant le TE (c'est le même type de problème que la navigation supplémentaire impliquée par la dénormalisation).

Elles supporteront par contre indifféremment toute distribution du nombre de valeurs par entité et supporteront facilement des évolutions de cette distribution, `max` pouvant être égal à N, c'est-à-dire à une valeur inconnue et aussi grande que l'on veut.

Le stockage des différentes valeurs de l'attribut multivalué dans un type d'entité distinct améliorera bien évidemment les accès séquentiels sur le type d'entité d'origine (le facteur de blocage augmentant).

La première solution sera utilisée quand les valeurs rencontrées sont "plus ou moins identifiantes" comme *adresse* à la figure-10 A.

---

6 La concaténation sera logiquement implémentée comme un attribut en longueur variable. En effet, si on implémentait la concaténation comme un attribut en taille fixe, cette solution n'aurait aucun avantage sur la troisième solution, gaspillant autant de place (longueur fixe réservée) et nécessitant de toute façon un coût de "déconcaténation" par l'application lors de la lecture d'une valeur.

7 La variation en taille des attributs provoque de lourdes réorganisations physiques. On imagine en effet très bien le problème qui se pose lorsqu'une entité contenue dans une page physique devrait s'étendre et qu'il n'y a plus de place dans la page. Certains SGBD essaient de pallier à ce problème en proposant la spécification de certains paramètres physiques (PCTFREE de ORACLE par exemple), mais ceci se fait souvent au détriment des performances des recherches d'information.

Par contre, la seconde solution sera surtout employée si la cardinalité de l'ensemble des valeurs effectivement présentes dans l'attribut est relativement petite (composants dans la figure-10 B).

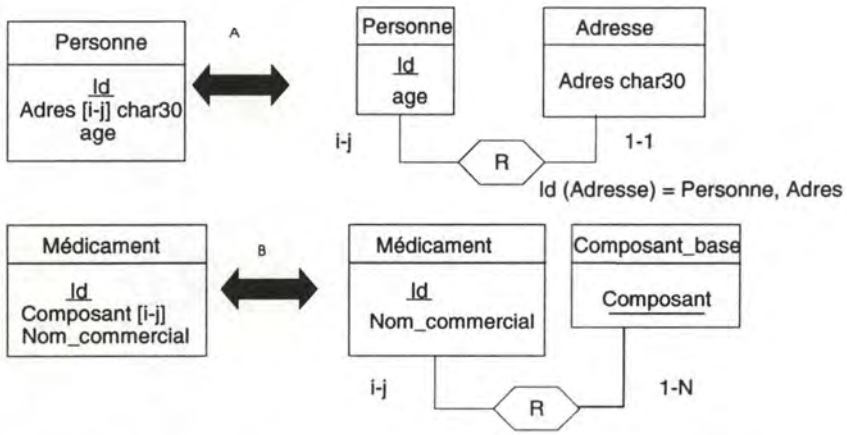
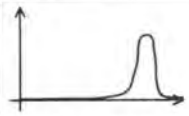


Figure-10 : *Représentation des instances des adresses des personnes (A); Représentation des valeurs des composants de base de médicaments (B).*

On peut dire que la première solution est sans doute la plus générale.

On peut donc dire en résumé :

	Instances	Valeurs	Juxtaposition	Concaténation
 Distribution avec forte variance.	Peut convenir, accès séquentiels au TE A de la figure-?? optimisés, augmentation du facteur de blocage oblige.	Peut convenir. Accès séquentiels également très performants, et **très faible consommation** de place si du moins l'implémentation de l'association n'est pas trop goumande.	Entraîne un **gaspillage de place** substantiel dû à la réservation de place même pour les entités n'ayant pas beaucoup de valeurs. Diminution des performances des accès séquentiels au TE A.	Peut convenir. Performances des accès séquentiels dans la moyenne
 Faible variance, moyenne basse	Peut convenir. Remarques identiques à la distribution uniforme	Peut convenir. Remarques identiques à la distribution uniforme	Totalement **inadaptée**, gaspillage de place énorme.	Peut convenir.
 Faible variance, moyenne élevée.	Peut convenir. Remarques identiques à la distribution uniforme	Peut convenir. Remarques identiques à la distribution uniforme	Peut convenir. Meilleure des trois distributions pour cette solution.	Peut convenir.
Ordre des valeurs important	**Inutilisable** en restant au niveau logique. Il faudrait imposer un ordre physique correspondant à l'ordre logique désiré par le biais d'une Clustering Key par exemple.	Inutilisable, à moins d'ajouter dans l'association many-to-many un attribut donnant le numéro d'ordre.	Peut convenir.	Peut convenir.
Recherche basée sur les valeurs de l'attribut multivalué	Performances acceptabes mais nombreuses navigations, voir 3.2.1.4.	Plus mauvaises que la solution par les instances vu que cette solution génère une association Many-to-Many qu'il faut implémenter en une table et qui provoque donc beaucoup de navigation	Il faut connaître la position de l'attribut dans la liste ordonnée des attributs sur lequel on veut faire la recherche. Si on ne connaît pas la position, il faudra utiliser la recherche séquentielle et le mot réservé UNION de SQL.	Totalement **inadéquat** pour de telles recherches, même séquentielles La formule de concaténation n'est en effet pas connue du SGBD mais de l'application. On ne saurait donc employer la clause WHERE de SQL.

Obtention des valeurs de l'attribut multivalué.	Navigation importantes	Navigations très importantes	Très très bonnes performances	Très bonnes performances
Evolution temporelle notable de la distribution (insertions et suppressions)	**Solution la plus souple**, sans aucune réservation inutile de place. Seul l'attribut de référence consomme de la place	Solution souple mais nécessite des vérifications supplémentaires lors des insertions, il ne retient en effet qu'une seule fois chaque valeur, ce qui nécessite le respect de la contrainte d'identification. de Ea2 de la figure-??	La distribution peut évoluer vers une des trois distributions génériques ci-dessus et donc s'avérer plus ou moins bien adaptée. Il n'y a cependant pas de maintenance (insertions, suppressions ou mises à jour d'attributs en tailles variables)	Les mises à jour des attributs en taille variable peuvent nécessiter des autoréorganisations des pages du SGBD très coûteuses ou l'adoption de mesures non propices aux lectures séquentielles. Cette solution est donc **à éviter** dans ce cas.
Containtes d'intégrité portant sur l'attribut	On tombe ici dans le problème de fission des contraintes d'intégrité. A fortement déconseiller	A fortement déconseiller	Convient	Ne convient pas
Cardinalité maximum inconnu	Peut convenir.	Peut convenir.	**Ne convient pas.**	Peut convenir.

Tab-1 : *Tableau comparatif des avantages et inconvénients des trois solutions envisagées pour l'élimination des attributs multivalués.*

Remarquons encore que l'on peut **mixer les solutions**. Supposons par exemple que l'on ait le cas suivant :

```
[min-max] = [2-20], les probabilités d'avoir 2, 3, ..., ou 20 valeurs
pour l'attribut sont égales (distribution uniforme, grande variance) et
l'évolution de cette distribution est inconnue.
```

La traduction suivante sera alors avantageuse (figure-11) :

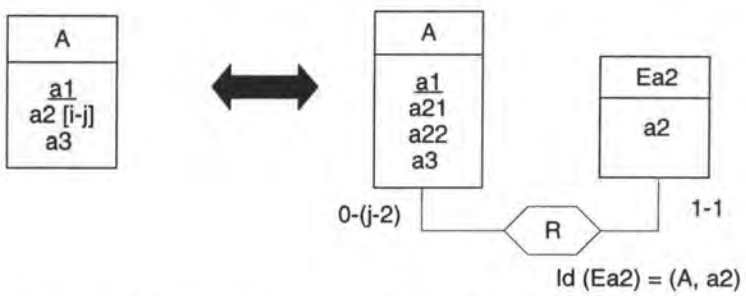


Figure-11 : *Traduction d'un attribut multivalué mixant plusieurs solutions de base.*

Elle permet d'avoir accès directement à deux attributs qui pourront peut-être suffire à la plupart des traitements (pas de navigation supplémentaire dans beaucoup de cas), permet de gérer facilement les évolutions des nombres de valeurs par entité, et permet de grandes variances (une entité peut avoir 20 valeurs qui lui sont associées alors qu'une autre n'en aura que deux, et cela sans gaspiller de place avec des valeurs NULL ou au prix de grandes réorganisations lors de l'association d'une nouvelle valeur à une entité.

3.3.3. Attributs décomposables

Il y a essentiellement deux techniques d'élimination des attributs décomposables non conformes au modèle relationnel, la **décomposition** (figure-12) de l'attribut, et la **concaténation** de ses composants. La concaténation est semblable à la technique utilisée à la figure-8 pour les attributs multivalués. Pour la décomposition, on remplace l'attribut composé par ses composants qui deviennent alors des attributs de premier niveau.

On peut dire que la concaténation n'a rien pour elle dans ce cas, si ce n'est qu'elle simplifie quelque peu le schéma et donc le catalogue du SGBD et surtout qu'elle met bien en évidence la notion d'agrégat. On ne saura par contre pas utiliser de critère de sélection basé sur un des composants, la concaténation étant gérée par l'application. Supposons par exemple que nous ayons l'attribut décomposable *adresse (rue, numero, code_postal),* du TE Client. Pour retrouver tous les clients ayant un certain code postal, il faudra lire client après client et décomposer (à l'aide de l'application) la concaténation *adresse_concat* remplaçant l'attribut composé.

Si on ne sait pas faire de sélection à l'aide d'un mécanisme d'accès, ce sont les performances des SELECT, DELETE et UPDATE qui s'en ressentiront.

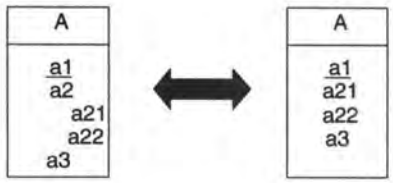


Figure-12 : *Elimination d'un attribut composé par décomposition.*

3.3.4. Ajout d'informations techniques

Des structures techniques **redondantes** peuvent être ajoutées de manière à accélérer certaines requêtes. Par exemple, la recherche des a2 associés aux entités de A vérifiant un certain critère (figure-11) nécessite de faire une jointure sur de A et Ea2. On écrira par exemple en SQL :

```
SELECT      Ea2.a2
FROM        A, Ea2
WHERE       A.a3= :x ;
```

Or, cette opération de jointure échouera peut-être dans 80 % des cas, les entités de A n'étant que rarement associées à des entités de Ea2. Elle consommera cependant plusieurs accès physiques avant de constater cet échec[8]. On peut éviter cette jointure lorsqu'il n'y a pas de Ea2 associé en ajoutant à la table A une colonne Ass_Ea2 indiquant si oui ou non une entité de A est associée à au moins une entité de Ea2. On pourra alors écrire le traitement suivant :

```
SELECT      Ea2.a2
FROM        A, Ea2
WHERE       A.a3= :x
AND         A.Ass_Ea2 = true ;
```

Cependant, comme avec tout ajout de redondance, le gain obtenu pour les opérations de consultations se répercute en ralentissement des insertions et suppressions. Chaque fois que l'on insèrera ou supprimera une entité dans Ea2, il faudra vérifier et modifier la valeur de Ass_Ea2 si nécessaire. Ces vérifications pourront se faire via les checks ou les triggers.

Cette technique ne concerne évidemment pas que les attributs multivalués mais toutes les associations. Elle sera d'autant plus intéressante que la sélectivité de "Ass_Ea2 = false" est forte.

8 Ces accès inutiles sont caractéristiques des SGBD-R. Ils sont absents par exemple des SGBD CODASYL ou orientés-objets.

3.4. Optimisations des requêtes SQL

Dans les SGBD-R, à aucun moment, un utilisateur n'a à connaître la représentation physique des informations, ce n'est, par contre, pas le cas pour les SGBD CODASYL ou hiérarchique. Cette séparation entre physique et logique constitue une force des SGBD-R.

D'autre part, le Langage de Manipulation de Données (LMD) prédicatif SQL, définit le résultat d'une requête, ce que l'on veut obtenir, sans jamais spécifier comment faire pour l'obtenir. De plus, l'ordre dans lequel sont spécifiés les différents prédicats d'une requête est non significatif. La même requête peut avoir une très grande variété de formulations toutes équivalentes **quant au résultat exprimé**.

Puisque l'utilisateur ne connaît pas la structure de stockage et ne spécifie pas la manière d'exécuter ses requêtes, c'est le SGBD-R qui doit prendre en charge le passage de la représentation relationnelle à la structure de stockage et la transformation des requêtes en une suite d'opérations appliquées aux informations stockées. Cette suite d'opérations constitue la stratégie d'exécution de la requête (query plan).

Plusieurs solutions sont possibles pour répondre à une même requête. Le SGBD-R doit choisir la solution de coût minimum (en termes d'E/S et éventuellement en temps de traitement UC et espace MC consommé). Ce choix est pris en charge par "l'optimiseur" du SGBD-R.

Prenons un exemple pour illustrer l'importance du travail réalisé par "l'optimiseur".
Soit les relations (figure-1) :

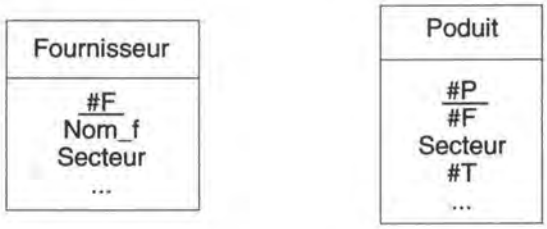


Figure-1 : *Un schéma relationnel composé de deux relations, #F est une Foreign Key*

avec 10000 produits de 50 différents types (#T), 100 fournisseurs et soit la requête suivante faite par l'utilisateur :

```
-- Nom des fournisseurs fournissant le produit 'P2'.

SELECT      Nom_f
FROM        fournisseur, produit
WHERE       produit.#F = fournisseur.#F
and         produit.#T = 'P2'
;
```

Supposons que le facteur de blocage pour chaque relation soit de 1 (on aura donc un accès physique pour un accès logique), qu'aucun index ne soit présent et que l'on ne dispose pas de grands buffers (les relations ne savent donc pas tenir en MC).

Plusieurs stratégies globales d'exécutions sont possibles, en voici deux :

((fournisseur (#F) * produit (#F)) (#T = 'P2')) [Nom_f] -St_1-

c'est-à-dire faire la jointure (*) sur #F, puis filtrer le résultat pour ne retenir que les lignes ayant la caractéristique (#T = 'P2') et enfin projeter sur Num_f
ou
((produit (#P = 'P2') (#F)) * (fournisseur (#F))) [Nom_f] -St_2-

c'est-à-dire d'abord sélectionner les produits tels que (#T = 'P2'), faire la jointure (*) sur #F avec le TE fournisseur et enfin projeter le résultat sur Nom_f. []

Avec St_1, on fait d'abord la jointure. Pour cela, on lira les 10.000 lignes de la table PRODUIT et pour chacune de ces lignes, on devra trouver la ligne de FOURNISSEUR qui lui est associée (par la FK #F). La recherche de cette ligne coûtera (100+1)/2 accès en moyenne pour chaque ligne de PRODUIT. Jusque-là, cela nous aura coûté 10.000+**10.000x**50,5 = 515.000 accès. Le résultat de cette jointure est en fait une table contenant 10.000 lignes, elle ne peut pas tenir en mémoire. Il aura donc fallu faire 10.000 écritures. Il faudra ensuite relire ces 10.000 lignes pour n'en sélectionner que 50. Cela ferait en tout 545.000 accès physiques (écriture et lecture).

Avec St_2, on lira les 10.000 lignes de PRODUIT, on en sélectionne 50, ce qui fait 10.050 accès (écriture et lecture). Pour chacune de ces 50 lignes, il faudra retrouver la ligne de FOURNISSEUR qui lui est associée. Cela fera au total 10.050+50+**50x**(100+1)/2 = 12625 accès. Cela fait un rapport de 1:50.
St_2 est donc 50 fois meilleure que St_1.

L'optimiseur **doit donc être capable de choisir la meilleure des stratégies** au risque, comme l'exemple le montre, d'afficher des performances inacceptables au point de vue du nombre d'accès. Dans la pratique, il y aura des buffers et les pages auront des capacités supérieures à 1, mais il est clair que la seconde stratégie sera de toute façon préférée.

Le choix de stratégie fait par l'optimiseur dépendra, par exemple, des statistiques et devrait prendre en compte les index disponibles au moment de l'exécution de la requête.
Il choisira par exemple d'utiliser un non clustering index[1] sur *nom* si le prédicat de sélection de la requête est `where nom = 'Dupont'`. Le SGBD-R gère dans son propre catalogue des informations donnant la sélectivité des différents attributs ou en d'autres mots la cardinalité de la relation et le nombre de valeurs différentes de cet attribut présentes dans la table.

Ces choix pourront donc prendre en compte l'évolution de la BD en taille afin de toujours donner le plus court temps d'exécution.

A ce sujet, il faudra veiller à ce que les statistiques évoluent avec la base de données. Il faut par exemple demander explicitement une nouvelle évaluation des statistiques avec ORACLE V7.0 alors que Rdb/VMS V4.1 le fait périodiquement. Ces statistiques et leurs

1 Index imposant un ordre physique aux entités dans le fichier.

évolutions sont évidemment très importantes pour que l'optimiseur ne soit pas induit en erreur et fasse par exemple un accès à l'aide d'un index alors que la table ne contient plus que quelques rares articles.

Dans les SGBD de type CODASYL par contre, les choix de stratégie sont faits une fois pour toute. Le "comment faire" pour exécuter le traitement est en effet compris dans l'algorithme d'accès CODASYL

Dans la réalité, on peut dire que, étant donné une requête, l'optimiseur essaie de trouver une **bonne** stratégie d'exécution de cette requête et n'essaie pas d'atteindre à tout prix la meilleure stratégie. On peut même aller plus loin en disant que s'il cherchait la meilleure stratégie à tout prix, celle-ci ne serait plus la meilleure au moment de son exécution si on prend en compte le temps que lui a pris l'élaboration de celle-ci. L'optimiseur est un organe assez complexe du SGBD-R. Certains peuvent même décider de changer de stratégie en cours d'exécution, se rendant compte que la stratégie en cours prend trop de temps (optimisation dynamique dans Rdb).

Le comportement de celui-ci influencera fortement les performances pour, souvent le meilleur, mais parfois le pire.

Le fait que **différentes formulations de la même requête** (équivalente du point de vue du résultat) **ne donnent pas nécessairement lieu à la même stratégie** (figure-2) dans beaucoup de SGBD-R peut amener des pertes de performances involontaires. Idéalement cela ne devrait pas être le cas.

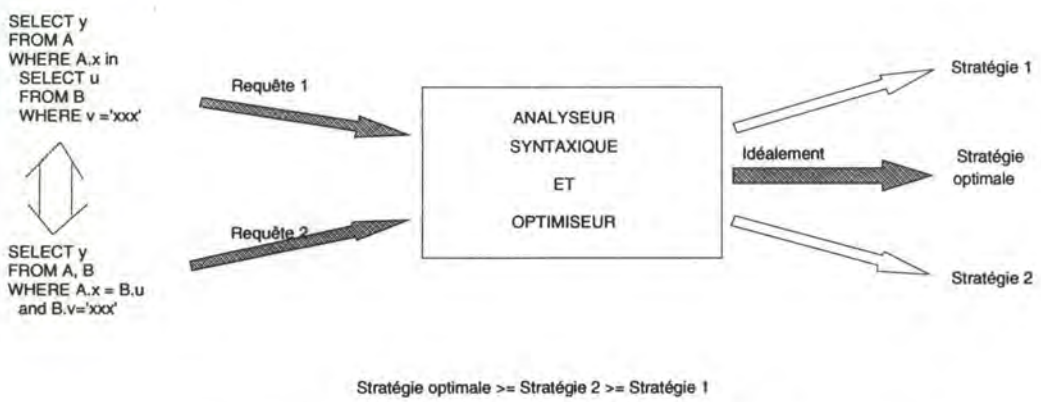


Figure-2 : *L'optimiseur réagit différemment à des syntaxes différentes de la même requête.*

Nous allons maintenant donner **quelques conseils pour écrire les requêtes plus efficacement, en tenant compte des réactions de l'optimiseur**[2]. Nous allons uniquement voir que l'optimiseur refuse d'utiliser les index dans certains cas et discuter de la nécessité du mot réservé "DISTINCT" après avoir illustré comment on peut obtenir des renseignements à l'aide de la commande EXPLAIN de ORACLE. Il est en effet important, après avoir choisi un

2 Ces réactions parfois inattendues auraient dû idéalement rester dans une boîte noire et être en accord avec l'idée que l'on avait d'une stratégie optimale.

design physico-logique, de s'assurer que les hypothèses que l'on fait quant aux stratégies d'exécution sont vérifiées[3].

Alors que la plupart des optimisations proposées jusqu'ici (restructuration, ajout d'index) peuvent améliorer les performances à un certain point de vue mais aussi avoir des effets néfastes à d'autres points de vue, **les conseils que nous allons donner ici pour réécrire les requêtes ne peuvent qu'améliorer la rapidité d'exécution des traitements.** La réécriture des requêtes n'entraîne, en plus, aucune modification globale de l'application opérationnelle, par opposition à une découpe verticale par exemple, qui nécessite la transformation de tous les traitements utilisant la table partitionnée, et la transformation des données. Ces techniques sont donc sans aucun danger et n'engendrent pas de grands coûts de maintenance.

Les techniques de réécriture de requêtes présentées ci-après ont été testées durant le stage réalisé chez **OBLOG Software S.A.**; des résultats appréciables et appréciés ont été obtenus. Le SGBD-R était en l'occurrence **Rdb/VMS V4.1**.

La première chose à faire est de repérer les requêtes qui s'exécutent trop lentement. On peut repérer une requête qui s'exécute trop lentement de deux manières :

- elle effectue énormément d'accès au disque (une requête ne devant récupérer qu'un seul article fait une lecture de toute la table) ou,
- en regardant sa stratégie d'exécution (query plan), on s'aperçoit que des index ajoutés durant la conception physique ne sont pas utilisés.

Des logiciels de contrôle des accès disques et donnant les plans d'exécution sont à la disposition de l'administrateur pour les différents SGBD commercialisés.

Prenons un exemple concret montrant comment on peut se servir du plan d'exécution d'une requête. Soit la requête suivante :

```
SELECT      DISTINCT job
FROM        employe
WHERE       salaire NOT > 40000 ;
```

On suppose en outre que l'on a un index sur *salaire*. On sait en plus que le prédicat "`WHERE salaire NOT > 40000`" a une forte sélectivité vu que, par exemple, l'entreprise a la réputation de bien payer.

La syntaxe ORACLE pour connaître le plan d'exécution de la requête est la suivante :

```
explain plan
set statement_id = 'toto'
into query_plan
for select distinct job from employe where salaire not > 40000;.
```

Le plan d'exécution est alors stocké dans une table spéciale, que l'on peut consulter.

---

3 Dans l'hypothèse de l'utilisation d'un outil tel que RdbExpert (voir 3.2.1.5), il est aussi important de bien écrire les requêtes afin que l'outil (qui se sert du comportement de l'optimiseur) puisse proposer un bon design physique.

Opération	Option	Object name	Id	Previous id
sort	distinct		1	
table scan	full	employe	2	1

Tab-1 : *Un plan d'exécution de la requête : `select distinct job from employe where salaire not > 40000;.`*

On peut voir dans Tab-1 que le SGBD effectue une lecture complète de la table (table scan full) et effectue un tri (sort) dû au mot DISTINCT. L'index n'est donc pas utilisé alors qu'il nous semblait judicieux vu la forte sélectivité du prédicat.

3.4.1. Non utilisation des index

On le voit, il est important de **s'assurer que les index auxquels on avait pensé durant le design physique sont effectivement utilisés.** Beaucoup d'optimiseurs génèreront en effet un plan d'exécution, une stratégie, qui n'incluera pas l'utilisation d'un index dans les cas suivants [SHA92], [LON92] :

- **une expression arithmétique** est présente dans un prédicat de sélection.

```
WHERE salaire/12 >=4000
```
serait avantageusement remplacé par

```
WHERE salaire >= 12*40000.
```

- Une requête utilisant la fonction SUBSTR (elle repère les strings contenus dans une chaîne de caractères).

- Une comparaison avec **NULL.**

Soit par exemple la requête suivante :

```
SELECT      id
FROM        contribuable
WHERE       bareme is NULL;.
```

Pour cette requête, la plupart des optimiseurs n'utiliseront pas d'index. La valeur NULL n'étant pas reprise dans le dictionnaire de celui-ci.

- Une comparaison avec NOT, comme dans `WHERE bareme NOT = 1.`

Il faudra aussi vérifier, que lorsqu'on écrit `"WHERE salaire NOT > 40000"` l'optimiseur soit assez intelligent pour traduire en `"WHERE salaire <= 40000"`. Ce n'était apparemment pas le cas dans notre exemple ci-dessus (tab-1). On réécrira alors la requête en supprimant le NOT afin que l'optimiseur prenne en compte l'index sur salaire.

- Une requête contenant une requête emboîtée. On rencontre en effet souvent, des requêtes du genre :

```
SELECT      nom_cli, ad_cli
FROM        client
WHERE       n_cli IN
                  (SELECT     n_cli
                  FROM        commande
                  WHERE       produit = 'P1' ) ;
```

Un index sur *CLIENT.n_cli* ne sera pas utilisé lors de l'exécution de cette requête dans beaucoup de SGBD commercialisés. Par contre, la requête suivante, équivalente à la première, fera utiliser, à coup sur l'index par l'optimiseur :

```
SELECT      c.n_cli
FROM        client c, commande co
WHERE       c.n_cli = co.n_cli
AND         co.produit = 'P1' ;
```

C'est à l'aide de cette transformation que nous avons pu améliorer fortement (de l'ordre de 30 %) les performances de certaines requêtes de l'outil Oblog CASE V1.00. Rdb/VMS V4.1. n'utilisait en effet pas l'index sur *n_cli* dans le cas de la première requête.

Il faut cependant vérifier que l'optimiseur du SGBD employé réagisse de la sorte avant d'entreprendre une transformation aveugle de toutes les requêtes comportant des requêtes imbriquées en une jointure.

Nous avons en effet remarqué, après avoir "migré" Oblog CASE d'une plate-forme Rdb à une plate-forme ORACLE que l'optimiseur de cette dernière était indifférent aux deux syntaxes.

Voici rapidement présentée cette transformation de requêtes.

1. On combine d'abord les arguments des clauses FROM
2. On met ensuite toutes les clauses WHERE sur un pied d'égalité en les unissant par un AND.
3. Remplacer le IN par = et conserver la clause SELECT de la requête extérieure.

Cette transformation fonctionne pour des requêtes emboîtées d'un niveau quelconque mais il ne doit pas y avoir, dans la requête emboîtée, de référence aux tables présentes dans la requête supérieure (cond-1). Il faudra, cependant, parfois s'inquiéter de la présence de doublons et ajouter DISTINCT dans la clause SELECT de la requête transformée.

Si toutefois intervient dans la requête une des fonctions COUNT, AVG, MIN, ou encore MAX ou si la condition cond-1 est violée, on consultera [GANS87] avant de tenter cette transformation, celle-ci devenant alors plus complexe.

3.4.2. Utilisation adéquate de "DISTINCT"

Après s'être assuré de l'utilisation des index, on vérifiera que l'on n'utilise pas à tort le mot réservé DISTINCT lors de l'écriture des requêtes, celui-ci entraînant un tri ou d'autres coûts supplémentaires.

En fait, on peut remarquer que le mot DISTINCT est inutile si, et seulement si, les deux conditions suivantes tiennent :

- Toutes les tables ayant au moins un champ dans la clause SELECT ont un de leur champ identifiant dans cette même clause.
- Toutes les tables n'ayant aucun champ dans la clause SELECT sont jointes (par égalité) à l'aide d'un de leurs identifiants à une table vérifiant la première condition.

Donnons quelques exemples.

La requête suivante appliquée au schéma décrit en début de section ne nécessite pas le mot DISTINCT :

```
SELECT      produit.#P, produit.#F
FROM        fournisseur, produit
WHERE       fournisseur.#F = produit.#F
```

Dans la requête suivante, par contre, le mot DISTINCT est nécessaire si l'on ne veut avoir de doublons dans le résultat :

```
SELECT      produit.#F
FROM        fournisseur, produit
WHERE       fournisseur.#F = produit.#F
```

La première condition n'est en effet pas vérifiée.

Dans le dernier exemple suivant, c'est la seconde condition qui n'est pas vérifiée. Le résultat comportera donc des doublons si l'on n'ajoute pas DISTINCT.

```
SELECT      produit.#P, produit.#F
FROM        fournisseur, produit
WHERE       fournisseur.#secteur = produit.#secteur
```

Nous venons de voir que l'optimiseur des SGBD-R avait ses limites, que des efforts comme choisir un bon ensemble d'index pouvaient ne servir à rien, l'optimiseur ne les utilisant pas.

Cependant, rappelons-le, l'optimiseur et le fait de ne pas devoir connaître le schéma physique de la base de données constituent une des forces de SGDB-R. Ne pas pleinement profiter du langage SQL et de l'optimiseur serait une erreur qui pourrait coûter cher en temps d'exécution comme le montrent S. Larcen et R. Yevich dans [LAR93].

Comme ils le disent, cette erreur provient souvent de la migration d'une application basée sur une plate-forme hiérarchique vers une plate-forme SQL. Nous ne dirons rien de plus à ce sujet quittant ici l'optimisation de la base de données proprement dite pour celle de l'application accédant à cette base de données.

Nous voulions seulement mettre en évidence que les mauvaises performances d'une application accédant à une base de données ne sont pas toujours dues à un design non optimisé de la base de données.

Chapitre 4

Sujets non abordés et fonctionnalités pour un outil CASE

" Il faut compter ses richesses parmi les moyens qu'on a de satisfaire ses désirs. "

Abbé Antoine François Prévost

Nous avions choisi de traiter les optimisations de base de données réalisables lors de la conception logique. Il existe cependant nombre de paramètres et composants des SGBD mais aussi de l'environnement et des applications qui peuvent influencer les performances. Nous allons en citer pêle-mêle quelques-uns

Pour ce qui est de l'**administration**, on veillera par exemple à ne pas **localiser**, si possible, **des fichiers** sur lesquels on fait beaucoup d'accès séquentiels avec des fichiers sur lesquels on fait principalement des accès directs, et ce afin de limiter les mouvements des têtes de lecture. Les fichiers contenant les catalogues devraient aussi être localisés sur des disques très rapides. Dans le cas des SGBD-R, on essaiera de ne pas stocker les index et les données dans les mêmes fichiers. La littérature (ACM et IEEE) et les guides d'administrations sont riches à ce sujet.

L'administrateur pensera aussi à profiter de toutes les possibilités offertes par les SGBD. On pensera par exemple, à allouer la taille appropriée aux fichiers de données, afin que le SGBD ne doive pas trop souvent ajouter des extensions aux fichiers et ainsi causer la fragmentation des fichiers.

Il faudra aussi mettre au point **le sous-système de Recovery**. Celui-ci peut être la source de contre-performances importantes.

Pour ce qui est de l'**application**, il faudra aussi faire attention à la taille des transactions et éviter à tout prix, les interactions avec l'utilisateur. En effet, des locks sur certaines ressources étant détenus par la transaction, ils ne seront libérés qu'à la fin de la transaction qui ne peut se terminer que lorsque l'utilisateur aura entré les informations demandées. Il faudrait d'ailleurs aussi ajuster la granularités des locks.

Le choix des chemins et clés d'accès ainsi que de leur implémentation (B-tree, ISAM, hash, dense, non-dense, de tri ou non ...) ainsi que le choix des clusters [SHA92] est un sujet à part entière. Nous l'avons dit, des logiciels "experts" font leur apparition dans le domaine : RdbExpert, IBM/RDT [RDBEX], [FIN88].

Au **niveau logique**, il faudrait aussi étudier les mécanismes d'ajout d'identifiant "simples" (souvent relatif aux performances des mécanismes d'accès), et d'ajout de valeurs par défaut. Nous avons en effet relevé quelques problèmes dus à la valeur NULL dans les SGBD-R par exemple. La gestion et les conséquences de l'utilisation de la valeur NULL sur les performances constituent, elles aussi, un vaste sujet encore très peu formalisé [DAT83].

Les transformations de schémas que nous avons présentées : **dénormalisation, découpes/fusions horizontales et verticales** ne sont pas supportées pour l'instant dans l'outil CASE **DB-MAIN**.

Pour la transformation de normalisation/dénormalisation, il faudrait disposer de la notion de dépendance fonctionnelle. Or, cette notion est absente de l'outil DB-MAIN et demanderait sans doute quelques modifications dans le code existant. La dénormalisation et la fusion de deux TE unis par une relation One-to-One pourraient être supportées par le même outil, celui-ci apportant des dépendances fonctionnelles dans le cas de la dénormalisation.

La découpe/fusion verticale serait elle implémentable à court terme.

Si cette transformation est implémentée, il ne faudrait pas permettre de "couper" dans un groupe[1]. Autrement dit, nous voulons que les composants d'un groupe se retrouvent dans le même fragment après que l'on ait réalisé une découpe verticale du TE contenant ce groupe (voir 3.2.2). Les groupes et unions de groupes non disjoints formeraient alors des atomes non fissibles.

[1] On consultera [DBMAIN] pour une description de la notion de groupe. C'est en quelques mots un élément destiné à "grouper" des éléments (attributs, rôles, groupes, ...) dans divers buts comme par exemple pour signifier qu'un ensemble d'attributs est un identifiant d'un TE.

Pour la saisie de la découpe, le concepteur disposerait d'une liste d'atomes parmi lesquels il choisira ceux qui formeront les deux fragments. Il choisira également les deux noms de rôle ainsi que le nom de l'association 1-1 qui unit les deux fragments. Il serait souhaitable à ce niveau de ne présenter que les atomes afin de minimiser les erreurs de manipulations de l'utilisateur.

Une proposition d'interface pour cette découpe serait la suivante (figure-1) :

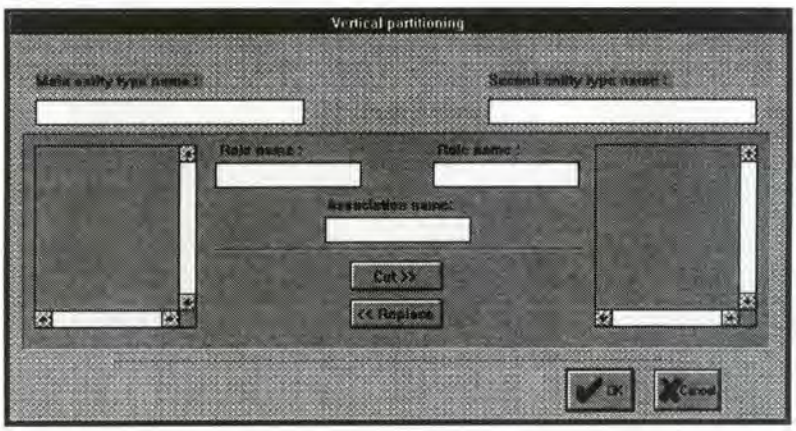


Figure-1 : *Une proposition d'interface pour la découpe verticale d'un TE.*

Comme nous l'avons dit au chapitre 3, il faudrait idéalement pouvoir choisir un design logique après l'avoir optimisé physiquement. Il pourrait donc être intéressant de disposer de simulateurs couplés à DB-MAIN, intégrant un éditeur de traitements, et pouvant donner le résultat (fonction de coût) d'une proposition de design physico-logique Mais ceci ne saurait être réalisé qu'à très long terme ou par couplage à un outil existant.

Il est maintenant temps de tirer les conclusions de ce travail.

Chapitre 5

Conclusion

" Je sens que je progresse à ceci que je recommence à ne rien comprendre à rien ."

Charles Ferdinand Ramuz

Nous avons vu à plusieurs reprises (dénormalisation, découpes horizontale et verticale), qu'il n'est pas toujours facile de déterminer **à priori** si les restructurations logiques de schéma apporteront un gain de performance.

Nous avons, en effet, montré que les éventuels gains de performance apportés par certaines restructurations de schémas dépendaient fortement du design physique (et même parfois de l'écriture des traitements - voir 3.4 -) et ne devenaient parfois effectifs que si on devait utiliser l'accès séquentiel aux données. Or, ce type d'accès est toujours plus coûteux que les accès par clé lorsque l'on ne considère que les coûts en accès logiques. **On ne peut donc évaluer les gains probables des restructurations en ne considérant que les coûts logiques**. Ceux sont pourtant les seuls coûts que l'on peut considérer si l'on veut absolument rester au niveau logique et respecter une méthodologie classique de conception de base de données comme celle présentée à la figure-1 du chapitre 1.

Si l'on suit une méthode classique de conception de BD, après avoir produit le schéma logique des données, on procèdera à son optimisation physique pour essayer d'atteindre le minimum[1] d'une fonction de coût d'utilisation de la BD.

---

[1] Rappelons que l'on ne peut espérer qu'un bon design physique, la recherche de l'optimum étant démontrée être un problème NP-complet.

Or, nous avons aussi vu que certains schémas logiques optimisés physiquement, sont plus efficaces que d'autres schémas logiques eux aussi optimisés physiquement.

En optimisant physisuement le schéma logique produit de la conception logique classique qui ne peut prédire si c'est bien le meilleur schéma logique, on ne peut en fait qu'atteindre un minimum **local** (figure-1).

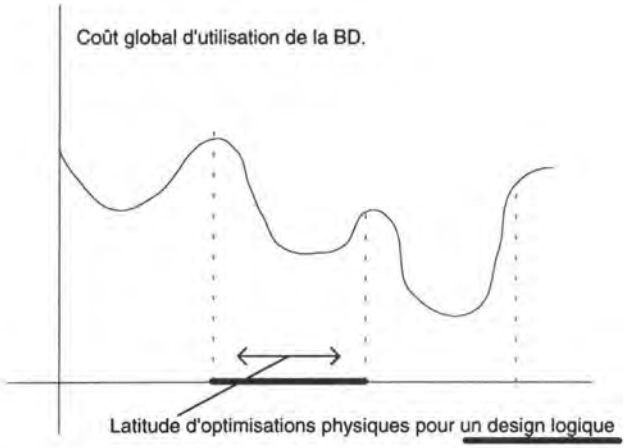


Figure-1 : *Pour un schéma logique, il existe une certaine latitude dans les optimisations physiques. Un "minimum locale" d'une fonction de coût d'utilisation de la BD peut être atteint pour un schéma logique avec les optimisations physiques. Le "minimum" ne peut être atteint que si l'on considère plusieurs schémas logiques.*

Ayant remarqué, le difficile respect des phases d'une méthode de conception de base de données si l'on veut considérer les performances et atteindre un minimum[1] de la fonction de coût d'utilisation de la BD, nous avons proposé (figure-11 chapitre 3) une approche testant les coûts physico-logiques de différents schémas logiques restructurés ou non pour, après, choisir le meilleur design physico-logique. Les méthodes classiques de conception de base de données devraient alors être revues pour intégrer une phase de restructuration logique avec laquelle on essaiera d'atteindre le minimum global (figure-2).

Il est évidemment non réaliste d'envisager une telle approche sans le support d'outil CASE. Nous avons proposé (sans étude de faisabilité) l'utilisation de plusieurs outils commercialisés pour supporter cette nouvelle méthode.

L'intégration de ces outils ne pourra cependant être réalisée à court terme. Il faudra alors se baser sur le bon **sens** pour choisir le meilleur schéma logique qui sera optimisé physiquement comme prévu dans les méthodes classiques de conception.

Nous avons d'ailleurs insisté sur l'importance de conserver la **sémantique** ou plus exactement la facilité d'utilisation du schéma logique pour le programmeur. Nous avons par exemple déconseillé l'utilisation de méthodes de recherche au niveau logique, de bonnes restructurations dont le but est "l'optimisation à tout prix", pour plutôt préconiser des méthodes respectant la sémantique que le programmeur peut attribuer au schéma sur lequel il travaille.

Les transformations d'élimination des attributs multivalués peuvent elle être utilisées en se cantonnant uniquement au niveau logique (voir tableau récapitulatif section 3.3).

Tout au long de ce mémoire, nous avons pu le constater, l'optimisation de bases de données est un problème très complexe et le sujet est loin d'être épuisé.

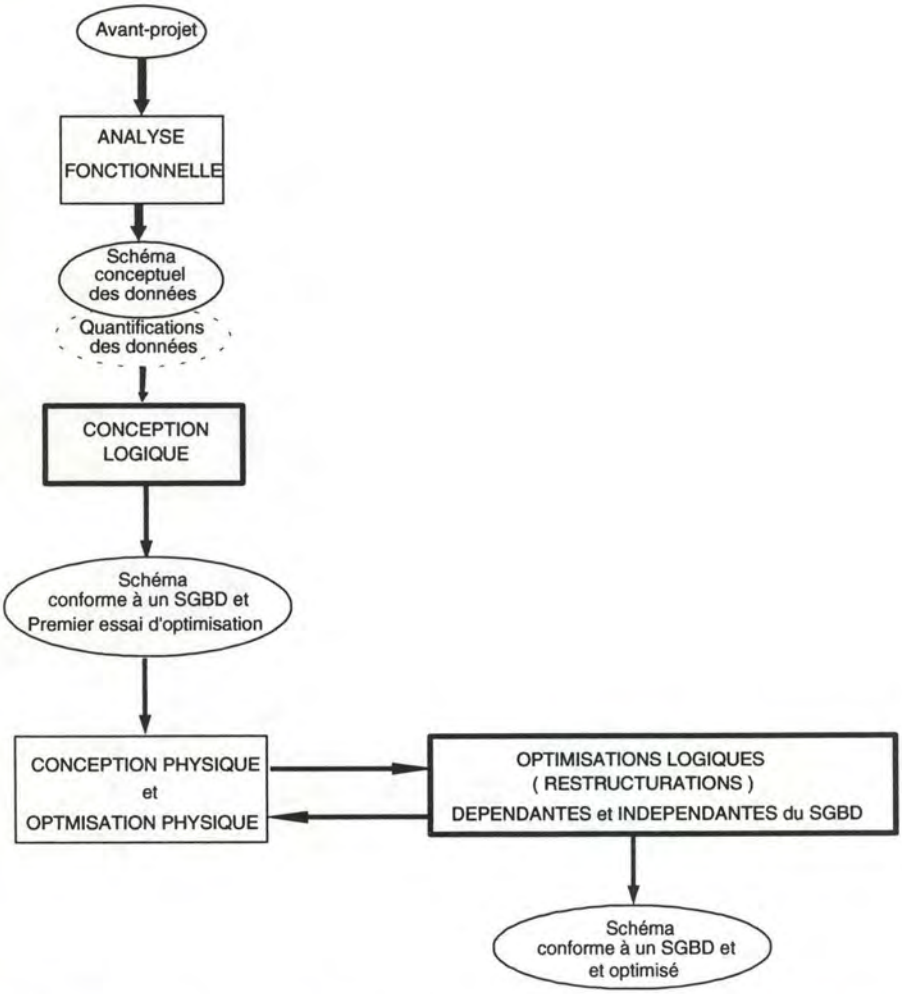


Figure-2 : *Une méthode de conception intégrant une phase de restructuration logiques*

Bibliographie

[BAT92] Conceptual Database Design. C. Batini, S. Ceri, S. Navathe; The Benjamin/Cummings Publishing Company 1992

[BER76] Synthesising third normal form from functional dependencies. P. Berstein. ACM transactions on database systems1, no 4 1976.

[BOD89] Conception Assistée des Systèmes d'Information. François Bodart, Yves Pigneur; Masson 1989.

[BOU92] Distributed Database Systems. D. Bell, J. Grimson Addisuon Wesley 92

[DAT83] An Intoduction to Database Systems. (Volume II) C.J. Date; Addison Wesley 1983

[DAT90] An Intoduction to Database Systems. (Volume I) Fifth Edition C.J. Date; Addison Wesley 1990

[DBMAIN] The DB-MAIN Database Engineering Case Tool. Functions Overview. Project Technical Report, Publication Institut d'informatique FUNDP. July 92

[COM78] The dificulty of optimim index selection. D. Comer. ACM Trans. Database Syst. Vol. 3, N° 4 (Dec 78) 440-445.

[CORN87] A vertical partitionning algorithm for relationnal databases. 3th internatinnal confrence on data eengineering. Los Angeles 87.

[DEL91] Bases de Données: Des Systèmes Relationnels Aux Systèmes à Objets. Claude Delobel, Christophe Lécluse, Philippe Richard; Inter Edition 1991

[FAG77] The Decomposition Versus the Synthetic Approach to Relational Database Design. R. Fagin. In proc. 3th inter. conf. on Very Large Database. Tokyo 1977.

[FICH82] Eléments de programmation linéaire. J. Fichefet. Institut d'informatique FUNDP Namur Sept. 82.

[FIN88] Physical Database Design for Relational Databases. S. Finkelstein. ACM Transactions on Database Systems Vol. 13, No 1, March 88

[GANS87] Optimization of Nested SQL Query Revisited. R.A. Ganski ACM SIGMOD Conference 1987, 23-33.

[GUIG87] Analyse des Données et Choix à Critères Multiples. Chap 2. Partition des Données. J.L Guigou. Dunod 87.

[HAI86] Conception Assistée des Applications Informatiques. Jean-Luc Hainaut; Masson 1986

[HAI89] A Generic Entity-Relationship Models. J-L Hainaut. Proc. of the IFIP WG 8.1 Conf. on Information System Concepts : an in-depth analysis, North-Holland, 1989.

[HAI91] Entity-Generating Schema Transformations for Entity-Relationship Models Jean-Luc Hainaut; 10th Conf. on Entity-Relationship Approach, San Mateo (CA) Oct 1991.

[HAI92] A Temporal Statistical Model for Entity-Relationship Schem J-L Hainaut Institut d'informatique FUNDP (Belgique) 92.

[HAI93] Schema Transformation Techniques for Database Reverse Engineering J-L. Hainaut, C. Tonneau, M. Joris, M. Chandelon; 12th Conf on Entity-Relationship Approach - Arlington (Texas), Dec 1993

[HAM79] A Heuristic Approach to Attribute Partitioning. Hammer and Niamir. In proceedings ACM SIGMOD. International Conference on Managment of Data (Boston 1979), ACM, New-York

[HOFF75] The Use of Cluster Aanalysis in Physical Database Design. J. Hoffer, Proceedings 1st Internationnal Conference on Very Large Database (Framingham 75).

[IBM81] IBM. SQL Data Systems Concepts and Facilities. IBM, 1981.

[LAR93] Mightier than the Sword. S. Larsen, R. Yevich. Database Programming and Design. June 93. 33-39.

[LON92] Guidelines and Good Practice for Developing SQL T. London tlondon@cix.compulink.co.uk Aug. 92.

[NAVA84] Vertival Partitioning Algorithms for Database Design. S. Navathe, S.Ceri. ACM Transaction on Database System Vol 9 N°4 Decembre 84.

[OBLOG] Oblog Software s.a. OBLOG CASE v 1.2 User's Guide. Lisbonne Portugal 1994

[RDBEX] DEC RdbExpert for VMS. User's Guide .Digital Equipment Corporation April 92.

[RUM91] Object Oriented Modeling and Design J. Rumbaugh. Printice-Hall 91

[SCH85] Estimating the Cost of Update in a Relational Database M. Schkolnick et P. Tiberio; ACM Transactions on Database Systems, Vol. 10, No. 2, June 1985, pages 163-179

[SHA92] Database Tuning - A Principled Approach. Dennis E. Shasha; Prentice Hall 1992.

[YAO77] An Attribute Based Model for Database Acces Cost Analysis. S. B. Yao ACM Transaction on Database System Vol 2 N° 1, Mars 77.