CONCEPTION ASSISTEE DES APPLICATIONS INFORMATIQUES

2. - Conception de la base de données

Jean-Luc Hainaut
Université de Namur - Institut d'Informatique
rue Grandgagnage, 21 ● B-5000 Namur (Belgique)
jlhainaut@info.fundp.ac.be - http://www.info.fundp.ac.be/~dbm

Avertissement

Ce document résulte de la mise en forme modernisée (encore qu' inachevée) des textes sources publiés en 1986 par Masson. A l'exception de quelques modification mineures, notamment dans les figures, le présent texte reste fidèle à sa version initiale.

REMERCIEMENTS

C'est pour moi un plaisir, bien plus qu'un devoir, de remercier toutes les personnes qui ont de près ou de loin contribué à la réalisation de cet ouvrage :

- mes collègues de l'équipe du projet de recherche "Bases de Données Administratives" (CIPS I2/5), Claude Deheneffe, Henri Hennebert, Baudouin Le Charlier, et Walter Paulus, ainsi que Sylvio Colucci et Dominique Guillebaud; c'est dans ce cadre en effet que de 1971 à 1977 sont nées les premières notions des modèles MAG et LDA;
- les étudiants des cours de Fichiers et Bases de Données dont j'ai eu la charge depuis 1977, qui ont su de manière constructive mettre en évidence les points forts et surtout les points faibles des éléments méthodologiques de la démarche de conception, notamment lors de la réalisation de mémoires et de laboratoires;
- mes collaborateurs Yves Delvaux, Benoît Van Houtte et Alain Delcourt qui dans le passé et actuellement affinent les concepts et réalisent les outils qui doivent supporter la démarche proposée;
- mes collègues François Bodart pour l'intérêt de longue date qu'il porte à nos travaux dans le domaine méthodologique et auxquels il a jadis donné l'impulsion initiale, et Axel van Lamsweerde pour les discussions qui m'ont permis d'éclaircir les relations que présente la démarche avec la conception des traitements;
- Noëlle Labidi qui a assuré la frappe de ce texte dans des conditions parfois difficiles;
- tous les autres enfin, que j'aurais oubliés.

TABLE DES MATIERES

CHAPITRE 7 : LES TRANSFORMATIONS

. . . . suite manquante

Chapitre 1 : INTRODUCTION

Pris au sens moderne du terme, le concept de base de données se voit associer deux objectifs. Le premier est d'être une représentation fidèle d'un système réel (ce que l'on appelle aujourd'hui le "*réel perçu*"), essentiellement dans ses aspects statiques. Le second est de constituer un serveur de données opérationnel et efficace pour une gamme de traitements.

Le premier volume de cet ouvrage (BOD-PIGN,83) proposait un ensemble coordonné de modèles, d'une démarche, et d'outils destiné à produire la description conceptuelle d'un Système d'Information en général, et de sa base de données en particulier. De celle-ci, on ne retenait que le premier objectif de représentation fidèle. Ce deuxième volume est consacré à la mise-en-oeuvre d'une base de données opérationnelle et efficace répondant à cette description conceptuelle, considérée comme sa spécification. Il s'attaque donc au deuxième objectif assigné à la base de données du Système d'Information. Il présente un ensemble de modèles et une démarche qui prolongent celui de la phase conceptuelle. On y évoquera également les principes d'outils de réalisation d'une telle démarche. Modèles et démarche seront cependant présentés indépendamment des outils, ce qui en rend l'exposé d'une portée tout-à-fait générale.

Si la phase de spécification d'un programme est depuis longtemps reconnue comme essentielle à sa construction et à son évolution, nul n'oserait pour autant prétendre, du moins dans l'état actuel des choses, que son développement à partir de cette spécification ne constitue en toute généralité qu'un art mineur. La construction d'une base de données présente une forte analogie avec celle d'un programme. Sa spécification prend la forme d'un schéma conceptuel, et sa mise-en-oeuvre, tout aussi importante, consiste à produire une définition correcte, efficace et opérationnelle sur une machine réelle, des structures de la base de données à partir de son schéma conceptuel.

Or on constate chez de nombreux auteurs que la phase de mise-en-oeuvre et parfois même la construction entière de la base de données sont réduites à leur plus simple expression, voire même ignorées dans certaines démarches de développement de Systèmes d'Information.

Certaines approches de conception dites "par les traitements", ou du type "software engineering" ne conçoivent encore souvent les données externes que comme de simples sources dont le contenu et la structure doivent être adaptés aux besoins des programmes. La définition des fichiers découle de l'analyse des traitements dont elle n'est qu'un sous-produit secondaire. On ignore ainsi complètement le premier rôle d'une base de données (représentation fidèle d'un système réel), pour ne retenir que le second (serveur de données efficace). Les défauts de cette approche sont bien connus, puisqu'ils servent de base à la comparaison traditionnelle des approches "fichiers" et "bases de données". Ils se traduisent généralement par un risque important d'instabilité de la définition des structures de données vis-à-vis de nouveaux besoins. On trouvera par exemple dans (JACKSON,83) un paragraphe de deux pages consacré à la con-

ception de la base de données, et qui consiste principalement à en démontrer l'inutilité. Sur ce point, l'approche de Jackson relève de celle des types abstraits comme moyen d'expression de schémas externes (voir chapitre 4 du présent volume). Le problème de leur synthèse en vue de constituer un schéma global, stable et général (c'est-à-dire conceptuel) n'est pas abordé. Tout au plus l'auteur renvoit-il aux références classiques (MARTIN,77), et (DATE,81), qui malheureusement n'abordent pas, ou d'une manière très incomplète (ce n'est d'ailleurs pas leur objectif), les problèmes de la conception. On trouvera aussi dans l'excellent ouvrage (ROBINSON,81), une comparaison critique des approches par les traitements ("System design") et du type "Bases de Données".

Les bases de l'approche "Base de Données" ne sont cependant pas neuves. On rappellera notamment que les propositions d'une "algèbre de l'information" ont été développées par CODASYL en 1962 (CODASYL,62), que le modèle relationnel a été largement publié dès 1970 (CODD,70), que la notion de schéma conceptuel l'a été à partir de 1975 (ANSI,75), et que le modèle Entité/Association a été popularisé en 1976 (CHEN,76). Ces travaux constituent des apports fondamentaux à l'approche sémantique de l'analyse des données sans référence aux applications utilisatrices. On citera encore le rapport du WG3 de l'ISO/TC97/SC5 comme état de l'art en la matière (ISO,82).

De nombreux auteurs proposant une démarche de conception de bases de données décrivent d'une manière précise les modèles, les problèmes et les processus de conception relatifs au niveau conceptuel. Les phases de mise-en-oeuvre sont par contre souvent traitées de manière très succinte. Ces phases sont considérées comme pure formalité, le schéma final de la base de données se déduisant immédiatement du schéma conceptuel grâce à quelques règles très simples. Le SGBD est alors considéré comme un outil quelque peu idéalisé susceptible de résoudre tous les problèmes sub-conceptuels. Tout au plus admet-on une phase finale de réglage de quelques paramètres physiques. Cette opinion s'est renforcée par l'arrivée imminente, puis effective, des SGBD relationnels qui devaient libérer la programmation de l'accès navigationnel, et donc de toute préoccupation de performances. S'il est vrai que ces SGBD peuvent rendre moins critiques certaines phases de mise-en-oeuvre, il n'en reste pas moins que certains problèmes restent à résoudre, et même que ces SGBD en induisent de nouveaux (qui seront abordés dans cet ouvrage). Quelles que soient les motivations de ces auteurs, il apparaît que la pratique sur le terrain ne confirme pas cette simplicité du développement, même dans le cas des bases de données relationnelles, sauf si la base est peu volumineuse ou si l'on est peu exigeant du point de vue des performances. Assez curieusement, la littérature est par contre très riche en études, procédés, modèles, abaques et formulaires conduisant à la définition de l'organisation et des paramètres physiques d'un fichier. Citons à titre d'échantillon (SENKO,69), (KING,74), (WIEDERHOLD,77), (CARDENAS,73), (SCHKOLNICK,75), (SCHKOLNICK,79); voir aussi les Transactions on Database Systems des ACM. Cependant, l'utilisation de ces résultats dans une démarche complète et cohérente reste à réaliser par le concepteur du Système d'Information, et n'est pas toujours directement exploitable.

La démarche proposée ici se veut complète en ce qu'elle conduit, à partir d'une description conceptuelle, à une solution exécutable, sous la forme d'un schéma de structure de données selon l'outil opérationnel destiné à gérer ces données, et de programmes rédigés dans un langage de programmation exécutable. Cependant, certains aspects du développement ne seront pas traités (les problèmes liés à la répartition des données et des traitements par exemple), tandis que d'autres ne seront qu'abordés (protection contre les incidents et régulation de la concurrence par exemple). Pour l'essentiel, et ceci, du moins l'espérons-nous, sans verser dans un excès de formalisme, cette démarche est construite sur des bases rigoureuses conduisant autant que possible à une systématisation des phases essentielles de la conception, et par là même à la possibilité de les automatiser ou d'en assurer l'assistance par ordinateur.

A simplement survoler cet ouvrage, le lecteur percevra la forte interaction qui y est mise en évidence entre conception de la base de données et celle des traitements. Il apparaît en effet qu'il n'est plus possible, dans les phases de mise-en-oeuvre (alors que ce l'était au niveau conceptuel), de construire une base de données opérationnelle indépendamment de l'analyse des traitements. En outre, l'usage des bases de données à structure sémantique complexe s'accompagne d'une algorithmique spécifique qui est (très) rarement abordée ou même simplement reconnue. C'est pour ces deux raisons que cet ouvrage aborde avec précision certains problèmes de conception des traitements, dans la mesure où ils sont liés à la base de données. On notera cependant que le problème général de la conception des traitements fait l'objet du troisième volume de cet ouvrage.

Les principes qui sont développés dans cet ouvrage sont nés à la fois de la recherche, d'un enseignement en matière de technologie et de conception de bases de données dispensé tant aux étudiants informaticiens des Facultés Universitaires de Namur, qu'aux professionnels de l'entreprise, ainsi que du développement effectif de Systèmes d'Information opérationnels.

Cet ouvrage n'est bien sûr ni le premier, ni heureusement le dernier à traiter des problèmes de la conception d'une base de données. Il nous semble que l'essentiel de son apport réside, comme nous l'avons évoqué plus haut, dans la rigueur (cependant sous-tendue par l'intuition) d'une démarche qui conduit sans lacune à une solution exécutable, correcte et efficace. Le lecteur intéressé par une perception éventuellement différente des problèmes de conception consultera par exemple les ouvrages suivants : (TAR-NAN-PAS,79), (TAR-ROCH-COL,83), (ROBINSON,81), (CERI,83), (TEOREY-FRY,82), (DELO-ADI,83), (MARTIN,77), (WIEDERHOLD,77), (BEN-ROL,79), (FLORY,82), (MIRANDAS,84), (NBS,85).

Signalons enfin que cet ouvrage est à aborder avec des connaissances suffisantes en matière de bases de données, de gestion de fichiers et de langages de programmation. En particulier, nous ferons usage sans les définir, de notions propres aux SGBD relationnels, CODASYL, ainsi qu'aux fichiers et langage COBOL. Les ouvrages généraux ne manquent pas en cette matière : (DATE,81), (MARTIN,77), (DELO-ADI,83), (GARDARIN,84), (MIRANDAS,84), (PEETERS,84), (CLARINVAL,81) pour n'en citer que quelques-uns.

Le chapitre qui suit présente les principes généraux de la démarche et décrit brièvement l'organisation de l'ouvrage.

Chapitre 2 : PRINCIPES GENERAUX DE LA CONCEPTION DE MISE-EN-OEUVRE

2.1 LE CADRE GENERAL

Rappelons que parmi les principes postulés dans le premier volume (I.1.2) se trouvent celui d'une approche descendante d'analyse et de conception, et celui de l'établissement de modèles servant de fondement à une méthode, elle-même supportée par un jeu d'outils automatisés. Ces principes, par ailleurs communément acceptés à l'heure actuelle, sont bien sûr adoptés également dans cet ouvrage.

Le développement d'un Système d'Information est le processus qui conduit à la production d'un système opérationnel et de ses règles d'utilisation (on ignorera les composants non automatisés). Il est suivi, selon le modèle traditionnel du cycle de vie des systèmes informatiques, d'une phase d'utilisation et de maintenance. Le processus de développement est décomposé en quatre phases : l'étude d'opportunité, l'analyse conceptuelle, la conception de mise-en-oeuvre, la réalisation et la mise-au-point. Les deux premières phases sont traitées dans le premier volume de cet ouvrage, les deux dernières ainsi que certains aspects de l'exploitation et de la maintenance sont développés dans le présent volume.

La phase de conception de mise-en-oeuvre consiste à traduire les spécifications conceptuelles du Système d'Information (la solution conceptuelle) en une description exécutable par un système matériel/logiciel (la solution opérationnelle ou physique). Pour des raisons de convenance, et aussi parce que ces domaines ont des caractéristiques propres, nous distinguerons la mise-en-oeuvre des traitements de celle de la base de données. Cependant, ainsi que nous l'avons précisé dans le chapitre précédent, ces deux domaines seront souvent fortement liés. Ce deuxième volume est consacré à la mise-en-oeuvre de la base de données, ainsi que des composants des traitements qui sont plus spécifiquement concernés par l'accès à la base de données. Il sera suivi d'un troisième volume traitant de la mise-en-oeuvre des traitements.

2.2 OBJECTIFS DE LA DEMARCHE DE MISE-EN-OEUVRE

Nous poserons ci-après les objectifs qui ont été assignés à la démarche de conception de mise-en-oeuvre, ainsi que leur justification.

1. La démarche doit prendre en charge complètement et naturellement la spécification con-

ceptuelle pour conduire à la définition d'une solution opérationnelle, correcte et efficace.

2. La démarche doit être générale vis-à-vis des classes de problèmes. Elle doit également être indépendante des outils opérationnels sur lesquels le système sera implanté, et en particulier du système de gestion de données. Les solutions opérationnelles pourront indifféremment s'exprimer en termes de systèmes de gestion de base de données ou de systèmes de gestion de fichiers traditionnels. De cette indépendance découle le fait que l'on ne trouvera pas dans cet ouvrage de discussion de fond sur les mérites comparés des différentes classes de Systèmes de Gestion de Données (Bases de Données comparés aux Fichiers, Relationnel comparé à CODASYL, etc).
3. La démarche doit prendre en charge tous les critères de décision correspondant aux objectifs du système final (par exemple correction, performances, conformité aux contraintes d'un SGBD, indépendance des programmes vis-à-vis des données, adaptabilité par rapport aux besoins futurs). On adoptera le principe de hiérarchisation des décisions, qui en outre seront groupées selon des classes de critères (ou paramètres lorsque ces décisions sont quantifiables) homogènes. Cette classification conduit à un découplage des décisions, qui peuvent alors être prises en charge par des processus de conception spécifiques et indépendants les uns des autres (ou au moins faiblement couplés). Cette architecture modulaire présente plusieurs avantages. Le premier est une réduction de la complexité par sa fragmentation. Le second est l'adaptativité de la démarche à des conditions d'application variables. D'un cadre général constitué d'un réseau de processus, il est possible de déduire des méthodes particulières, caractérisées par exemple par des outils opérationnels spécifiques (SGBD particulier, langages de 4ème génération), par d'autres principes méthodologiques, par l'absence de certains critères de conception et donc par la suppression des processus correspondants, par la prise en charge de nouveaux critères de conception, et l'inclusion des processus qui les prennent en charge, ou encore par l'adaptation à des outils de conception extérieurs. On observera que ces propriétés d'extensibilité et de reconfigurabilité sont celles des systèmes modulaires en général.
4. Selon la règle désormais traditionnelle, la démarche doit s'appuyer sur des modèles adéquats à son objet. Cependant, elle sera basée plus sur des classes de modèles que sur des modèles tout-à-fait spécifiques. L'adoption d'un autre modèle à tel niveau entraînera essentiellement la définition d'autres règles de transformation vis-à-vis des autres niveaux. Cette relative indépendance résulte de l'architecture modulaire de la démarche.
5. La démarche doit être basée sur des principes intuitifs mais rigoureux. De cette rigueur doivent découler une plus grande garantie de correction des différentes solutions, ainsi qu'une systématisation poussée de la conception. Cette dernière qualité est la condition indispensable à une automatisation des processus de conception, ou tout au moins (et de manière plus réaliste) d'une conception assistée.

2.3 LES MODELES

La démarche propose des modèles adaptés aux différents niveaux sémantiques et techniques de représentation. En ce qui concerne la spécification des structures de données, nous ferons usage d'un modèle spécifique, apte à la fois à représenter la sémantique d'un schéma conceptuel, et à décrire avec précision l'organisation des données du point de vue des accès techniques. Nous serons ammenés également à spécifier des algorithmes d'accès aux données. Nous utiliserons alors un modèle descriptif d'algorithmes.

2.3.1 MODELE DE SPECIFICATION DE STRUCTURES DE DONNEES

Le Modèle d'Accès Généralisé (MAG) propose un jeu de concepts à la fois proches de la pratique des fichiers et des bases de données (sans cependant être spécifiques d'aucun d'entre eux), et basés sur un modèle rigoureux (une variante des modèles relationnels binaires). Cette proximité des outils opérationnels et cette origine en font un outil d'expression adapté à la conversion d'une description conceptuelle en description exécutable.

Les concepts essentiels du MAG sont ceux d'article et de type d'articles, d'item et de valeur d'item (correspondant aux champs ou rubrique), de (types de) chemins d'accès inter-articles (liens d'accès entre articles, typiques des modèles hiérarchiques et en réseau), de fichier, de base de données, de clé d'accès et d'ordre d'une séquence. A ces concepts de base sont associées des contraintes d'intégrité. Ces dernières ne peuvent, en toute généralité être exprimées que si l'on dispose d'un langage de désignation de données. Aux structures de données sont associées des primitives de manipulation.

2.3.2 MODELE DE DESCRIPTION D'ALGORITHMES

Ce modèle est principalement constitué d'un pseudo-langage (appelé LDA) permettant d'exprimer des algorithmes de complexité quelconque, en particulier en ce qui concerne la désignation des données d'une base, et leur manipulation. Les données sont perçues via des structures MAG. Ce langage doit permettre des expressions de désignation et de manipulation primitives (comme dans les langages de programmation traditionnels), mais aussi des expressions de haut niveau (comme dans certains SGBD relationnels).

2.3.3 TRANSFORMATION DE SPECIFICATION

La conception hiérarchisée ("top-down") d'un système complexe étant basée sur la transformation de spécifications, il est essentiel de disposer, tant pour le MAG, que pour ADL, de règles de transformation conservant la correction d'une spécification.

2.4 PRINCIPES GENERAUX DE LA DEMARCHE

La solution opérationnelle résultant du processus global de conception est constituée principalement d'un schéma de base de données (ce terme ici recouvre également la technique des fichiers traditionnels) ainsi que de programmes, tous exécutables par une machine réelle, constituée d'un complexe matériel/logiciel permettant de gérer les données selon ce schéma, et d'exécuter ces programmes. Traditionnellement, une machine réelle sera constituée d'un ordinateur, d'un système d'exploitation, d'un système de gestion de données (ou SGD), d'un compilateur d'un langage de programmation, ainsi que de processeurs tels qu'un gestionnaire d'écran, un moniteur de télétraitement, etc.

Ce processus global est décomposé en deux phases majeures dont la première consiste à produire une solution technique vérifiant les spécifications conceptuelles, qui soit opérationnelle non pas sur la machine réelle finale, mais sur une machine abstraite, elle-même strictement indépendante de la machine finale. Cette machine abstraite est définie comme étant constituée d'un SGBD virtuel obéissant aux principes (modèle de données et primitives) du MAG, et d'un processeur de procédures LDA.

La solution technique sur machine abstraite, ou solution logique, est CORRECTE, en ce qu'elle vérifie les spécifications conceptuelles, EFFICACE, c'est-à-dire optimisée du point de vue des accès, et INDEPENDANTE de la machine réelle. Cette solution est obtenue par la phase de CONCEPTION LOGIQUE. La deuxième phase, appelée CONCEPTION PHYSIQUE, consiste à rendre exécutable la solution logique en la transformant en une SOLUTION PHYSIQUE. Cette solution est CORRECTE, EFFICACE et EXECUTABLE sur la machine réelle.

La phase de conception logique comprend principalement les processus suivants : production d'un schéma MAG à partir du schéma conceptuel (le schéma des Accès Possibles), construction d'une architecture de modules et d'algorithmes (dits prédicatifs) à partir des spécifications conceptuelles des traitements, optimisation de ces algorithmes du point de vue des accès aux données (on obtient ainsi les algorithmes effectifs), puis collecte des besoins des algorithmes effectifs pour produire le schéma des Accès Nécessaires.

La phase de conception physique est constituée des principaux processus suivants : transformation du schéma des Accès Nécessaires de manière à le rendre conforme au SGD, transformation des algorithmes effectifs de manière à les rendre conformes au SGD et au langage de programmation, expression du schéma conforme dans le langage de déclaration du SGD, définition des sous-schémas destinés aux différents utilisateurs, rédaction des programmes, définition du schéma interne de la base de données (fixation des paramètres physiques). En pratique cette conception physique pourra suivre deux voies : soit effectuer explicitement la traduction de la solution logique, soit réaliser la machine abstraite de manière à rendre exécutable la solution logique. C'est dans le cadre de cette deuxième approche que seront étudiés les principes des modules de gestion de données, vus comme technique spécifique de la programmation orientée base de données.

Les différents niveaux de description des solutions, et les schémas de la base de données qui leur correspondent s'inspirent directement des résultats des travaux en matière d'indépendance et d'architecture de SGBD, en particulier des recommandations de l'ANSI/X3/SPARC (ANSI,75).

2.5 DESCRIPTION DE L'OUVRAGE

Les principes établis ci-dessus nous permettent de définir la découpe de cet ouvrage. Le chapitre 3 est consacré au rappel des principes de l'analyse conceptuelle et de ses modèles. Le chapitre 4 étudie quelques approches dans la description multi-niveau d'une base de données, tant dans le cadre des SGBD que de la conception des bases de données proprement dite. Les modèles spécifiques de la démarche de mise-en-oeuvre font l'objet du chapitre 5 (Modèle d'Accès Généralisé) et du chapitre 6 (Langage de Description d'Algorithme). Les transformations de spécifications exprimées dans ces deux modèles sont étudiées dans le chapitre 7. Le chapitre 8 est consacré à la définition de la démarche de conception de mise-en-oeuvre, y compris les plans d'installation et d'exploitation. Cette démarche générale est alors particularisée pour trois SGD généralement considérés comme concurrents : un SGBD CODASYL, un SGBD Relationnel et un simple système de gestion de fichiers (COBOL en l'occurrence). Les problèmes spécifiques à la programmation orientée base de données sont étudiés en clôture de ce chapitre. L'étude d'un cas est développée dans le chapitre 9. On y réalise la conception logique ainsi que la conception physique pour des SGD CODASYL, Relationnel et COBOL. Les programmes et les schémas exécutables correspondant à ce cas sont reportés en annexe. Le chapitre 10 entreprend une brève discussion sur les outils susceptibles de supporter une telle démarche.

Chapitre 3 : RAPPEL DES NOTIONS DE SPECIFICATION CONCEPTUELLE

Ce chapitre reprend très brièvement, du premier volume de l'ouvrage (BOD-PIGN,83), les principes de modélisation, la démarche et les résultats relatifs à la phase d'analyse conceptuelle d'un système d'information. On ne retiendra de ces éléments que ceux qui interviendront directement ou non dans les processus de conception de la base de données.

3.1 LES MODELES

L'analyse conceptuelle repose sur un modèle de structuration des informations de la mémoire du Système d'Information (le modèle ENTITE/ASSOCIATION), et sur des modèles de structuration des traitements.

Le modèle ENTITE/ASSOCIATION met en évidence la notion d'ENTITE, abstraction d'un objet ou concept individualisé du réel (une interprétation proche serait de dire que cet objet réel est une entité). Les ENTITES sont partitionnées en classes appelées TYPES d'ENTITES. Des ENTITES peuvent être en association les unes avec les autres. Les associations sont classées en TYPES D'ASSOCIATIONS; chacun d'eux précise le nombre, le type et le rôle des entités participant à une association de ce type. Chaque membre d'un type d'associations peut être caractérisé par sa CONNECTIVITE (dans combien d'associations de ce type doit-on au minimum et peut-on au maximum trouver une même entité). Les propriétés des objets du réel seront représentées par des attributs que l'on adjoindra aux entités et aux associations. On parlera d'ATTRIBUT d'un Type d'Entités ou d'un Type d'Associations; tandis qu'on parlera de VALEURS D'ATTRIBUT attachées à une entité ou une association. Un attribut est SIMPLE ou REPETITIF, ELEMENTAIRE ou DECOMPOSABLE, OBLIGATOIRE ou FACULTATIF.

Outre ces objets de base, le modèle précise aussi la notion de CONTRAINTE D'INTEGRITE statique (définissant les états valides des données) et dynamique (définissant les changements d'états valides des données). Parmi les principales contraintes d'intégrité non encore citées figurent la notion d'identifiant pour les Types d'Entités et les Types d'Associations, les contraintes d'existence, d'exclusion et d'inclusion pour les Types d'Associations , les dépendances fonctionnelles pour les Types d'Entités et les Types d'Associations, ainsi que les domaines et contraintes de valeurs pour les attributs.

Enfin, la description du réel au moyen des concepts du modèle doit s'accompagner des EXPRESSIONS SEMANTIQUES précisant leur signification exacte.

Les traitements sont spécifiés au moyen de modèles dont nous retiendrons les suivants. Le modèle de STRUCTURATION donne la décomposition arborescente d'un traitement en ses composants (classés en projet, applications, phases et fonctions). Le modèle de la DYNAMIQUE permet de spécifier les règles d'activation des composants des traitements. Enfin le modèle de la STATIQUE des traitements permet essentiellement de spécifier, pour chaque composant de traitement, les informations d'entrée, les informations de sortie, les règles de production de ces dernières à partir des premières, ainsi que les dialogues (forme et contenu) entre chaque composant et l'environnement du Système d'Information.

3.2 LA DEMARCHE D'ANALYSE CONCEPTUELLE

La démarche d'analyse conceptuelle comprend globalement trois phases fondamentales : la construction du sous-schéma conceptuel de chaque phase de traitement, l'élaboration du schéma conceptuel global du projet, et le relevé des quantifications.

Le SOUS-SCHEMA CONCEPTUEL D'UNE PHASE regroupe le sous-schéma des informations et celui des traitements. Le SOUS-SCHEMA CONCEPTUEL DES INFORMATIONS d'une phase se présente d'abord sous la forme dite "BRUTE", qui est obtenue de manière progressive par adjonction de propositions successives qui décrivent le réel. Ces propositions viennent de l'analyse des messages, des règles de traitement, d'applications informatiques préexistantes et de leurs fichiers, d'interviews, etc. Ce sous-schéma brut est alors AFFINE par élimination des homonymes et des synonymes, détermination des identifiants stricts, élimination ou expression des redondances, désagrégation de types d'entités et de types d'associations, décomposition de types d'associations, normalisation de types d'entités et de types d'associations, etc. Le sous-schéma est alors VALIDE quant à sa correction et sa complétude. Le SOUS-SCHEMA CONCEPTUEL DES TRAITEMENTS passe par une phase de construction puis une phase de validation. La CONSTRUCTION consiste à décomposer la phase en fonctions, à exprimer la dynamique de cet ensemble de fonctions et à donner la spécification statique de chaque fonction : informations en entrée et en sortie, règles d'obtention des sorties en fonction des entrées. On définira aussi les scénarios de dialogue entre l'utilisateur et la phase. La VALIDATION consiste en des contrôles de cohérence et de complétude, en tests de réalisabilité (par simulation par exemple), et d'effectivité (par utilisation d'une maquette par exemple).

Le SCHEMA CONCEPTUEL GLOBAL d'une application ou d'un projet est obtenu essentiellement par consolidation incrémentale des sous-schémas des phases constitutives. L'élaboration du schéma conceptuel global des informations conduit à résoudre les cas d'homonymes et de synonymes, à arbitrer les conflits de représentation et de contraintes d'intégrité. L'élaboration du schéma conceptuel des traitements est réalisée par mise à jour des

sous-schémas des phases en fonction de l'homogénéisation des sous-schémas des informations, puis par juxtaposition. Le schéma conceptuel global sera enfin validé dans son ensemble.

Le RELEVE DES QUANTIFICATIONS fournit une description statistique des populations des types d'information : par type, nombre d'entités, nombre d'associations, nombre de valeurs; nombre moyen (ou mieux, distribution de ces nombres) d'associations où apparaît une entité d'un type, longueur moyenne des valeurs d'un type, etc. Quant aux composants des traitements, on en spécifiera la fréquence d'activation. A noter que pour ces derniers, la simulation produit des informations quantitatives précises sur l'activité des différents composants.

Chapitre 4 : NIVEAUX DE DESCRIPTION D'UNE BASE DE DONNEES

4.1 LES ORIGINES DU CONCEPT DE HIERARCHIE DE DESCRIPTION

Des différents travaux de recherche en matière de bases de données qui ont été entrepris depuis le début des années 70, il ressort clairement que l'un des acquis fondammentaux est, à l'heure actuelle, la reconnaissance d'une hiérarchie de niveaux dans la description d'une base de données. La définition de ces niveaux trouve son origine dans deux axes de recherche a priori distincts : l'architecture des SGBD et la conception des bases de données. Le principe est simple, et dépasse d'ailleurs largement le cadre des bases de données : d'un même ensemble de données implanté dans un ordinateur, il est utile de donner des descriptions différentes en fonction des critères de perception de ces données, qui vont d'une perception totalement non technique jusqu'à une perception complète des techniques mises en oeuvre. Précisons que la notion de hiérarchie est plus large que celle d'arborescence, celle-ci étant typique de l'approche traditionnelle dite "top-down".

L'apport d'une perception hiérarchisée, tant dans l'architecture des SGBD que dans la conception d'une base de données, se situe sur deux plans, celui de la réduction de la complexité d'un système, ensuite celui de son indépendance vis-à-vis des perturbations qu'entraînera son évolution. Le principe de réduction de la complexité consiste à offrir un jeu de niveaux caractérisés chacun par un sous-ensemble de critères ou paramètres homogènes. La vue des données qui est offerte à un niveau cache les propriétés qui sont spécifiques des autres niveaux. A cette vue correspond donc une perception simplifiée, et par là même plus aisée à appréhender. Le principe de l'indépendance vis-à-vis des perturbations tient à ce qu'une description n'est pas touchée par la modification d'un paramètre spécifique d'un niveau inférieur.

C'est donc le partitionnement en classes homogènes, de tous les paramètres relatifs à une base de données, qui définit les niveaux de perception de cette base. C'est bien sûr par le mode partitionnement que se distingueront les différentes approches de description. Nous décrirons brièvement quatre approches de description hiérarchique, intéressantes par leur impact pratique (ANSI/X3/SPARC, SGBD), par leur précision (DIAM), ou par leur représentativité par rapport aux pratiques actuelles.

4.2 LE MODELE DE REFERENCE DE L'ANSI/X3/SPARC

Le Standards Planning and Requirements Commitee du comité X3 (Computer and Information Processing) de l'ANSI diffuse de manière restreinte en février 1975 un rapport préliminaire proposant un cadre (ou modèle) de référence pour l'architecture des SGBD (ANSI,75), (ANSI,78). Ce rapport, qui connait immédiatement une diffusion beaucoup plus large que prévue, émane d'un groupe d'étude spécifique (Study Group on Data Base Management Systems) dont l'objectif, défini en 1972, était d'identifier clairement les domaines où l'on pouvaient entreprendre l'établissement de normes, dont le besoin se faisait déjà sentir de manière pressante. Le modèle d'architecture identifie et spécifie les composants fonctionnels d'un SGBD, les différentes classes d'utilisateurs, ainsi que les interfaces entre composants et entre composants et utilisateurs. Le point qui nous intéresse plus particulièrement est que cette architecture est basée sur l'existence de trois niveaux de description de la base de données : la description conceptuelle, la description interne et la description externe (cfr schéma 4.1).

La perception conceptuelle des données consiste en une collection d'objets (articles, lignes de tables, graphes, etc) représentant les entités, les propriétés et les associations auxquelles on s'intéresse dans le système réel. Cette perception fait l'objet d'une description sous la forme d'un schéma conceptuel. Ces objets ont pour seul objectif la représentation stable du système réel, et ne sont en rien liés à une quelconque technologie. Le schéma conceptuel est donc indépendant des paramètres techniques de la base de données ainsi que de l'utilisation que l'on fera des données.

La perception interne des données voit dans celles-ci des objets techniques stockés sur des supports de mémoire, et dont les caractéristiques visent à optimiser les consommations de ressources lors de l'exploitation de la base de données. Ces objets et leurs caractéristiques sont décrits dans le schéma interne de la base de données. Tout changement technologique, ou de paramètre technique entraîne une modification du schéma interne, mais pas du schéma conceptuel. Il existe une correspondance entre le schéma interne et le schéma conceptuel qui spécifie comment sont construits les objets conceptuels à partir des objets internes, et inversement. Cette correspondance est instable comme l'est le schéma interne.

La perception externe des données est celle des utilisateurs de la base de données. Les besoins étant différents de l'un à l'autre, on admet l'existence d'un nombre quelconque de schémas externes, adaptés chacun à un utilisateur (ou classe). Il existe en principe une correspondance entre chaque schéma externe et le schéma conceptuel permettant de construire les objets externes à partir des objets conceptuels et inversément. En pratique, on admet de réaliser la composition des deux correspondances de manière à transformer directement les objets internes en objets externes et inversement.

Ce rapport a eu rapidement un impact considérable. Il était cependant fort ambig~u quant à la définition des niveaux de perception. On n'en veut pour preuve que les interprétations radicalement opposées qui sous-tendent d'une part les travaux de l'ISO TC97/SC5/WG3 sur les modèles conceptuels (ISO,82) et d'autre part les propositions d'admettre comme modèles conceptuels les structures "hiérarchiques" (en clair IMS d'IBM), et les structures en réseau

(en clair CODASYL). On s'est en particulier posé le problème de l'apparente contradiction des termes dans des expressions comme "conceptual record". Ces problèmes d'interprétation semblent actuellement résolus, les notions de modèle et schéma conceptuels ayant depuis quelques années été développés indépendamment de ce modèle de référence d'architecture, pour être utilisé comme outil d'expression dans une démarche de conception. La hiérarchie à trois niveaux du modèle de référence ANSI/SPARC a servi de base à de nombreuses méthodes de conception de bases de données. Elle est cependant souvent considérée comme insuffisante dans le cadre de l'utilisation des SGBD actuels. Ces derniers n'offrent généralement pas la gestion d'un schéma conceptuel et les niveaux externe et interne du modèle ne couvrent pas les possibilités de ces SGBD, qui s'organisent le plus souvent en trois niveaux (voir 4.5).

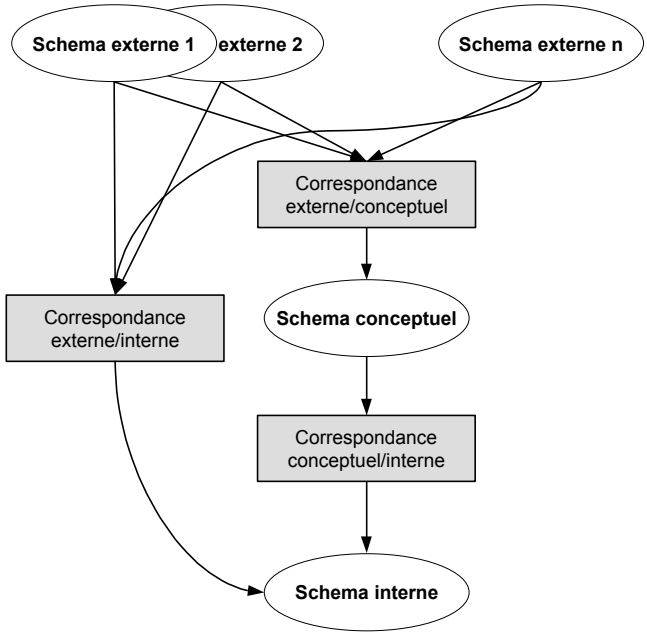


Figure 4.1 - Hiérarchie des modèles ANSI/X3/SPARC

4.3 LE MODELE DE REFERENCE DIAM

La référence fondamentale des travaux de M. Senko (et de son équipe), qui travaillait à l'IBM Watson Research Center de Yorktown Heights, est certainement (SENKO,73), qui expose les principes d'une description basée sur un jeu de quatre modèles (DIAM). Nous considérerons cependant la référence (SENKO,75), qui présente une hiérarchie plus complète (DIAM II).

Tout comme le modèle ANSI/SPARC, DIAM offre surtout un jeu de modèles de données dans le cadre d'un SGBD plus qu'un modèle de base d'une méthode de conception. On y envisage cinq niveaux de perception.

Le niveau END-USER concerne l'utilisateur particulier, qui désire traiter les données selon une forme qui lui est spécifique (cfr niveau externe d'ANSI/SPARC).

Le niveau INFORMATION offre une description indépendante des techniques physiques (cfr niveau conceptuel d'ANSI/SPARC mais sans référence à des objets informatiques). Le modèle retenu est du type relationnel binaire.

Au niveau STRING, les données sont organisées selon des listes de représentations (qui sont des valeurs ou des listes). On propose trois types de listes qui permettent de modéliser l'essentiel des structures de données connues (ce niveau est celui du MAG du chapitre 5 ainsi qu'une partie du niveau interne d'ANSI/SPARC).

Au niveau ENCODING, les listes sont représentées par le stockage de données dans une mémoire abstraite linéaire et homogène.

Au niveau PHYSICAL DEVICE, l'espace de cette mémoire est projeté sur des mémoires réelles, discontinues et inhomogènes.

4.4 LES NIVEAUX DE REFERENCE de l'International Workshop on Data Structure Models for Information Systems, Namur, May 1974)

Dans son rapport introductif (DSMIS,74), le comité de préparation du congrès propose trois niveaux de référence en ce qui concerne la description des données lors de la conception d'un Système d'Information. Les objectifs de cette hiérarchie sont essentiellement la réduction de la complexité et l'indépendance des spécifications par rapport à des critères de conception plus techniques.

Au niveau CONCEPTUEL ou FONCTIONNEL, on donne une description dénuée de toute caractéristique technique, dont le seul objectif est d'être une représentation correcte et naturelle du système réel.

Au niveau de l'IMPLEMENTATION LOGIQUE, on élabore une description qui reprend les spécifications conceptuelles enrichies de la spécification des chemins d'accès aux données nécessaires aux applications.

Au niveau de l'IMPLEMENTATION PHYSIQUE, on spécifie les paramètres physiques des techniques de stockage et des techniques d'accès.

On peut établir une comparaison avec le modèle ANSI/SPARC : absence de niveau externe et décomposition du niveau interne en deux niveaux qui sont conformes à l'architecture de la

plupart des SGBD. On notera également l'approche essentiellement statique de la description des données, typique de l'époque.

4.5 LES NIVEAUX DANS LES SYSTEMES DE GESTION DE BASES DE DONNEES

Les SGBD eux-mêmes n'échappent pas à cette découpe hiérarchique en niveaux de description ou de spécification. Citons simplement le SGBD IMS d'IBM et les SGBD d'inspiration CODASYL.

IMS offre trois niveaux de description. Le premier est constitué d'une perception complète des structures de données (DB physiques + liens logiques), indépendamment des techniques physiques et de l'usage que l'on en fera. Le second précise les paramètres physiques, en particulier les organisations de fichiers. Le troisième permet d'offrir à des classes d'utilisateurs des vues différentes des données (les PCB d'un PSB). Ces niveaux correspondent assez précisément aux niveaux d'implémentation logique, au niveau d'implémentation physique ou interne, et au niveau externe.

Les systèmes CODASYL offrent quant à eux la notion de SCHEMA, correspondant au schéma d'implémentation logique, celle de STORAGE SCHEMA, correspondant au schéma d'implémentation physique ou interne, et celle de SUB-SCHEMA qui correspond aux schémas externes. Il ne semble plus qu'actuellement l'on cherche encore à placer le SCHEMA au niveau conceptuel, car si le SET (lien inter-articles) peut constituer un moyen naturel de représentation des associations entre entités, il n'en est pas moins également un chemin d'accès, même dans les SGBD postérieurs aux spécification de 1978, beaucoup plus claires cependant sur le point de l'indépendance que celles de 1971/73. Cette ambivalence sera discutée dans les chapitres 5, 8 et 9.

Certains SGBD n'offrent que peu de possibilités de percevoir les données selon différents niveaux. Tel est le cas de TOTAL et d'IMAGE, par exemple. Dans ce cas, c'est l'un des rôles des méthodes de conception et de programmation que de restituer ces niveaux (voir section 8.7 en particulier).

4.6 LES NIVEAUX DE REFERENCE DES APPROCHES DE CONCEPTION ACTUELLES

Toutes les démarches générales de conception actuelles sont basées sur les hiérarchies présentées ci-dessus. Les méthodes spécifiquement liées à un SGBD exploitent évidemment les niveaux propres à ce SGBD, mais ajoutent habituellement une phase préalable de niveau conceptuel. Dans ces cas, les modèles le plus communément utilisés sont le modèle Entité/Asso-

ciation (certaines approches d'IBM en amont d'IMS et d'SQL), le modèle de Bachman (dépouillé de ses connotations d'accès) (PERRON,81), le modèle relationnel de CODD associé à des procédés de normalisation par synthèse ou décomposition, le modèle RM/T (CODD,79), et le modèle NIAM (NIJSSEN,**). Les démarches non spécifiques comprennent généralement cinq niveaux au maximum, illustrés dans le schéma 4.2, et que nous décrirons brièvement.

Au niveau de l'analyse des besoins sont produits des SCHEMAS LOCAUX, de nature conceptuelle, mais liés chacun à un sous-système du système réel.

Au niveau conceptuel est défini un SCHEMA CONCEPTUEL GLOBAL synthétisant les perceptions des sous-systèmes.

Au niveau logique est défini un SCHEMA LOGIQUE des données décrivant les structures de données selon les concepts habituels des SGBD, c'est-à-dire correspondant à la perception d'un programmeur d'application. Ce niveau est important, car il est généralement offert par les SGBD, qui constituent souvent les outils privilégiés de gestion de données complexes.

Au niveau physique est défini le SCHEMA INTERNE ou PHYSIQUE, dans lequel sont spécifiés les paramètres physiques des structures de données et de leur usage.

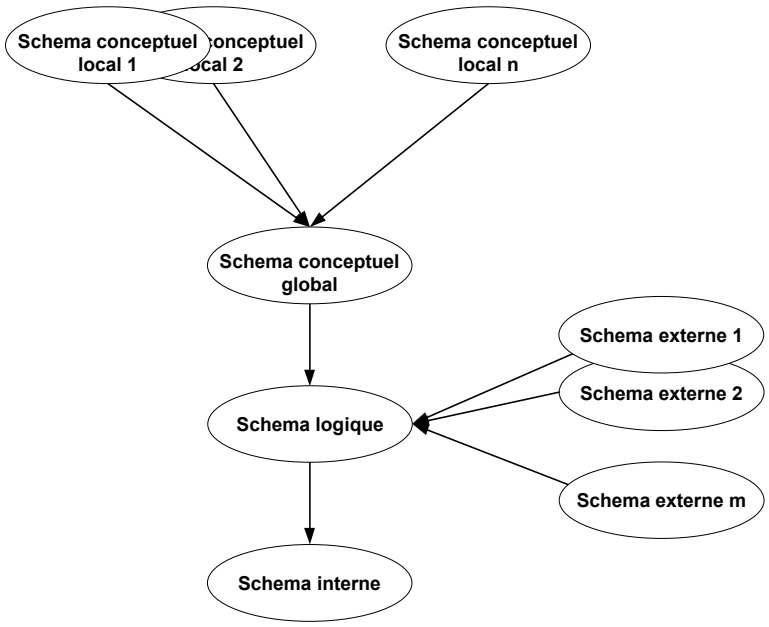


Figure 4.2 - Hiérarchie des niveaux de conception

Au niveau externe sont définis les SCHEMAS EXTERNES propres chacun à une classe d'utilisateurs.

C'est cette découpe en niveaux qui est adoptée dans les deux premiers volumes de cet ouvrage.

Chapitre 5 : MODELE DE SPECIFICATION DE STRUCTURES DE DONNEES - LE MODELE D'ACCES GENERALISE

5.1 INTRODUCTION

Limitée aux structures de données et aux primitives de manipulation de celles-ci, l'analyse montre que sous des apparences très diverses et des nomenclatures divergentes, les systèmes de gestion de données sur mémoires secondaires (que l'on désignera par SGD), qu'il s'agisse de bases de données (ces systèmes seront appelés Systèmes de Gestion de Bases de Données, ou SGBD) ou, plus simplement, de fichiers (on parlera alors de Systèmes de Gestion de Fichiers, ou SGF) mettent en oeuvre les mêmes concepts, en très petit nombre, et que ce qui les distingue fondamentalement est l'ensemble des restrictions qu'ils imposent sur les assemblages possibles de ces concepts.

Le *Modèle d'Accès Généralisé* (MAG) présenté ici est un relevé de ces concepts essentiels, sans référence, sauf à titre d'exemple, aux SGDs où ils apparaissent. On y précisera leur définition, leurs propriétés, ainsi que les assemblages les plus courants. On a cherché autant que possible à conserver aux concepts du modèle les noms sous lesquels ils sont connus dans la communauté informatique, indépendamment de tout SGD et de toute technique de réalisation. Nous proposerons en outre, pour les concepts qui s'y prêtent, une représentation graphique.

Ce modèle décrit les structures de données non seulement sous l'angle de la sémantique qu'expriment ces données, mais aussi sous celui des accès dont elles peuvent faire l'objet. En ce sens, ce modèle ne peut être qualifié de conceptuel. Comme on le verra cependant, il est possible de le décomposer d'une part en un noyau indépendant des accès (*noyau sémantique*) et d'autre part en un ensemble de spécifications des *mécanismes d'accès*. Les concepts du modèle sont typiquement ceux que le programmeur d'application a coutume d'utiliser. Ce modèle nous permettra de spécifier des structures d'accès aux données (des schémas d'accès) qui soient aisément dérivables d'un schéma conceptuel, indépendantes des particularités des SGD, mais proches de leurs concepts intrinsèques, et qui soient systématiquement transformables.

D'autres démarches, comme celle de Robinson (ROBINSON,81), ou la méthode Merise (TAR-NAN-PAS,79) font appel à cette notion de modèle intermédiaire entre le modèle conceptuel et celui du SGD. Le modèle CODASYL (DBTG,71), ou les diagrammes de Bachman (BACHMAN,69) sont parfois utilisés à cet effet. Ces formalismes peuvent cependant s'avé-

rer fort réducteurs par rapport aux fonctions de certains SGD. Ils sont en outre limités aux structures de données, ne prenant généralement pas en charge la spécification des primitives, la désignation de données, l'expression des contraintes d'intégrité, et les règles de transformation systématique.

Défini dans (HAI-LECH,74) comme modèle sémantique, puis dans (DEH-HAI-TAR,75) comme modèle d'accès complet dans une démarche de conception, ce modèle a servi de base à la construction de SGBD (CIPS,74), (CIPS,78), (HAI-LECH,78), parallèlement à son développement comme support méthodologique et pédagogique et comme outil de standardisation (HAINAUT,82).

Nous présenterons les *concepts de base* du modèle, puis ses *contraintes d'intégrité*. L'étude de ces dernières sera précédée de la présentation d'un *langage de désignation de données* permettant d'exprimer ces contraintes. Après la spécification des *primitives* et de l'aspect *quantitatif* des données, nous utiliserons le MAG comme *modèle de référence* pour décrire et spécifier quelques SGD existants.

5.2 LES OBJETS DE BASE DU MODELE

Les objets de base sont les articles et types d'articles, les items et valeurs d'item, les chemins d'accès et leurs types, les fichiers et les bases de données, les clés d'accès et les ordres.

5.2.1 ARTICLE ET TYPE D'ARTICLES

L'*ARTICLE* est une entité stockée d'information qui peut être créée et supprimée, et à laquelle il est possible d'accéder, dans la base de données. L'article est l'unité logique de communication entre la base de données et un programme. Les articles d'une base de données sont distincts, quelles que soient les valeurs d'items qui leurs sont associées et quels que soient les chemins auxquels ils appartiennent.

Tout article appartient à un et un seul *TYPE D'ARTICLES* qui en définit les propriétés communes. A tout instant, 0, 1 ou plusieurs articles peuvent être associés à un type d'articles; ces articles sont appelés la population du type d'articles. Un type d'articles est désigné par un nom qui l'identifie parmi les types d'articles de la base de données. Un type d'articles sera représenté graphiquement par un cartouche contenant le nom de ce type (schéma 5.1).

Figure 5.1 - Types d'articles

5.2.2 VALEUR D'ITEM ET ITEM

Une *VALEUR D'ITEM* est un élément d'un type de données (appelé DOMAINE) qui peut être associé à un article. Il est possible d'accéder aux valeurs d'items à partir de l'article auquel elles sont attachées. A tout instant, 0, 1 ou plusieurs valeurs d'item peuvent être associées à un article déterminé. Une valeur d'item est typiquement un symbole, ou une suite de symboles, qui peut être assigné à une variable d'un programme d'application.

A un type d'articles peuvent être associés 0, 1 ou plusieurs *domaines*. Dans chacune de ces associations, le domaine joue un rôle. Ce rôle est appelé ITEM du type d'articles. Deux items sont dits comparables si leurs domaines le sont (cette notion est celle des langages de programmation traditionnels). Chaque item porte un nom qui l'identifie parmi les items d'un type d'articles. Ainsi, le domaine des ENTIERS peut être associé au type d'article PERSONNE dans les rôles NPERS (numéro de personne) et NTEL (numéro de téléphone). On dira que NPERS et NTEL sont deux items de PERSONNE du type ENTIER (schéma 5.2).

ITEMS ELEMENTAIRES et ITEMS DECOMPOSABLES : le domaine d'un item élémentaire est un ensemble de valeurs atomiques, c'est-à-dire indécomposables en fragments qui soient significatifs dans le contexte du système d'information. Les items NOM et PRENOM du schéma 5.2 sont élémentaires. Une valeur d'un item décomposable est une suite de valeurs significatives, appartenant chacune à un domaine, appelé composant de l'item décomposable. Un composant peut lui-même être décomposable. Par analogie avec la notion d'item, ce composant sera considéré comme un item. ADRESSE et LOCALITE sont, dans le schéma 5.2, des items décomposables.

ITEMS SIMPLES et ITEMS REPETITIFS : un item est simple si à chaque article ne peut être associée qu'une seule valeur de cet item. Tel est le cas de l' item NOM associé au type d'articles PERSONNE (schéma 5.2). Un item est répétitif si à un article peuvent être associées plus d'une valeur de cet item. Dans ce cas, la répétitivité peut être fixe (le nombre de valeurs est le même pour chaque article), limitée (ce nombre est variable selon les articles mais ne peut dépasser une valeur maximum) ou illimitée (ce cas étant essentiellement d'un intérêt théorique). Les items PRENOM et NTEL associés à PERSONNE sont répétitifs. Un item à répétitivité non fixe est généralement accompagné d'un item spécial jouant le rôle de compteur de valeurs pour le premier (NPRE accompagne PRENOM dans le schéma 5.2). Le concept de (non-)répétitivité est aussi d'application pour un item composant par rapport à un item décomposable.

ITEMS OBLIGATOIRES et ITEMS FACULTATIFS : un item d'un type d'articles est obligatoire si à tout article est associée au moins une valeur de cet item. Dans le schéma 5.2, tous les items sont obligatoires sauf NOM-JF (nom de jeune fille).

ITEMS IDENTIFIANTS et ITEMS NON IDENTIFIANTS : un item est identifiant si pour toute valeur de cet item, il n'existe pas plus d'un article associé à cette valeur. Cette dernière propriété sera traitée plus loin, dans un cadre plus général. Dans le schéma 5.2, NPERS (numéro de personne) et NTEL (numéro de téléphone) sont deux identifiants de PERSONNE.

REPRESENTATION GRAPHIQUE : l'item sera représenté par son nom. Son association à un type d'articles sera représentée par un arc entre la représentation du type d'articles et celle de

l'item. Cet arc sera orienté vers la représentation de l'item de manière à préciser qu'il existe par définition un accès, à partir d'un article, vers ses valeurs d'items. On distinguera graphiquement item simple identifiant (pas de symbole), item simple non identifiant (triangle), item répétitif identifiant (triangle) et item répétitif non identifiant (diabolo). On spécifiera également le caractère obligatoire (barre) ou facultatif (absence de barre) d'un item. Ces symboles et ces concepts sont similaires à ceux qui seront utilisés pour les Types de chemins.

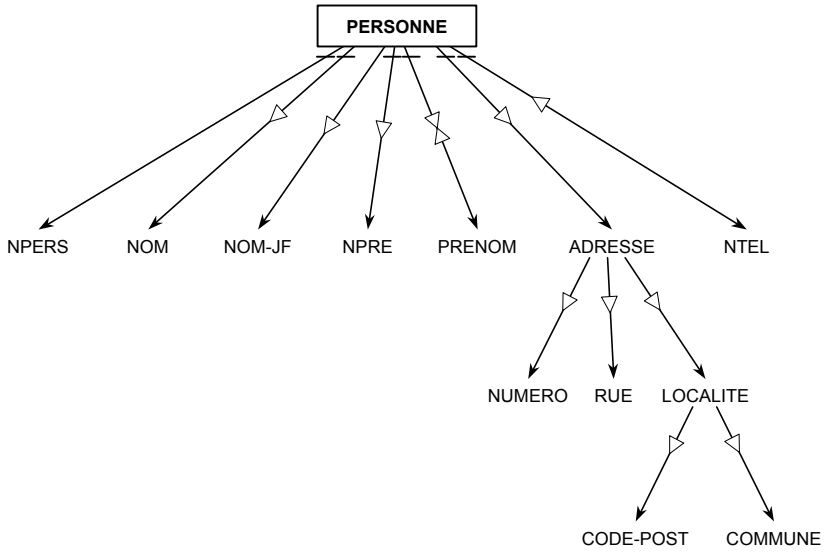


Figure 5.2 - Items d'un type d'articles

5.2.3 CHEMIN D'ACCES (INTER-ARTICLES) ET TYPE DE CHEMINS D'ACCES

Le *CHEMIN D'ACCES INTER-ARTICLES* (appelé désormais *CHEMIN D'ACCES*, ou *CHEMIN*) est un mécanisme 1) qui associe à un article (appelé ORIGINE du chemin) une suite de 0, 1 ou plusieurs articles (appelés CIBLES du chemin), 2) tel qu'il est possible, à partir de l'article origine, d'accéder aux articles cibles successifs du chemin. Un chemin ne contenant pas de cibles ne contient que son article origine; on le qualifiera de vide. On notera que le chemin d'accès définit un accès unidirectionnel. L'accès d'un article cible vers l'article origine d'un chemin n'est possible que via un autre chemin (voir notion de types de chemins inverses).

Tout chemin appartient à un et un seul *TYPE DE CHEMINS (D'ACCES)*, qui en définit les propriétés communes. En particulier, un type de chemins d'accès est caractérisé par les types des articles qui peuvent être origines et par les types des articles qui peuvent être cibles. A tout instant, il y a autant de chemins d'un type qu'il y a d'articles des types origines. A chaque

type de chemins d'accès est associé un nom qui l'identifie parmi tous les types de chemins qui ont même origine et même cible.

L'"absence de nom" est admise comme nom d'un type de chemins; ce nom est représenté par une chaîne de caractères vide. Son nom n'identifiant pas un type de chemins, nous le désignerons, soit par son nom s'il n'y a pas d'ambiguïté, soit par l'expression "NOM du TYPE de CHEMINS (NOM ORIGINE, NOM CIBLE)". Ainsi en est-il du chemin PL(PRODUIT, LIGNE) dans le schéma 5.3.

Bien que les types de chemins à un seul type origine et un seul type cible soient les plus fréquents, on pourra trouver et définir, à l'occasion, des types de chemins à plusieurs types origines (un chemin de ce type possède une origine de l'un des types définis), et/ou à plusieurs types cibles (une cible d'un chemin de ce type peut être de l'un des types définis). Les SGBD qui offrent des types de chemins multi-cibles ne sont pas rares; certains offrent des types multi-origines. Les types de chemins (appelés parfois récursifs) où un même type d'articles apparaît à la fois comme cible et comme origine ne sont pas non plus acceptés par tous les SGBD.

NOTION DE TYPES DE CHEMINS INVERSES : si deux types de chemins, notés TC1 et TC2, sont déclarés inverses, alors pour toute cible C d'un chemin TC1 d'origine O, il existe un chemin TC2 d'origine C dont O est une cible, et inversément. Dans le schéma 5.3, les types PL et LP sont inverses l'un de l'autre. PL permet d'accéder aux LIGNES d'un PRODUIT, et LP permet d'accéder au PRODUIT d'une LIGNE. Bien que chacun des deux types de chemins représente les mêmes associations entre les deux types d'articles concernés (leur "sémantique" est la même), il est utile, à plus d'un titre, de les distinguer. D'abord, cette distinction existe effectivement dans les SGD, qui peuvent offrir des accès inter-articles unidirectionnels ou bidirectionnels. Ensuite et surtout, dans un processus de conception, il est utile de n'exiger que les accès strictement nécessaires; dans bien des cas, un accès unidirectionnel est suffisant et beaucoup moins coûteux qu'un accès bidirectionnel. C'est l'une des faiblesses des modèles d'accès du type CODASYL (le modèle du SGBD n'est évidemment pas mis en cause ici) ou Bachman que de ne pas distinguer explicitement le sens des accès.

CLASSE FONCTIONNELLE D'UN TYPE DE CHEMIN : les types de chemin seront répartis en quatre classes fonctionnelles caractérisant le nombre maximum d'articles d'un membre que l'on peut associer à chaque article de l'autre membre.

On définira ainsi les classes fonctionnelles :

- **1-N**, si un chemin peut contenir un nombre quelconque de cibles, alors qu'une cible ne peut l'être que d'un seul chemin (cfr CL dans le schéma 5.3);
- **N-1**, si un chemin ne peut contenir qu'une seule cible, alors qu'une cible peut l'être d'un nombre quelconque de chemins (cfr CC dans le schéma 5.3);
- **1-1**, si un chemin ne peut contenir qu'une seule cible, alors qu'une cible ne peut l'être que d'un seul chemin (cfr PCLI dans le schéma 5.3);
- **N-N**, si un chemin peut contenir un nombre quelconque de cibles, et si une cible peut l'être d'un nombre quelconque de chemins (cfr STOCK dans le schéma 5.3).

REMARQUES IMPORTANTES. On notera que si un type de chemin est 1-N, son inverse est N-1, et inversement, et que s'il est 1-1 ou N-N, son inverse l'est également. En outre le cas de figure où un type est 1-N et son inverse est 1-N également (souvent décrit comme "*1-N dans un sens et 1-N dans l'autre*") qui est trop souvent considéré dans de nombreux ouvrages (et non des moindres) comme une définition de la classe N-N, correspond en fait à une dégénérescence en classe 1-1. En effet, si un type est de classe 1-N, il correspond à l'inverse d'une fonction, qui elle-même a pour inverse une autre fonction; ce qui définit une bijection.

CONTRAINTE D'EXISTENCE : elle consiste à imposer à tout article d'un membre d'un type de chemin d'être associé à au moins un article de l'autre membre. On dira aussi que le type de chemins est obligatoire pour le type d'articles. Ainsi dans le schéma 5.3, tout article COMMANDE doit être associé via CC à un article CLIENT et via CL à au moins un article LIGNE.

REPRESENTATION GRAPHIQUE : on représentera un type de chemins par un arc orienté, étiqueté du nom de ce type, partant de la représentation de l'origine et aboutissant à celle de la cible. On généralisera le principe aux types de chemins multi-cibles et multi-origines. Lorsque deux types de chemins sont inverses, on pourra les représenter par un arc bidirectionnel. On admet dans ce cas que leur nom est le même. Dans certains cas il y aura lieu de distinguer les deux types de chemins. Leurs représentations seront alors distinctes, et on indiquera, par un symbole approprié le fait qu'ils sont inverses l'un de l'autre (cfr PL et LP dans le schéma 5.3). La classe fonctionnelle sera indiquée par un symbole spécifique : 1-N (triangle), N-1 (triangle), N-N (diabolo), 1-1 (absence de symbole). La contrainte d'existence se notera par une barre à proximité du membre contraint. On remarquera que les deux dernières notations sont celles adoptées pour les items. En effet, si l'on ignore la distinction entre item et type d'article, les propriétés représentées sont identiques.

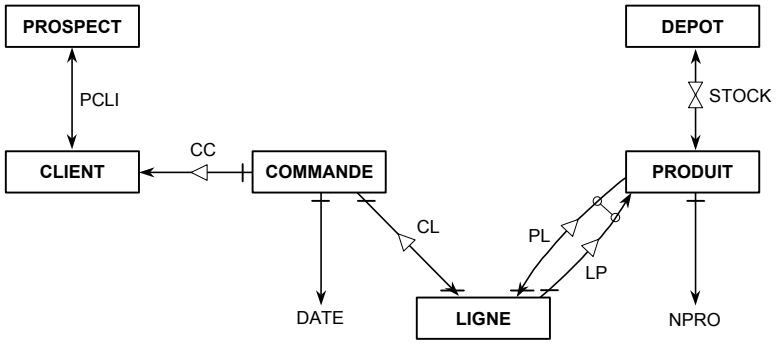


Figure 5.3 - Types de chemins inter-articles

5.2.4 IDENTIFIANT COMPOSE

La notion d'*identifiant*, introduite en 5.2.2 comme propriété d'un item, peut être étendue non seulement à plusieurs items mais aussi à une *combinaison d'items et de types d'articles* associés au type d'articles identifié. Soit un type d'article A, auquel sont associés les types B1,

B2, ..., (items ou types d'articles via des types de chemins). On dira que B1, B2, ..., constituent un identifiant de A si, étant donné un élément de B1, un élément de B2, ..., il n'existe pas plus d'un article A qui soit simultanément associé à ces éléments. On trouvera dans le schéma 5.4 trois figures qui expriment respectivement qu'il n'y a pas plus d'une LIGNE d'un même PRODUIT pour une même COMMANDE, qu'au sein d'un DEPARTEMENT les EMPLOYE ont des NUME qui les identifient, et qu'il n'y a pas deux COMMANDES du même client (NCLI) le même jour (DATE). On y trouvera également une représentation graphique de cette notion (parallélisme partiel des arcs).

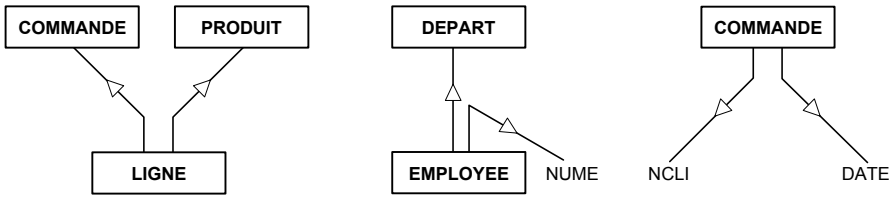


Figure 5.4 - Identifiants composés

5.2.5 LE TYPE D'ARTICLE SYSTEME

On admet que chaque base de données contient un article spécial, appelé l'*article SYSTEME*. Ce concept est introduit essentiellement pour identifier une base de données (il représente l'existence de la base de données), pour désigner des ensembles de données et pour définir des types de chemins (et des ordres) dont l'origine est indifférente. Il a cependant parfois une représentation concrète dans certains SGBD. On conviendra de donner à son type le nom de la base de données.

5.2.6 LE FICHIER

Un fichier est une collection dynamique d'articles. chaque article appartient à un et un seul fichier. En toute généralité, des articles de plusieurs types peuvent se trouver dans un même fichier; d'autre part les articles d'un même type peuvent être répartis dans plusieurs fichiers. On donnera à un fichier un nom qui l'identifie parmi tous les fichiers d'une même base de données. On remarquera que cette notion, si elle est présente dans tous les SGBD, n'est pas toujours accessible au programmeur.

5.2.7 LA BASE DE DONNEES

Une *base de données* est la collection des articles d'un ensemble de fichiers et de toutes les structures de données qui leur sont associées. Une base de données porte un nom qui l'identifie parmi les bases de données connues dans un contexte déterminé.

5.2.8 LES CLES D'ACCES

Une *clé d'accès* est un item, ou un groupe d'items d'un même type d'articles, tel qu'il existe un mécanisme permettant d'accéder successivement aux articles auxquels est associée une valeur déterminée de cette clé, et à eux seulement. Cette notion est parfois locale à un fichier ou à un chemin (cas de CODASYL). La notion de clé d'accès est présente dans la plupart des SGD (l'exception concerne ceux qui n'offrent que l'accès séquentiel). Cependant, certains SGD récents, comme les SGBD relationnels, masquent ce mécanisme au programmeur.

REPRESENTATION GRAPHIQUE : considérant que la notion de clé d'accès représente un accès d'un item (ou groupe d'item) vers un type nous la représenterons par un arc orienté de la représentation de l'item vers celle du type d'articles. A l'occasion, nous adopterons une simplification analogue à celle qui concerne les types de chemins inverses. Dans la première figure ci-dessous, NCLI et NOM sont deux clés d'accès de CLIENT, tandis que dans la seconde, NPRO et SAISON constituent une clé d'accès de PRODUIT. Dans certaines circonstances (transformation p. ex.), on sera amené à distinguer l'accès type d'articles/item de l'accès item/type d'articles, d'où la représentation explicite de ces deux accès, comme dans la troisième figure du schéma 5.5, équivalente à la première.

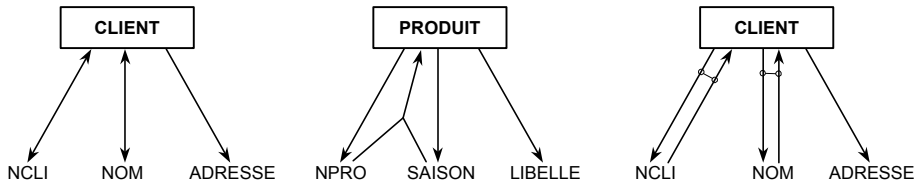


Figure 5.5 - Clés d'accès

5.2.9 ORDRE D'UNE SEQUENCE D'ARTICLES

Plusieurs types d'agrégats d'articles définissent, pour l'utilisateur qui y accède, une *séquence* de ces articles. Il s'agit essentiellement des notions de base de données, de fichier, de chemin d'accès et d'articles associés à une valeur déterminée d'une clé. Pour chacune de ces quatre constructions, l'accès séquentiel à ses articles s'effectue, en l'absence de spécification contraire, dans un ou plusieurs ordres prédéterminés. Sur une séquence qui ne contient que des articles d'un seul type on retiendra les ORDRES suivants :

- *pas d'ordre significatif*;
- ordre lié à l'instant de l'insertion dans la séquence : *chronologique* (First in, first out), ou *anti-chronologique* (Last in, first out). Tout article est inséré soit à la fin, soit au début de la séquence.
- *ordre programmé*, c'est à dire défini par les programmes d'application; cet ordre n'est plus sous le contrôle du SGBD. Dans ce cas le programme d'application choisit pour chaque article l'endroit dans la séquence où il sera inséré.

- *ordre trié* : les articles se présentent par ordre croissant ou décroissant des valeurs d'un item ou groupe d'items, appelé CLE DE TRI.

5.3 LANGAGE DE DESIGNATION DE DONNEES

Nous développerons ici les éléments d'un langage permettant de *désigner des sous-ensembles de données* (valeurs d'items et articles). Ce langage nous permettra d'exprimer des contraintes d'intégrité (section 5.4) et servira de base au langage d'expression d'algorithmes LDA (chapitre 6). Ce langage de désignation s'appuye sur une description des données correspondant au noyau sémantique du MAG. Dans ce noyau, on ne retient d'un item et d'un type de chemins que l'association entre deux ensembles de données, sans référence au sens de l'accès qui y est spécifié. D'autre part, ce langage profitera de l'extrême similitude qui existe entre types d'articles et items (similitude dèjà évoquée à l'occasion de l'expression graphique d'un schéma). De nombreuses constructions, en effet, seront d'application pour les items et pour les types d'articles, ce qui simplifiera le langage ainsi que les règles de transformation qui lui seront associées. Nous serons ainsi amenés à parler d'*ELEMENTS* pour désigner soit des articles soit des valeurs d'items, de *TYPES D'ELEMENTS* pour désigner soit des types d'articles soit des items, et de *TYPES D'ASSOCIATIONS* entre TYPES D'ELEMENTS pour désigner les diverses associations entre items et types d'articles. En particulier, on admettra qu'il existe un type d'associations sans nom entre un type d'articles et chacun de ses items, et entre un item et chacun de ses composants. On trouvera dans (HAI-LECH,74) les premières spécifications du noyau sémantique et de son langage de désignation. Ce noyau est une variante de la classe des *modèles relationnels binaires* (ABRIAL,74), (SENKO,75), (BRACCHI,76) qui, s'ils ne sont pas complets (voir par exemple la notion d'identifiant composé, qui est de nature n-aire; voir aussi (DATE,83)) sont universellement reconnus comme un moyen d'expression sémantique simple et élégant.

5.3.1 LES VARIABLES

Un ou plusieurs articles peuvent être désignés par une variable. Chaque valeur d'une telle variable, dite variable d'article, est la *référence d'un article*. Plus traditionnellement, il existe des variables pouvant contenir des valeurs d'item. Ce concept sera étendu lors de l'étude du langage LDA (Chapitre 6).

5.3.2 LA SEQUENCE

Une séquence est une *collection ordonnée* d'éléments et/ou de valeurs. En toute généralité, ces éléments ne sont pas nécessairement distincts, sinon par leur rang. Cependant, on ne sera souvent intéressés que par les éléments distincts d'une séquence, ne considérant alors que *l'ensemble de ses éléments*.

5.3.3 LES FONCTIONS INTRINSEQUES

Si S est l'expression de désignation d'une séquence, et si s désigne un élément de S, les fonctions suivantes peuvent être définies :

- size(S): taille de la séquence S;
- order(s, S): rang de l'élément s dans la séquence non vide S (voir aussi 5.3.9.2);
- min(S), max(S): valeur minimum, maximum de la séquence non vide S,
- etc.

5.3.4 ENSEMBLES ET SEQUENCES NON QUALIFIES

Le nom d'un ensemble prédéfini désigne cet ensemble lui-même. Ce nom peut être extrait du schéma, ou être celui d'une variable contenant une ou plusieurs valeurs. Un ensemble peut être désigné par le nom d'un type d'éléments de la BD, le nom d'une variable, d'une constante. Il peut aussi être défini par citation de chacun de ses éléments ou comme intervalle. L'ordre de la séquence est celui selon lequel se présentent naturellement les éléments dans cet ensemble. Quelques exemples :

- NCLI désigne l'ensemble des valeurs du domaine de l'item NCLI. COMMANDE désigne l'ensemble des articles COMMANDE.
- CLI désigne l'article CLIENT que référence la variable d'article CLI.
- PRIX désigne la valeur de la variable entière PRIX.
- 'DUPONT' est une constante qui désigne une valeur du type chaîne de caractères.
- (12, 32, 7) désigne un ensemble de 3 entiers.
- (CLI1, CLI2) désigne l'ensemble des articles CLIENT que référencent les variables d'article CLI1 et CLI2.
- 1..10 désigne l'intervalle de 1 à 10 (il s'agira d'entiers lorsqu'il s'agira de manipuler explicitement ses valeurs).
- () désigne l'ensemble vide.

5.3.5 ENSEMBLES ET SEQUENCES

Leur désignation est constituée de celle d'un ensemble non qualifié (soit ENS), suivie éventuellement de l'expression d'une condition de sélection (soit CSEL) et de celle de l'ordre des éléments (soit ORDER). La forme générale est la suivante (/VAR y est facultatif) :

ENS/VAR CSEL ORDER

Elle désigne les éléments de l'ensemble ENS qui vérifient la condition CSEL, considérés selon l'ordre ORDER. VAR est le nom d'une variable du type de ENS; elle désignera chaque élément de ENS lorsque l'on vérifiera qu'il satisfait CSEL. L'expression suivante (son interprétation, et d'ailleurs une forme plus simple, seront données plus loin) désigne les arti-

cles CLIENT dont la valeur de NOM est égale à 'DUPONT' considérés par ordre croissant de NCLI :

```
CLIENT/CLI ( NOM(: CLI) = 'DUPONT' ) sorted(NCLI)
```

5.3.6 LES CONDITIONS

La forme générale d'une condition de sélection (CSEL ci-dessus) est une expression booléenne (opérateurs logiques "and", "or", "not", "→" (implication), parenthèses) de conditions simples. La forme primitive d'une condition simple est la suivante :

(E1 rel E2)

où

- E1 et E2 sont des expressions désignant des ensembles d'éléments compatibles (articles du même type ou valeurs comparables).
- rel est une relation parmi les suivantes : in, not-in, =, =S=, not=, >, not>, <, not<, not=S= ainsi que les formes traditionnelles <> (au lieu de not=), <= (au lieu de not>), >= (au lieu de not<).

Interprétation : la condition est vraie si les ensembles E1 et E2 vérifient la relation rel. Plus précisément :

- E1 in E2 les éléments de E1 (non vide) appartiennent à E2;
- E1 not-in E2 aucun élément de E1 n'appartient à E2;
- E1 = E2 les ensembles E1 et E1 sont identiques;
- E1 not= E2 les ensembles E1 et E2 ne sont pas identiques;
- E1 < E2 E1 et E2 contenant chacun une valeur, l'élément de E1est inférieur à celui de E2;
- E1 =S= E2 E1 et E2 contenant chacun une valeur, l'élément de E1est semblable à celui de E2 (notion d'équivalence phonétique de type SOUNDEX par exemple).
- les autres relations s'interprètent d'une manière analogue.

De manière à simplifier les expressions de condition les plus courantes, nous userons de deux formes particulières, la *condition d'association*, et la *condition d'appartenance*.

5.3.7 LES CONDITIONS D'ASSOCIATION

Nous examinerons principalement la condition qui porte sur le nombre et les propriétés des éléments associés à chaque élément évalué. La forme générale est la suivante :

(A : CARD ENS)

où

- ENS est l'expression de désignation d'un ensemble d'éléments du type NOM2;
- CARD est la désignation d'un entier ou d'un intervalle d'entiers;
- A est le nom, éventuellement absent, d'un type d'associations entre les types NOM1 et NOM2;
- NOM1 est le nom du type des éléments évalués;

Interprétation : (un élément du type NOM1 est retenu) s'il est associé par A à un nombre K d'éléments de l'ensemble ENS tel que K appartienne à CARD.

Si l'on développe ENS et si l'on complète la forme générale, en :

E1(A: CARD E2 COND)

où

- COND est une condition éventuellement absente,
- E1 et E2 sont des noms de types d'éléments ou des variables,

l'interprétation peut aussi s'exprimer selon la forme primitive suivante :

E1/X (size(E2((A: X) and COND)) in CARD)

où

- E2(A:X) désigne les éléments de E2 associés à X par A.

REMARQUE. Un intervalle s'exprime sous la forme i..j et désigne les entiers de i à j, où i et j sont les désignations de deux entiers. Le symbole * représente "*le nombre d'éléments du type NOM2 associés par A à l'élément du type NOM1 en cours d'évaluation*", c-à-d ici : size(E2(A:X)). Il peut désigner un entier dans l'expression d'un intervalle.

EXEMPLES D'INTERVALLES :

- 1..4 signifie "de 1 à 4",
- 0..1 signifie "de 0 à 1", c'est-à-dire "au plus 1",
- 1 mis pour 1..1, signifie "1 exactement",
- 1..* signifie "de 1 à tous", c'est-à-dire "au moins 1",
- * mis pour *..* signifie "tous",
- 4..* signifie "de 4 à tous", c'est-à-dire "au moins 4",

Cas particulier : l'absence de CARD dans l'expression de la condition représente l'expression la plus fréquente : 1..*, c'est-à-dire "au moins 1".

EXEMPLES DE CONDITIONS :

1. CLIENT(CC: COMMANDE) ou CLIENT(CC: 1-* COMMANDE) : les CLIENTS associés par CC à au moins une COMMANDE.
2. NOM(: CLIENT) : les valeurs de NOM qui sont associées à au moins un article CLIENT.
3. PRODUIT(PL: 0 LIGNE) : les PRODUITS qui ne sont associés par PL à aucune LIGNE.
4. PRODUIT(: NPRO = 123) : les PRODUITS associés à une valeur de NPRO égale à 123.
5. COMMANDE(CL: LIGNE(: Q > 500)) : les COMMANDES associées par CL à au moins 1 LIGNE spécifiant une Quantité supérieure à 500.
6. PRODUIT(PL: * LIGNE(: Q < 100)) : les PRODUITS tels que toutes les LIGNES qui lui sont associées par PL ont une Quantité inférieure à 100.
7. NOM(: CLIENT(: ADRESSE(: LOCALITE = "LIEGE"))) : les NOMS des CLIENTS dont l'ADRESSE a une LOCALITE égale à LIEGE.

5.3.8 LES CONDITIONS D'APPARTENANCE

La condition porte sur l'appartenance, ou la non appartenance, de l'élément évalué à une collection d'éléments d'un type compatible. Cette condition sera surtout utilisée pour qualifier des valeurs d'item. Les formes générales sont les suivantes :

REL ENS et **(REL ENS)**

où

- REL est une relation du type défini en 5.3.5,
- ENS est la désignation d'un ensemble d'éléments compatibles avec les éléments évalués.

INTERPRETATION : la condition est vraie si l'ensemble constitué de l'élément évalué et l'ensemble ENS vérifient la relation REL.

En complétant la forme générale, on peut exprimer :

E1 REL ENS ou **E1(REL ENS)**

sous la forme primitive

E1/X (X REL ENS)

EXEMPLES :

1. NOM = "DUPONT" : la valeur de l'item NOM qui est égale à "DUPONT"
2. PRIX in 3..10 : les valeurs de l'item PRIX comprises dans l'intervalle (3..10), c'est-à-

dire de 3 à 10 inclus;

3. PRIX not-in 0..10 : cette expression peut aussi s'écrire PRIX > 10;
4. CLIENT(:NOM = NOM(:CLI)) : si CLI est une variable désignant un CLIENT, cette expression désigne les CLIENTS qui ont un NOM égal à celui du CLIENT CLI.
5. CLIENT/C(:NOM=COMMUNE(:LOCALITE(:ADRESSE(:C)))) : l'ensemble des articles CLIENT dont le NOM est égal au nom de leur COMMUNE.

REMARQUE : on admettra ici, et ici seulement, d'écrire "=" au lieu de "in" et "<>" au lieu de "not-in". L'expression "PRIX not-in 0..10" peut donc s'écrire également :

PRIX <> 0..10

5.3.9 EXTENSIONS

Les extensions, dont nous donnerons une brève description, concernent essentiellement les composants FICHIERS et ORDRE qui sont susceptibles d'être porteurs de sémantique. Nous y ajouterons la description de l'opérateur de composition.

FICHIER

On convient qu'à chaque article est associée une valeur d'un item virtuel nommé FILE, constituée du nom du fichier auquel cet article appartient. Exemples :

- FILE(: CLI) : nom du fichier de l'article CLI,
- CLIENT(: FILE='ARCHIVE81') : articles CLIENT du fichier ARCHIVE81.

ORDRE

Le rang d'un élément s dans la séquence S s'exprime par :

ord(s, S)

On admet que ce rang vaut 0 si s n'appartient pas à S.

Par exemple, le rang de l'article COM dans l'ensemble des COMMANDES associées à son CLIENT, dans l'ordre des DATES croissantes s'exprime par :

ord(COM, COMMANDE(CC:CLIENT) sorted(DATE))

Dans l'expression de la séquence, la désignation de l'article CLIENT n'a pas été précisée car il n'y en a qu'un seul, étant donné la classe fonctionnelle de CC. On peut également désigner l'ordre naturel (quand il n'y en a qu'un) d'une séquence ordonnée. On pourrait ainsi écrire :

ord(COM, COMMANDE(CC: CLIENT))

Cette fonction peut être utilisée dans une condition :

ord(L, LIGNE(CL:COMMANDE)) = 2..*

et servir à désigner des articles. Quelques exemples :

- LIGNE/L ((:Q > 100) and ord(L, LIGNE(CL:COMMANDE)) = 2..*);
  cette expression désigne les LIGNES dont la Quantité est supérieure à 100, et qui ne sont pas la première de leur COMMANDE.

- LIGNE/L (ord(L, LIGNE((CL:COMMANDE) and (: Q > 100))) = 1);
  on désigne ici, pour chaque COMMANDE, la première ligne dont la Quantité est supérieure à 100.

- COMMANDE/C (LIGNE/L ((CL:C) and (ord(L,LIGNE(CL:C)) in 2..3 -> Q(:L) > 100))
  = LIGNE/L ((:C) and (ord(L,LIGNE(CL:C)) in 2..3)))
  cette expression désigne les COMMANDES telles que toute LIGNE dont le rang est compris entre 2 et 3 a une Quantité supérieure à 100. On donnera à cette condition une forme spécifique (il s'agit d'une forme de la condition d'association) :

  COMMANDE (CL: #2..3 LIGNE (:Q > 100))

La désignation d'une sous-séquence contiguë d'une séquence s'exprimera comme dans l'exemple suivant :

- PRENOM/P ((:PERS) and order(P, PRENOM(:PERS)) in 1..3)
  qui désigne les 3 premiers prénoms de la personne PERS. On admettra la forme suivante, plus simple, et plus traditionnelle :

  PRENOM (:PERS) [1..3]

LA COMPOSITION

La composition de deux types d'associations ayant un membre (type d'éléments) en commun correspond à un type d'associations virtuel (il correspond à des données dérivées et non réellement présentes dans la base de données). Considérons R(A,B) et S(B,C) deux types d'associations ayant un membre commun. La composition notée R.S définit un type d'associations virtuel de A vers C noté T(A,C) et défini comme suit :

Pour tout élément X de C, A(T:X) = A(R:B(S:X))

Pour tout élément Y de A, C(T:Y) = C(S:B(R:Y))

Considérons à titre d'exemple le schéma 5.3. Etant donné CL(COMMANDE,LIGNE) et LP(LIGNE,PRODUIT), on peut considérer un type d'associations virtuel de COMMANDE vers PRODUIT, qui serait défini comme la composition de CL et LP. Il serait donc défini par :

CL.LP

L'expression d'une composition peut être utilisée dans le langage de désignation partout où le nom d'un type d'associations peut apparaître. Elle sera cependant surtout utile pour exprimer des contraintes de redondance ou l'équivalence de structures.

5.3.10 SIMPLIFICATION D'EXPRESSIONS

Les processus de transformation systématique peuvent conduire à des expressions qui, tout en étant correctes, peuvent être simplifiées. On retiendra surtout les formes suivantes.

Soient R(A,B) un type d'associations entre A et B,
CA une condition quelconque portant sur A,
CB une condition quelconque portant sur B;

1. Si R(A,B) est 1-N (ou 1-1) et si a in A(R: B)
   alors A(R: B(R: a)) = a
2. Si R(A,B) est 1-N (ou 1-1) et si A(CA) in A(R: B)
   alors A(R: B(R: A(CA))) = A(CA)
3. A(R: B in B(CB)) = A(R: B(CB))

5.4 LES CONTRAINTES D'INTEGRITE

La structuration des données selon les concepts de type d'articles, item, type de chemins, fichier et base de données conduit à des représentations trop générales des structures conceptuelles. Ces dernières sont en effet astreintes à respecter des règles appelées *contraintes d'intégrité*. Il est donc nécessaire d'adjoindre à une description en termes des objets de base du modèle d'accès, un ensemble de règles qui traduisent les contraintes d'intégrité de la description conceptuelle. Certaines contraintes d'intégrité ont déjà été spécifiées en 5.2. Il s'agit essentiellement des notions de classe fonctionnelle, de contrainte d'existence et d'identifiant simple et composé. Cependant il sera souvent nécessaire d'en exprimer une beaucoup plus grande variété. A titre d'exemple, des raisons de performance, de sécurité ou de modularité peuvent conduire à la duplication de structures de données, qui constituent ainsi des *redondances* (un même fait est représenté plus d'une fois). L'*équivalence* de ces structures de données est également une contrainte d'intégrité. Les contraintes autres que celles qui ont été définies en 5.2 s'exprimeront à l'aide du langage définit en 5.3. (Les contraintes vues en 5.2

peuvent être définies d'une manière rigoureuse dans ce langage, ainsi que pourra s'en convaincre le lecteur intéressé.)

Nous examinerons quelques contraintes d'intégrité fréquentes.

5.4.1 CONTRAINTE DE CARDINALITE

On précise ici des contraintes sur la taille de certains ensembles de données. Si par exemple il n'y a pas plus de 20 LIGNES par COMMANDE, on écrira :

pour tout C de COMMANDE, size(LIGNE(CL: C)) <= 20

5.4.2 CONTRAINTE D'INCLUSION

On précise qu'un ensemble d'éléments doit être un sous-ensemble d'un autre.

Exemple : NCLI(: COMMANDE) in NCLI(: CLIENT)

cette expression indique que la valeur de NCLI de tout article COMMANDE doit être également la valeur de NCLI d'un article CLIENT. (Cette contrainte est importante car elle accompagne la traduction relationnelle d'un type d'associations binaire).

5.4.3 CONTRAINTE D'EXPRESSION DE REDONDANCE

Une base de données peut contenir des *redondances*, c'est-à-dire une duplication d'information dont l'objectif est par exemple de diminuer les temps d'accès, de partitionner la base en modules connectables en ligne ou réorganisables de manière indépendante, ou encore d'obtenir une meilleure sécurité vis-à-vis des incidents ou un meilleur parallélisme. Ces critères étant essentiellement de nature technique, il est naturel que de telles redondances soient définies à ce niveau de description des données et non au niveau conceptuel. 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.

Exemple : Soit, dans le schéma 5.6, un item NOM associé au type d'articles COMMANDE et qui reprend la valeur de NOM associée à l'article CLIENT connecté via CC à chaque article COMMANDE.

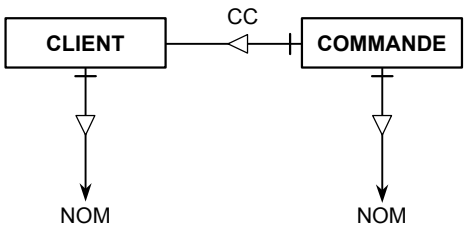


Figure 5.6 - Item redondant

Cette redondance s'exprimera par la contrainte :

(COMMANDE,NOM) = CC.(CLIENT,NOM)

ou encore, de manière équivalente :

pour tout C de COMMANDE, NOM(:C) = NOM(:CLIENT(CC:C))

Cette expression pourrait se paraphraser comme suit : "*le NOM d'une COMMANDE est le NOM du CLIENT de cette COMMANDE*".

REMARQUES

1. la contrainte d'identité des deux types d'associations implique que les membres de celles-ci soient les mêmes deux à deux. Or, dans l'exemple ci-dessus, les deux items concernés (NOM de CLIENT et NOM de COMMANDE) sont distincts. Il faut alors se rappeler qu'un item n'est que le rôle que joue un domaine vis-à-vis d'un type d'articles ou d'un item décomposable. A ces deux items correspondent en fait un même domaine, qui serait par exemple constitué des chaînes de 25 caractères au plus. On généralisera cette notion d'identité de domaines à celle de domaines comparables. Ainsi, un domaine constitué d'entiers et un domaine constitué de réels, étant comparables, pourraient être considérés sur ce plan comme identiques; en toute rigueur, on considérerait qu'ils appartiennent au sur-domaine des valeurs numériques.
2. La notion de redondance n'est pas toujours simple à appréhender. Ce qui semble de prime abord être une contrainte d'expression de redondance n'est parfois qu'une simple contrainte d'intégrité. Ainsi, le schéma 5.7 ne contient aucune redondance, bien que le projet d'un employé dépende du service auquel cet employé est attaché.

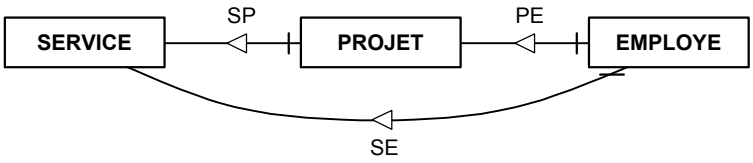


Figure 5.7 - Un schéma sans redondance

En effet, aucun type d'associations ne peut être supprimé sans qu'il s'ensuive une perte d'information. Par contre, il peut exister certaines redondances au niveau des occurrences (si un employé est attaché à un service et à un projet, le service de ce dernier est celui de l'employé). Il convient donc d'exprimer cette propriété par une contrainte d'intégrité telle que :

SP.PE in SE

et non par une expression telle que,

SP.PE = SE

qui exprimerait une redondance, ici inexistante au niveau des types. On vérifiera que l'on a aussi :

PE in SP.SE et PE.SE in SP

Remarque : les notions d'égalité (=, not=) et d'inclusion (in, not-in), que nous avons appliquées à des ensembles d'éléments, sont ici utilisées pour spécifier une relation entre types d'associations. Ceci est possible si l'on considère qu'à un type d'associations correspond un ensemble de couples défini comme suit :

Soit R(A,B) un type d'associations de A vers B; si a de A est associé à b de B par R, alors le couple (a,b) est dit appartenir au type d'associations R. L'expression :

R in S

s'interprète donc comme suit : "*tout couple de R appartient également à S*". On se souviendra aussi que le noyau sémantique du MAG n'est autre qu'un modèle relationnel binaire.

5.4.4 EXPRESSION DE DERIVATION OU D'EQUIVALENCE ENTRE SCHEMAS

Il sera nécessaire, dans plusieurs circonstances, d'exprimer l'équivalence de deux structures de données. Cette équivalence peut être exprimée sur deux plans : le plan strictement sémantique, qui correspond au noyau sémantique du modèle (types d'éléments et d'associations) et le plan des accès, qui correspond au modèle d'accès complet. Ce problème étant traité ailleurs, signalons simplement que l'équivalence sémantique (la seule que nous aborderons ici) est nécessaire à l'équivalence d'accès, mais qu'elle n'est pas suffisante.

Considérons les deux structures suivantes :

1. Les types d'articles CLIENT et COMMANDE; l'item NCLI, identifiant obligatoire de CLIENT; le type d'associations (ici type(s) de chemins) CC de COMMANDE vers CLIENT, de classe fonctionnelle N-1. Soit graphiquement :

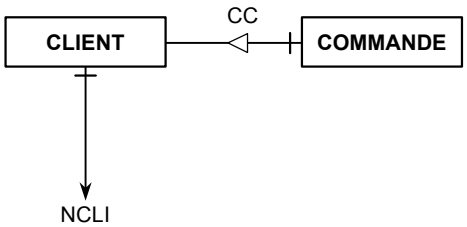


Figure 5.8 - Première structure

2. Les types d'articles CLIENT et COMMANDE; l'item NCLI identifiant obligatoire de CLIENT; l'item NCLIC de COMMANDE, tel que (schéma 5.9) :

NCLIC(:COMMANDE) in NCLI(:CLIENT)

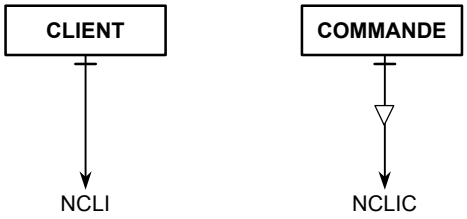


Figure 5.9 - Seconde structure

Il est clair que l'item NCLIC de COMMANDE dans la structure 2 est une représentation de CC dans la structure 1. On exprimera l'équivalence de ces structures en définissant les objets de chaque structure en fonction des objets de l'autre. En se limitant aux objets non communs, on spécifiera les équivalences suivantes :

Expression de la structure 1 (schéma 5.8 en fonction du schéma 5.9) :

CC(COMMANDE,CLIENT) = (COMMANDE,NCLIC).(NCLI,CLIENT)

Expression de la structure 2 (schéma 5.9 en fonction du schéma 5.8) :

(COMMANDE,NCLIC) = CC(COMMANDE,CLIENT).(CLIENT,NCLI)

5.5 LES PRIMITIVES

La description d'un type de données n'est complète que lorsque l'on a énuméré et spécifié les *opérations* permises sur les données de ce type. Nous appellerons PRIMITIVES les opérations de base, définies sur les données, et mises à la disposition de l'utilisateur de la base de données, et en particulier du programmeur. On distingue généralement les primitives d'*accès* aux données, les primitives de *modification* des données, et les primitives *annexes*, liées, le plus généralement, à la notion d'intégrité des données (protection contre les incidents, régulation de la concurrence, macro- primitives). Nous en donnerons une description générale qui couvre les différents types de primitives actuellement disponibles dans les SGD.

5.5.1 LES PRIMITIVES D'ACCES AUX ARTICLES D'UNE BASE DE

DONNEES

L'accès consiste à rendre disponible (à un utilisateur, à un programme, , etc) un ou plusieurs articles présents dans une base de données, cet accès n'étant souvent qu'une étape conduisant à une extraction de valeurs d'items ou à une modification des articles ou de leur environnement. Les différents modes d'accès aux articles d'une base de données peuvent être décrits comme des variantes d'un principe unique:

l'accès aux articles d'une séquence.

On distinguera deux classes de primitives d'accès : celles qui livrent un ENSEMBLE D'ARTICLES (éventuellement ordonné), et celles qui livrent UN SEUL ARTICLE. On trouvera les premières notamment dans des SGBD relationnels pour ce qui concerne la programmation d'application, et dans de nombreux Query Systems. Tous les SGBD offrent des primitives de la deuxième classe, essentiellement dans le cadre des programmes d'application.

Les primitives de la première classe évoquée fournissent une séquence d'articles (ou ensemble, si aucun ordre n'est précisé). L'accès à ces articles est donc spécifié par la séquence de ces articles. Les primitives de la deuxième classe évoquée fournissent un seul article d'une séquence déterminée. L'accès à un article est donc dans ce cas spécifié par la séquence contenant l'article et par la position dans la séquence de l'article désiré. Nous étudierons successivement ces deux spécifications.

SEQUENCE D'ARTICLES

Une séquence d'articles est définie d'une part par l'*ensemble des articles* constituant cette séquence, et d'autre part par l'*ordre* (éventuellement absent, c'est-à-dire aléatoire) dans lequel se présentent ces articles dans la séquence. Dans le cadre de ce modèle, un ensemble d'articles sera défini par une expression du langage de désignation de données (cfr 5.3). L'ordre d'une séquence sera souvent défini comme étant celui d'une séquence naturelle de la base de données (ordre des articles d'un fichier, d'un chemin d'accès par exemple). Il pourra également être défini spécifiquement pour la séquence concernée (comme les ordres triés dans les SGBD relationnels). D'une manière plus précise l'on sera amené à spécifier, implicitement ou explici- tement, une séquence par l'ensemble de ses articles et par l'ordre de ceux-ci.

- *L'ENSEMBLE DES ARTICLES* est désigné par une expression d'ensemble. Si celle-ci contient une condition de sélection cette dernière peut être de deux types. A une condition du premier type correspond un mécanisme d'accès dont l'utilisation conduit à la sélection des articles : accès par clé(s), accès par chemin, articles d'un fichier, et combinaisons de ceux-ci. On dira qu'il s'agit d'une CONDITION EVALUABLE PAR ACCES. Une condition du second type ne peut être évaluée par simple activation de mécanismes d'accès; on appellera cette condition, un FILTRE. Il est clair que l'étendue de ces deux types est une caractéristique du SGD, certains étant plus riches (SGBD relationnels par exemple) que d'autres (SGD COBOL par exemple).

- *L'ORDRE DES ARTICLES* de l'ensemble peut être indifférent, naturel (l'ordre par défaut du sous-ensemble duquel les articles sont extraits), prédéfini autre que naturel (spécifié dans le schéma de la BD), ou enfin non prédéfini (trié sur un critère spécifique à la séquence).

POSITION D'UN ARTICLE

On adoptera les positions traditionnelles suivantes : *i-ème* article à partir du premier, *i-ème* article à partir du dernier, *premier* article, article *suivant*, *dernier* article, article *précédent*.

L'analyse qui précède permet de dériver les primitives d'accès les plus fréquentes dans les SGD. Elle offre également la possibilité de définir simplement et d'une manière uniforme les primitives d'un SGD. Nous donnerons ci-après le jeu minimum de primitives qui permet d'effectuer les accès aux données. Ces primitives seront classées par types traditionnels, chaque type incluant au moins les accès positionnels "*accès au premier*", et "*accès au suivant*". On retiendra les types de base d'accès séquentiel, d'accès par clé et d'accès par chemin. Ces accès correspondent aux PRIMITIVES DE BASE du MAG.

- ACCES AUX ARTICLES D'UN TYPE
  exemple de séquence : CLIENT
- ACCES AUX ARTICLES D'UN TYPE DANS UN FICHIER
  exemple de séquence : CLIENT(:FILE='FICHIER86')
- ACCES PAR CLE
  exemple de séquence : CLIENT(:NOM='DUPONT')
- ACCES AUX CIBLES D'UN CHEMIN DONT ON SPECIFIE L'ORIGINE
  exemple de séquence : LIGNE(CC: COM)
  où COM est la référence à un article COMMANDE,
- ACCES PAR CLE AUX CIBLES D'UN CHEMIN DONT ON SPECIFIE L'ORIGINE
  exemple de séquence : AGENT((AS: SER1) and (: NUMA=32))
  où SER1 est la référence d'un article SERVICE et
  NUMA une clé d'accès dans le type de chemins AS.

5.5.2 LES PRIMITIVES D'ACCES A DES VALEURS

Ces primitives visent principalement à obtenir, pour un article auquel on a accédé :

- les valeurs de certains items;
- le nom du fichier qui le contient;
- le nom de son type.

5.5.3 LES PRIMITIVES DE MODIFICATION DE DONNEES

Les primitives de modification sont plus complexes car elles sont soumises au respect des contraintes d'intégrité affectant les objets modifiés. Nous n'examinerons pas les opérations qui concernent la base de données et les fichiers, pour nous limiter à celles qui concernent les articles, les valeurs d'items et les chemins. Nous nous limiterons encore aux primitives opérant sur un article ignorant celles qui concernent des ensembles d'articles (cas des SGBD relationnels par exemple). On admettra en outre le caractère atomique de ces primitives, en ce sens qu'elles seront exécutées soit complètement soit pas du tout, laissant dans tous les cas la base de données dans un état cohérent, c'est-à-dire respectant les contraintes d'intégrités définies dans son schéma.

CREATION D'UN ARTICLE

Un nouvel article est inséré dans la base de données si sa spécification (fichier d'accueil, valeurs d'items auxquelles il doit être associé, chemins dans lesquels il doit être inséré notamment) est telle que l'article respectera toutes les contraintes d'intégrité de son type.

SUPPRESSION D'UN ARTICLE

Un article supprimé est retiré de toutes les séquences auxquelles il appartient et devient donc inaccessible. Si cette suppression doit entraîner la violation de contraintes d'intégrité, on admet deux modes de comportement du SGD. Le premier consiste à n'effectuer l'opération que si cette exécution n'entraîne aucune violation de contrainte. Le second consiste à effectuer l'opération, puis à poursuivre la suppression de tout article qui par ce fait ne respecterait plus les contraintes de son type.

MODIFICATION DES VALEURS D'ITEMS D'UN ARTICLE

Cette opération consiste à modifier l'ensemble des valeurs d'items attaché à un article. On sera souvent amené à distinguer dans ces items ceux qui jouent un rôle dans la position ou l'existence de l'article (identifiant, clé d'accès, clé de tri, partenaire dans une contrainte), de ceux qui n'en jouent aucun. En tout état de cause, l'opération sera effectuée si son exécution ne conduit pas à la violation de contraintes d'intégrité.

INSERTION D'UN ARTICLE DANS UN CHEMIN

S'il n'en était déjà membre, et pour autant qu'aucune contrainte d'intégrité ne soit violée, l'article spécifié est inséré, comme cible, dans le chemin spécifié. Si un inverse est associé au type du chemin concerné, l'origine du chemin spécifié est inséré dans le chemin du type inverse dont l'origine est l'article spécifié. La position d'insertion est fonction de la spécification de l'ordre des types de chemins.

RETRAIT D'UN ARTICLE D'UN CHEMIN

Si ce retrait n'entraîne la violation d'aucune contrainte d'intégrité, l'article est dégagé de ce chemin. Si le type du chemin spécifié est doté d'un inverse toutes les cibles du chemin inverse dont l'origine est l'article concerné, sont retirés également.

TRANSFERT D'UN ARTICLE D'UN CHEMIN VERS UN AUTRE

Cette opération consiste à effectuer simultanément (c'est-à-dire de manière atomique) le retrait de l'article du premier chemin et son insertion dans le second. Elle concerne les articles d'un type soumis à une contrainte d'existence vis-à-vis du type de chemins.

5.5.4 MACRO-PRIMITIVE (DE MISE-A-JOUR)

Il s'agit d'une séquence d'opérations dont l'exécution est déclarée atomique et telle que si la base de données est dans un état cohérent avant son exécution, elle le sera également à son terme. Par contre on admet, qu'au sein de cette séquence, certaines contraintes d'intégrité soient momentanément violées. Citons à titre d'exemple, selon le schéma 5.3, une séquence qui enregistre une COMMANDE, puis une première LIGNE. Entre ces deux opérations, la base est dans état incohérent puisqu'il existe une COMMANDE sans LIGNE. Cependant, à l'issue de l'exécution de cette séquence, la base a retrouvé son intégrité. Ce mécanisme permet de créer artificiellement des primitives de haut niveau, assurant le respect de n'importe quelle contrainte d'intégrité, qu'elle soit ou non prise en charge par le SGD. Concrètement, une macro-primitive est définie par une instruction d'ouverture et une instruction de clôture. Ces instructions (généralement accompagnées d'une troisième, permettant d'annuler tout effet de la séquence) ont en réalité une portée plus large dans les SGD, car c'est aussi par elles que l'on assure l'intégrité des données vis-à-vis des incidents et en régime de concurrence. Cette séquence atomique est alors appelée "*transaction de bases de données*" ou "*unité logique de traitement*". Nous ignorerons ces aspects ici. On consultera par exemple (GARDARIN,84) et (DATE,81) sur ces points.

5.6 DESCRIPTION STATISTIQUE D'UNE BASE DE DONNEES

Cette description précise la taille des populations d'objets de la base de données. On indiquera le nombre (estimé) d'articles de chaque type et, pour chaque item, le nombre de valeurs distinctes effectivement enregistrées ainsi que les tailles minimum, moyenne et maximum de celles-ci (idéalement la fonction de répartition de ces tailes). En ce qui concerne les types de chemins on déterminera pour chacun la proportion de chemins vides (origines sans cibles) et la proportion d'articles du type cible qui n'appartiennent pas à un chemin. On en déduira la taille moyenne des chemins non vides. Une description plus précise consisterait à donner

pour chaque type de chemins la fonction de répartition de la taille des chemins. Il importe également de relever le taux d'évolution de ces grandeurs dans le temps. Ces informations ne sont pas à récolter sur le terrain, mais sont dérivables, sauf ommision, des quantification du niveau conceptuel.

5.7 SPECIFICATION D'UN SYSTEME DE GESTION DE DONNEES DANS LE MAG

Le MAG est ici utilisé comme modèle de référence pour caractériser les possibilités et les fonctions d'un SGD. Nous spécifierons brièvement trois Systèmes de Gestion de Données : CODASYL 71/73, SQL/DS et COBOL.

5.7.1 SPECIFICATION DES SGBD CODASYL 71/73.

Les spécifications du DBTG de CODASYL proposent des structures de données assez proches de celles du MAG. Bien que les propositions de 1971 et celles de 1973 ne soient pas strictement équivalentes (portée de l'identifiant CALC d'un RECORD TYPE notamment), nous en donnerons une spécification commune. Des descriptions précises et/ou pédagogiques ne manquent pas (DBTG,71), (DBTG,73), (DATE,81), (GARDARIN,84) ainsi que les manuels des systèmes IDMS de Cullinet et ses voisins DBMS-10 et 20 de DEC, PHOLLAS de Philips, UDS1 de Siemens, IDS-2 et dans une certaine mesure IDS-1 de Bull, pour n'en citer que quelques-uns. Nous y renvoyons le lecteur. Les principales restrictions d'un schéma CODASYL par rapport au MAG sont les suivantes :

1. Pas d'items facultatifs;
2. Un seul identifiant par type d'articles dans un fichier (71) ou dans la base de données (73);
3. Une seule clé d'accès par type d'articles dans un fichier;
4. Tout identifiant du type 2 est une clé d'accès du type 3;
5. Tout type de chemins est 1-N ou N-1 et est doté d'un inverse;
6. Les types de chemins 1-N sont mono-origines;
7. Origine et cibles d'un type de chemins sont des types d'articles distincts;
8. Dans un type de chemins, une contrainte d'existence ne peut être posée que sur les cibles d'un type de chemins 1-N.

Sont par conséquent admis par exemple, les identifiants locaux à un type de chemins, les clés d'accès dans un type de chemins, les types de chemins multi-cibles. Par ailleurs, l'essentiel de la terminologie propre à CODASYL s'établit comme suit. Un (type d')articles est un RECORD (TYPE). Un item simple et élémentaire est un DATA ITEM. Un item répétitif et élémentaire est un VECTOR. Un item décomposable, répétitif ou non est un REPEATING

GROUP. VECTOR et REPEATING GROUP sont des DATA AGGREGATES. Un type de chemins ET son inverse constituent un SET TYPE; l'origine du type 1-N s'appelle OWNER et les cibles MEMBERS. Une contrainte d'existence consiste à déclarer le MEMBER AUTOMATIC MANDATORY, et son absence correspond à un MEMBER MANUAL OPTIONAL. Un fichier s'appelle AREA ou REALM, selon les contextes. A chaque article de la base de données est associé un identifiant interne (non significatif) qui est aussi une clé d'accès : la DATABASE KEY.

Les principales primitives offertes par les SGBD CODASYL 73 sont les suivantes (on précisera, par classe, les positions que l'on peut spécifier) :

1. Accès séquentiel aux articles (éventuellement d'un seul type) d'un fichier (6 positions);
2. Accès aux articles d'un type correspondant à une valeur de clé d'accès dans un fichier (premier, suivant);
3. Accès aux articles cibles (éventuellement d'un seul type) d'un chemin 1-N (6 positions), et à l'article cible d'un chemin N-1;
4. Accès par clé aux articles cibles d'un chemin 1-N (premier, suivant);
5. Accès par l'identifiant interne;
6. Obtention, pour un article, de valeurs d'items, du nom de son type, du nom de son fichier, de la valeur de l'identifiant interne;
7. Création d'un article;
8. Suppression d'un article, avec suppression éventuelle d'articles qui lui seraient associés par des chemins (et ainsi récursivement);
9. modification de valeurs d'items;
10. insertion, retrait et transfert d'un article cible d'un chemin 1-N.

5.7.2 SPECIFICATION D'UN SGBD RELATIONNEL

Nous choisirons le SGBD SQL/DS d'IBM qui est assez représentatif d'une classe de SGBD relationnels. Les spécifications qui vont suivre sont donc, pour l'essentiel, généralisables à d'autres SGBD (on citera par exemple ORACLE, SQL de CDC, UNIFY et dans une mesure moindre des petits systèmes comme OPEN ACCESS). On trouvera dans la plupart des ouvrages généraux une description d'un système relationnel, ou même de plusieurs : (DATE,81), (GARDARIN,84), (DELO-ADI,82). Les principales restrictions des structures relationnelles par rapport à celles du MAG sont les suivantes.

1. Il n'y a pas de types de chemins;
2. Les items sont simples (non répétitifs) et élémentaires (non décomposables);
3. Les items sont obligatoires (mais possibilité de valeur NULL dont l'interprétation est libre);
4. Il y a au moins un item par type d'articles;

5. Tout identifiant est une clé d'accès et une clé de tri (ces deux dernières notions peuvent souvent être ignorées du programmeur).
6. Pas d'article Système, ni de notion de fichier ni de base de données (sauf au niveau technique pour ces deux derniers). Cependant, la notion de type d'articles est très similaire à celle de fichier à un seul type d'articles;
7. Pas de contraintes d'intégrité autres que sur le type de valeurs d'un item; en particulier pas de contrainte d'inclusion (dite référentielle), du moins actuellement.

La terminologie relationnelle est évidemment spécifique. Un type d'articles est appelé TABLE ou RELATION. Un article est un TUPLE ou ROW ou LIGNE. Un domaine de valeurs est un DOMAIN (absent en SQL) et un item est appelé COLUMN ou ATTRIBUTE. Une clé d'accès est généralement un INDEX, et un identifiant est une KEY ou UNIQUE INDEX.

Les primitives offertes sont plus puissantes que dans les autres SGBD, même si l'on s'en tient au domaine de la programmation d'application. Certaines primitives opèrent sur des ensembles d'articles, d'autres sur un seul article à la fois. Les primitives qui opèrent sur des ensembles d'articles sont les primitives de modification. Il existe des primitives opérant sur un article à la fois pour toutes les opérations de base. Une caractéristique essentielle des SGBD relationnels est qu'il admettent la spécification de séquences par filtre (cfr 5.5.1); quant à l'ordre de cette séquence, il est soit aléatoire, soit trié sur une clé de tri non nécessairement prédéfinie.

1. accès aux articles d'une séquence éventuellement filtrée (premier, suivant) et à leurs valeurs d'items.
2. créer des articles dont les valeurs d'items sont une copie des valeurs d'articles dont on donne un filtre (cfr 5.5.1).
3. supprimer les articles qui vérifient un filtre.
4. modifier les valeurs d'items des articles qui vérifient un filtre.
5. créer, supprimer un article; modifier les valeurs d'items d'un article.

5.7.3 SPECIFICATION DU SGD COBOL ANSI-74

Nous adopterons les spécifications ANSI 1974 (CLARINVAL,81). Les principales restrictions par rapport au MAG des structures de données offertes au programmeur sont les suivantes :

1. A tout type d'articles est associé au moins un item;
2. Tout item est obligatoire;
3. Il n'y a pas de types de chemins;
4. Il n'y a pas de type d'articles système;
5. Tout identifiant est une clé d'accès;
6. S'il y a des clés d'accès, une au moins est identifiante;

7. Toute clé d'accès est aussi une clé d'ordre triée par conséquent, tout préfixe d'une clé d'accès est aussi une clé d'accès;
8. Un type d'articles est associé à un seul fichier;
9. Tous les types d'articles d'un même fichier ont mêmes identifiants et mêmes clé d'accès;
10. Une clé d'accès est un item (éventuellement décomposable ou composant);
11. Il n'y a pas de notion de base de données.

Les principales primitives se résument comme suit.

1. Accès séquentiel aux articles d'un fichier (premier, suivant; pour les fichiers sans clé d'accès : dernier, précédent) ainsi qu'aux valeurs de tous les items. Lorsqu'il existe au moins une clé d'accès, l'accès se fait selon la clé de tri correspondant à l'une d'elles (N.B: dans un fichier multi-type on ne peut connaître le type de l'article lu).
2. Accès aux articles dont la valeur de clé est égale, supérieure, inférieure, à une valeur donnée. La séquence est celle de la clé (cfr 7 ci-dessus);
3. Création d'un article (le dernier de la séquence si pas de clé d'accès; quelconque sur ce critère si au moins une clé d'accès);
4. Suppression d'un article (si au moins une clé d'accès)
5. Modification des valeurs d'item d'un article (sauf identifiant).

5.8 NOTION DE SCHEMA CONFORME

La description des structures d'une base de données exprimée selon les concepts MAG (c'est-à-dire son schéma MAG) est dit conforme à un SGD si les constructions de ce schéma obéissent à la spécification de ce SGD. On pourra ainsi évaluer la conformité d'un schéma déterminé aux structures COBOL, CODASYL 71, relationnelles, etc.

Chapitre 6 : LANGAGE DE SPECIFICATION D'ALGORITHMES D'ACCES

6.1 INTRODUCTION

De même que l'existence du MAG se justifie par la nécessité d'exprimer des structures de données indépendamment de celles des SGD, celle d'un langage d'expression d'algorithmes est rendue nécessaire dans une démarche de conception des traitements qui se veut générale et donc indépendante des langages de programmation effectifs. Si les "pseudo- langages", ainsi qu'on les désigne souvent, sont largement répandus, ils pèchent en particulier en ce qui concerne l'accès aux données à structure complexe et leur gestion. Ainsi en est-il des diagrammes de Nassi- Schneiderman, des ordinogrammes et autres pseudo-langages dits de programmation structurée. Certains langages généraux d'accès aux données ont été définis, tel UDL (DATE,81). Dans le domaine relationnel, on citera également PASCAL/R (SCHMIDT,78), qui intègre à PASCAL les types et des opérateurs relationnels. Quelques SGBD offrent des structures algorithmiques pilotées par les données : INGRES, RDB et DATATRIEVE (DEC), SOCRATE (SYSECA) pour ne citer que quelques exemples.

LDA (Langage de Description d'Algorithmes) tente de réaliser l'intégration de primitives d'accès aux données dans des structures algorithmiques traditionnelles. Basé sur le MAG, il permet d'exprimer tous les types d'accès, depuis l'accès filtré (cfr. SGBD relationnel par exemple) jusqu'aux accès ponctuels de bas niveau. On lui associera en outre des règles de transformation systématiques, que nous étudierons, ainsi que des expressions de calcul du coût probable d'un algorithme (HAINAUT,76), (HAINAUT,77).

De par l'objectif de ce volume, nous nous limiterons essentiellement aux aspects de LDA liés à la gestion des données.

6.2 DESIGNATION DE DONNEES EN LDA

Les principes de désignation de données ont déjà été développés en 5.3. Nous les adopterons tels quels, en précisant toutefois qu'en raison de la nature algorithmique du langage, les collections de données se présenteront toujours sous forme de séquences, c'est-à-dire d'ensembles ordonnés. Parmi les variables, outre les types traditionnels, nous introduirons surtout les variables d'articles, dont la valeur est la référence d'un article de la base de données, et la variable liste, qui peut contenir une séquence de valeurs de longueur quelconque.

6.3 STRUCTURES ALGORITHMIQUES EN LDA

On ne présentera surtout ici que les points sur lesquels LDA s'écarte des structures traditionnelles telles qu'on les trouvera dans PASCAL par exemple.

6.3.1 LA CONDITION LDA

Elle inclut la condition de sélection présentée en 5.3.5. Elle sera étendue aux fonctions à valeur booléenne. On admettra également la forme simplifiée suivante (cfr. 5.3.4.) :

ENS/VAR CSEL ou **ENS CSEL**

en lieu et place de :

ENS/VAR CSEL not= ()

Exemples :

- CLI(:NOM="DUPONT")
  est vraie si l'article désigné par CLI est associé à un NOM égal à "DUPONT";
- PRODUIT(PL: LIGNE)
  est vraie s'il existe au moins un PRODUIT associé à au moins une LIGNE;
- CLI
  est vraie si CLI désigne un article CLIENT.

Une condition valide sera dénotée ci-dessous par CONDITION.

6.3.2 LA SEQUENCE D'INSTRUCTIONS

Une séquence est une suite de zéro, une ou plusieurs instructions. Une séquence valide sera dénotée ci-dessous par SEQUENCE. On terminera par ";" chaque instruction d'une séquence.

6.3.3 L'ASSIGNATION

Cette instruction est étendue aux variables d'articles et aux variables listes.

Exemples :

- CLI := CLIENT(:NCLI = X);
- LISTE-ENTIERS := (1, 5, NCLI(:CLI), 100..120);
- LISTE-DDJ := COMMANDE(:DATE = DDJ);
- CLI := ();

6.3.4 L'ALTERNATIVE

Deux formes :

if CONDITION **then** SEQUENCE1 **else** SEQUENCE2 **endif**
if CONDITION **then** SEQUENCE1 **endif**

Exemple

```
• if CLI(CC: O COMMANDE) then
        print NUMC(:CLI), "sans commande";
  endif
```

6.3.5 LA BOUCLE WHILE

Forme

while CONDITION **do** SEQUENCE **endwhile**

Exemple

```
• while not (I = O) do
        create L := LIGNE( ... );
        I := I-1;
  endwhile;
```

6.3.6 LA BOUCLE ENUMERATIVE

Forme

for VAR := ORD SEQ-ELEMENTS **while** CONDITION **do**
    SEQUENCE1
if-no VAR **then**
    SEQUENCE2
endfor

où SEQ-ELEMENTS est l'expression d'une séquence d'éléments (constantes, valeurs de variables ou d'items, articles);

VAR est le nom d'une variable du type de SEQ-ELEMENTS; celle-ci est appelée variable de la boucle;

ORD (facultatif) est l'expression d'un entier ou d'un intervalle d'entiers. Son absence désigne tous les entiers.

les expressions "if-no VAR then SEQUENCE2" et "while CONDITION" sont

facultatives;

Fonction pour chacun des éléments de SEQ-ELEMENTS dont le rang appartient à ORD, et tant que la condition CONDITION est vraie, cet élément est assigné à VAR et SEQUENCE1 est exécutée. Si aucun élément ne satisfait à la spécification ORD SEQ-ELEMENTS, la séquence SEQUENCE2 est exécutée une fois.

Remarque une boucle énumérative définie sur une séquence d'éléments d'une base de données sera aussi appelée boucle d'accès.

Exemples :

- ```
  for I := 1..10 do
          print I;
  endfor;
  ```
- ```
  for C := COMMANDE(CC : CLI) do
          print DATE(:C), NCOM(:C);
  endfor;
  ```
- ```
  for L := LIGNE (CL: COMMANDE(:NCOM=X)) do
          LIGNE-FACTURE(L);
  if-no L then
          print "Commande",X,"erronée";
  endfor;
  ```
- ```
  for C := #1..10 CLIENT (: ADRESSE = (*, *, "Aix")) sorted(NOM) do
          print NOM(:C);
  endfor;
  ```

Cette dernière instruction imprime le nom des 10 premiers CLIENTS d'Aix, par ordre croissant de NOM.

L'accès à l'article système de la base de données (5.2.5) s'interprête comme l'accès à la base de données elle-même. C'est dans le corps de cette boucle que la base de données est disponible. Outre cette fonction de définition de contexte, cette boucle pourra jouer deux autres rôles : permettre d'emprunter des chemins ayant cet article pour origine, et résoudre les ambigu~ités de désignation qui peuvent surgir lorsqu'un algorithme travaille sur deux bases de données contenant des types d'articles de même nom. On considère alors qu'il existe un type d'associations virtuel, sans nom, entre le type d'articles système et chaque type d'articles de son schéma.

6.3.7 LES MODIFICATEURS DE BOUCLES

Formes

next VAR

exit VAR

où VAR (facultatif) est le nom d'une variable de boucle énumérative.

Fonction

- NEXT provoque l'itération prématurée de la boucle énumérative qui utilise VAR; si VAR est absent, l'itération concerne la boucle WHILE ou énumérative la plus intérieure dans laquelle se trouve l'instruction.
- EXIT provoque la sortie prématurée de la boucle concernée (mêmes règles que ci-dessus).

Exemple :

- ```
  QT := 0;
  for L := LIGNE (PL : PRO) do
          if QT + Q(:L) > QSTK(:PRO) then exit endif;
          QT := QT + Q(:L);
  endfor;
  ```

  cet algorithme peut également s'écrire :

  ```
  QT := 0;
   for L := LIGNE (PL : PRO) while QT+Q(:L) <= QSTK(:PRO) do
          QT := QT + Q(:L);
  endfor;
  ```

6.3.8 LES PROCEDURES ET LES FONCTIONS

La procédure LDA sera l'un des modes de représentation d'un module. Les arguments passés à une procédure (fonction) sont des variables (y compris du type liste), des constantes ou des expressions à valeur unique (à l'exception des fonctions first et next). Parmi les procédures prédéfinies, on compte des procédures de gestion de liste :

newlist(L)	qui initialise à vide la liste L.
addlist(L,E)	qui ajoute à la liste L l'élément E.
unique(L)	qui élimine les doubles de L.

Il existe également des fonctions spéciales fournissant respectivement le premier et le suivant (du dernier obtenu) des éléments d'une séquence. Ces fonctions sont :

first(SEQ-ELEMENTS) et next(SEQ-ELEMENTS)

où SEQ-ELEMENTS est une expression de désignation d'une séquence d'éléments.
  ```

6.3.9 LES BRANCHEMENTS

LDA admet la notion d'étiquette et celle de branchement.

6.3.10 FORMES PRIMITIVES DES INSTRUCTIONS STRUCTUREES

La démarche de conception des traitements pourra conduire à transformer un algorithme de manière à y éliminer certaines structures. Nous donnerons ci-après des schémas équivalents de quelques instructions structurées.

```
for VAR := SEQ-ELEMENTS  do
        SEQUENCE;                ⇔   VAR := ();
endfor;                              V := first(SEQ-ELEMENTS);
                                     while not V = () do
                                             VAR := V;
                                             SEQUENCE;
                                             V := next(SEQ-ELEMENT);
                                     endwhile;
```

```
while CONDITION do
        SEQUENCE;                ⇔   TESTW :
endwhile;                                f not CONDITION then
                                                 goto FINW; endif;
                                         SEQUENCE;
                                         goto TESTW;
                                     FINW :
```

```
if CONDITION
then SEQUENCE1;                  ⇔           if not CONDITION then
else SEQUENCE2;                                  goto FAUX; endif;
endif;                                   SEQUENCE1;
                                         goto FINIF;
                                     FAUX  :
                                         SEQUENCE2;
                                     FINIF :
```

```
if CONDITION
then SEQUENCE;                   ⇔           if not CONDITION then
                                                 goto FINIF;
                                         SEQUENCE;
                                     FINIF :
```

6.4 COMMANDE DE MISE-A-JOUR DES DONNEES

Ces commandes spécifient des opérations de modification de la base de données. Une opération est exécutée complètement si cette exécution laisse la base dans un état cohérent (c'est-à-dire vérifiant les contraintes d'intégrité). Elle n'est pas exécutée du tout dans le cas contraire. L'opération correspondant à une commande est donc atomique; elle constitue une primitive (voir 5.5.3). LDA possède trois commandes de modification permettant de créer, supprimer et modifier un article.

6.4.1 CREATION D'UN ARTICLE

Forme

create VAR := NOMTA CONDITION

où NOMTA est le nom d'un type d'article,

CONDITION (facultative) est une expression booléenne de conditions simples, liées par des "and", spécifiant à quelles valeurs et à quels articles sera associé l'article créé.

Fonction

créer un article qui vérifie la condition CONDITION. L'article ne sera créé que si toutes les contraintes d'intégrité qui lui sont relatives sont vérifiées. Dans ce cas, VAR désigne cet article.

Exemple

```
create L := LIGNE ((:QCOM = XQ)
                   and (CL : COM)
                   and (PL : PRODUCT(: NPRO = XP)))
```

6.4.2 SUPPRESSION D'UN ARTICLE

Forme

delete VAR CONDITION

Fonction

supprimer l'article désigné par VAR s'il vérifie la condition CONDITION. Si cette suppression entraîne une violation de contrainte d'existence pour certains articles, ceux-ci sont également supprimés.

Exemples :

- delete CLI;

supprime le CLIENT CLI ainsi que tous les articles COMMANDE qui lui sont attachés ainsi que toutes les LIGNES de ces COMMANDES.

- delete CLI(CC : 0 COMMANDE);
  supprime l'article CLI s'il n'est plus associé à des articles COMMANDE.

6.4.3 MODIFICATION D'UN ARTICLE

Forme

modify VAR CONDITION

Fonction

opérer les transformations nécessaires pour que l'article désigné par VAR vérifie la condition CONDITION. Ces transformations ne peuvent entraîner la création ou la suppression d'articles.

Exemples (basés sur les schémas 5.2 et 5.3) :

- modify P((:NOM = 'DUPONT') and (:ADRESSE(:RUE = X))) :
  modifier NOM et RUE d'ADRESSE de l'article PERSONNE P
- modify COM((:DATE = DDJ) and (CC : CLI2)) :
  changer la DATE et le CLIENT (via CC) de l'article COMMANDE COM
- modify DEP(STOCK : PRODUIT(:NPRO = N)) :
  associer le DEPOT DEP au PRODUIT dont NPRO = N
- modify DEP(STOCK : O PRODUIT) :
  dissocier le DEPOT DEP de tous ses PRODUIT
- modify DEP(PRODUIT(STOCK: DEP1) in PRODUIT(STOCK: DEP)) :
  associer au DEPOT DEP tous les PRODUIT du DEPOT DEP1

6.5 LES MACRO-PRIMITIVES

Les commandes suivantes permettent de définir une séquence d'instructions comme étant atomique, c'est-à-dire se comportant comme une primitive ("macro-primitive") exécutée complètement ou pas du tout (cfr 5.5.4). Le début d'une séquence est indiqué par :

begin-transaction

et se clôture par :

close-transaction

Tout incident survenant lors de l'exécution de cette séquence provoque la poursuite de l'exécution à l'instruction qui suit le "close-transaction", la base de données étant dans l'état qui précédait l'exécution de "begin-transaction". L'effet d'un tel incident (retour à l'état initial) peut aussi être programmé dans la séquence par l'exécution de

abort-transaction

Exemple :

Soit à transférer une somme de 1000 Fr. d'un compte vers un autre. Ce transfert se programmera par un retrait suivi d'un dépôt, cette séquence étant ininterruptible.

```
read NC1; read NC2;
begin-transaction;
        C1 := COMPTE(:NCOMPTE = NC1);
        if C1=() then abort-transaction; endif;
        modify C1(:MONTANT = MONTANT(:C1) - 1000);
        C2 := COMPTE(:NCOMPTE = NC2);
        if C2=() then abort-transaction; endif;
        modify C2 (:MONTANT = MONTANT(:C2) + 1000);
end-transaction;
```

6.6 ALGORITHMES CONFORMES

Un algorithme ADL est conforme à un SGD :

1. s'il travaille sur un *schéma conforme* à ce SGD (voir 5.8);
2. si les expressions de désignation et de modification de données qu'il spécifie sont *totalement évaluables* et exécutables par les primitives de ce SGD (voir 5.7);
3. si l'*enchaînement* de ces primitives est *conforme* à ce qu'autorise le SGD.

Cette troisième condition sera contraignante dans le cas de certains SGD qui limitent les combinaisons d'opérations sur les données.

Ainsi en COBOL, on ne peut trouver à la fois des accès et des créations d'articles dans un fichier sans clé d'accès (c'est- à-dire séquentiel). En DL/1 d'IMS (IBM), le parcours d'une structure d'arbre est strictement descendante et l'on ne peut parcourir plus de deux DB physiques d'affilée. A titre de dernier exemple, certains langages relationnels n'autorisent pas l'emboitement d'une boucle d'accès dans une autre (QUEL d'INGRES et de ses dérivés par exemple).

Considérons le schéma 6.1 qui est conforme à la fois à COBOL, CODASYL 73 et au relationnel (SQL/DS) et évaluons la conformité de quelques boucles énumératives.

```
for CLI := CLIENT do                conforme à COBOL (5.7.3, primitive 1)
```

```
        SEQUENCE;                  conforme à SQL/DS (5.7.2, primitive 1)
endfor;                            conforme à CODASYL (5.7.1, primitive 1)

for CLI := CLIENT(:NOM = X) do     conforme à SQL/DS (5.7.2, primitive 1)
        SEQUENCE;                  non conforme à COBOL et CODASYL
endfor;

for COM := COMMANDE(:NCLI = NCLI(: CLIENT(: NOM = X))) do
        SEQUENCE;                  conforme à SQL/DS;
endfor;                            non conforme à COBOL et CODASYL

for COM := COMMANDE(:NCOM > X) do  conforme à COBOL (5.7.3, primitive 2)
        SEQUENCE;                  conforme à SQL/DS (5.7.2, primitive 1)
endfor;                            non conforme à CODASYL
```

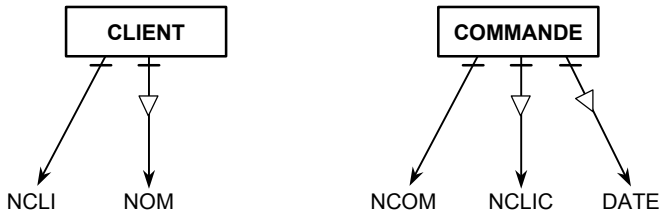


Figure 6.1 - Un schéma conforme à COBOL, CODASYL et SQL

Chapitre 7 : LES TRANSFORMATIONS

7.1 INTRODUCTION

De l'expression conceptuelle d'une solution à sa forme exécutable en machine, il existe un certain nombre d'expressions, qui toutes décrivent implicitement les mêmes concepts, mais dont la forme est liée à l'intégration de certaines contraintes logiques, techniques ou organisationnelles. Le nombre de ces expressions et les contraintes ou critères pris en charge sont spécifiques d'une démarche particulière. Nous proposons dans ce chapitre des règles systématiques permettant la génération d'une expression à partir d'une autre. Ces règles concernent la transformation de structures de données et celle d'algorithmes. Elles garantissent l'équivalence sémantique des expressions (c'est-à-dire la conservation de la spécification d'origine), certaines d'entre elles étant même réversibles (transformations 7.3). Si les jeux de règles proposés ne sont pas exhaustifs, il couvrent cependant l'essentiel des cas pouvant se présenter en pratique. La littérature propose des règles de transformation, en particulier en ce qui concerne les structures de données. Elles sont cependant souvent spécifiques à une classe de SGBD. C'est pourquoi nous les citerons plutôt au chapitre 8.

7.2 TRANSFORMATION D'UN SCHEMA CONCEPTUEL EN SCHEMA MAG

Les règles qui suivent sont destinées à produire, à partir d'un schéma Entité/Association, un schéma MAG qui n'a d'autres propriétés que d'être correct, et le plus proche possible du schéma E/A. On n'utilisera que le noyau sémantique du MAG, la spécification des accès étant prématurée à ce niveau. En d'autres termes, nous n'introduirons pas de spécification concernant clés d'accès, types de chemins inverses, ordres et fichiers. En outre, le sens des accès étant indifférent, nous n'orienterons pas ceux-ci, sauf pour des raisons de distinction des rôles dans un type d'associations récursif.

7.2.1 ENTITE et TYPE d'ENTITES

On représentera une entité par un article, et un type d'entités par un type d'articles.

7.2.2 VALEUR D'ATTRIBUT et ATTRIBUT d'un TYPE d'ENTITES

On représentera une valeur d'attribut par une valeur d'item et un attribut par un item associé au type d'articles représentant le type d'entités. La répétitivité, la décomposabilité et le caractère facultatif de l'item découlent immédiatement des propriétés de l'attribut.

7.2.3 TYPE D'ASSOCIATIONS BINAIRE SANS ATTRIBUT

Il y correspondra un type de chemins entre les types d'articles correspondants dont la classe fonctionnelle et les contraintes d'existence se déduisent de la connectivité du type d'associations.

7.2.4 TYPE D'ASSOCIATIONS au moins TERNAIRE, SANS ATTRIBUT

Chaque association est représentée par un article et le type d'associations par un type d'articles. On établira de ce type d'articles vers chaque type d'articles correspondant aux membres du type d'associations, un type de chemins N-1, obligatoire pour le nouveau type d'articles.

Remarque importante : si le type d'associations n'est le siège d'aucune dépendance fonctionnelle, multivaluée ou de jointure, la transformation proposée est la seule qui soit correcte. Toute solution basée sur une décomposition binaire du type d'associations est gravement erronée.

7.2.5 TYPE D'ASSOCIATIONS AVEC ATTRIBUTS

Le type d'associations est représenté comme en 7.2.4., même s'il est binaire. Chaque attribut est représenté par un item attaché au nouveau type d'articles. Si le type d'associations est binaire et de connectivité (i-1, k-*), le nouveau type de chemins concernant le type d'articles du côté i-1 sera de classe fonctionnelle 1-1. Ce dernier cas peut aussi être représenté plus simplement par un type de chemins 1-N, les items représentant les attributs étant alors attachés au type d'articles du côté i-1.

Remarquerons que la représentation d'un type d'associations par un type d'objets (comme en 7.2.4 et ici-même) est une application du principe d'agrégation, qui est l'un des principaux mécanismes d'abstraction dans le domaine de la modélisation conceptuelle (SMITH,77), (CODD,79).

7.2.6 IDENTIFIANTS et autres CONTRAINTES D'INTEGRITE

On déterminera les identifiants éventuels de chaque type d'articles en fonction des propriétés des types d'entités et des types d'associations qu'ils représentent. En particulier, si les types d'associations transformés en 7.2.4 et 7.2.5 ont pour identifiant l'ensemble des types d'entités membres, alors le type d'articles correspondant a pour identifiant les types d'articles représen-

tant les membres, via les nouveaux types de chemins. On établira de même l'expression MAG des autres contraintes d'intégrité du schéma E/A.

7.2.7 UN EXEMPLE SIMPLE

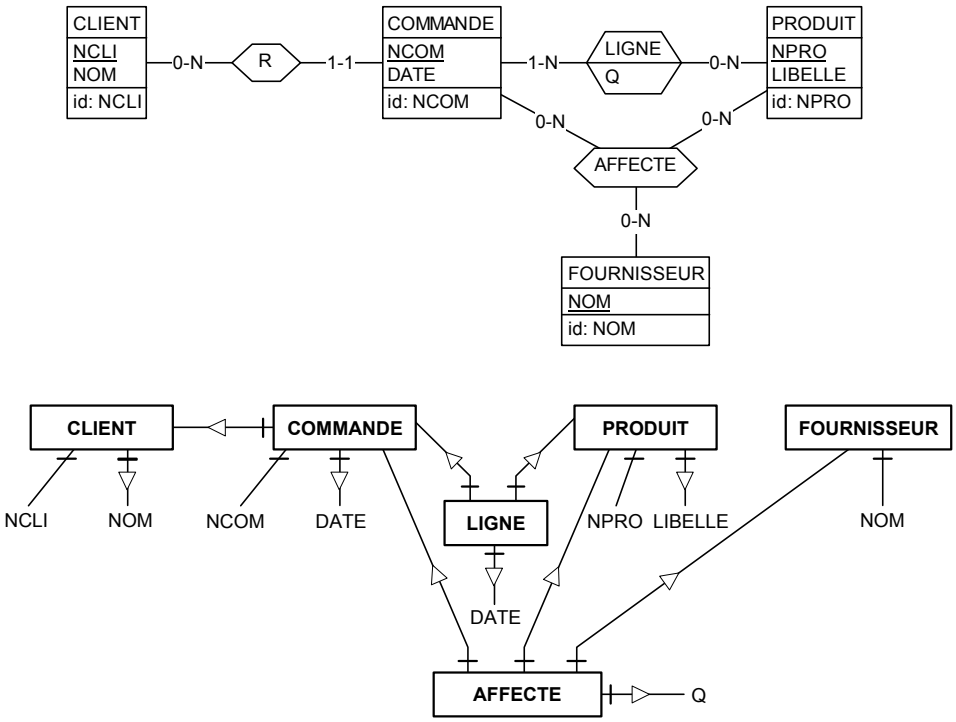


Figure 7.1 - Transformation en MAG d'un schéma Entité/Association

7.3 TRANSFORMATION DE SCHEMAS MAG

Il s'agit ici de produire, à partir d'un schéma MAG, un autre schéma MAG, équivalent au premier, mais qui respecte des contraintes que violait le premier. Un exemple d'application est la production de schémas conformes à un SGD à partir de schémas qui ne le sont pas. Nous nous limiterons ici aux transformations réversibles et irredondantes (pour être plus précis, n'introduisant pas de redondances supplémentaires). L'équivalence sera d'abord réalisée du point de vue sémantique puis du point de vue des accès, car les deux aspects sont distincts. Les transformations, que nous représenterons graphiquement, seront définies sur le noyau sé-

mantique du MAG. Lorsqu'un type d'objets peut être soit un item, soit un type d'articles, il sera représenté par un cercle. On observera enfin qu'une propriété définie sur un type de chemins N-N (resp. 1-N ou N-1) est aussi d'application pour un type de chemins 1-N, N-1 et 1-1 (resp. 1-1).

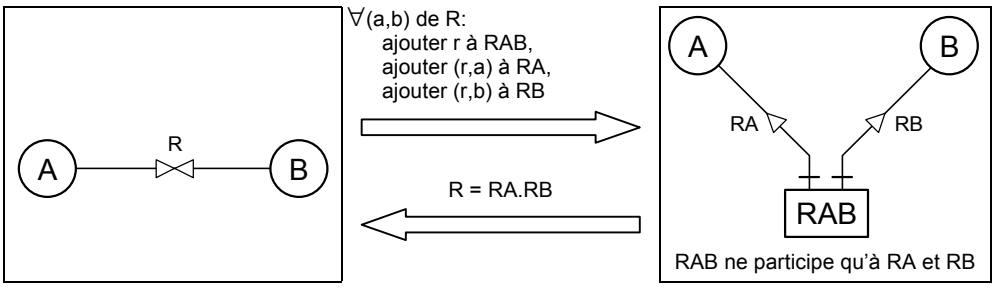
Les transformations se classent en deux familles : création/suppression d'un type d'articles, et rotation de types d'associations. Le lecteur intéressé trouvera dans (HAINAUT,81) une étude plus générale de ces transformations.

7.3.1 CREATION/SUPPRESSION D'UN TYPE D'ARTICLES

Ces transformations sont basées sur le principe d'agrégation et désagrégation déjà cité en 7.2.5.

PREMIERE TRANSFORMATION (Transf. 7.1)

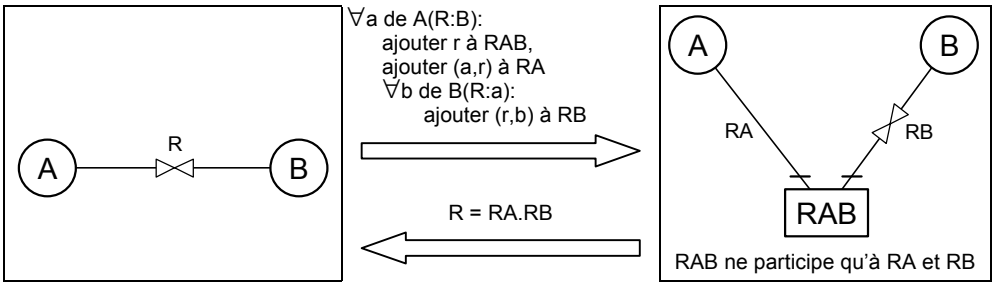
Il s'agit de l'expression précise d'un procédé classique d'élimination d'un type d'association N-N. Bien qu'applicable, elle offre peu d'intérêt pour les autres classes fonctionnelles.



Transfo 7.1 - Création/suppression d'un type d'articles (1)

DEUXIEME TRANSFORMATION (Transf. 7.2)

Bien qu'anodine en apparence, cette transformation est la généralisation de nombreux procédés couramment utilisés : élimination de types de chemins récursifs, d'items répétitifs, facultatifs, de clés d'accès non identifiantes, de deuxièmes clés d'accès, etc.



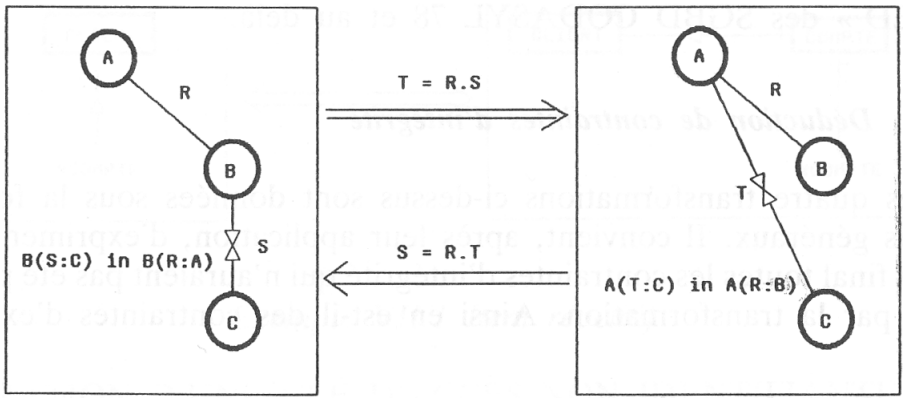
Transfo 7.2 - Création/suppression d'un type d'articles (2)

7.3.2 ROTATION D'UN TYPE D'ASSOCIATIONS

Intuitivement le schéma est le suivant : "si A est un identifiant de B, et si C est associé à B, on peut tout aussi bien l'associer à A". Nous ne présenterons que les deux cas particuliers principaux, d'ailleurs aisément généralisables. Ils serviront en particulier à éliminer des types de chemins, à diminuer le nombre de niveaux d'une hiérarchie, à effectuer des conversions CODASYL/relationnel et inversement.

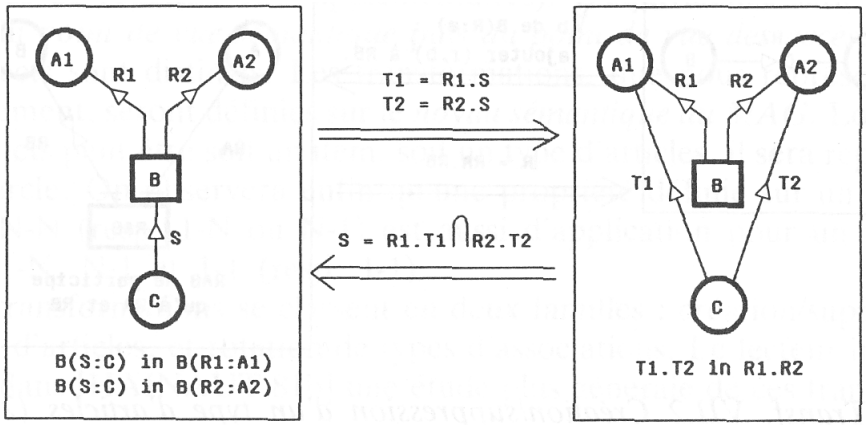
ROTATION SIMPLE D'UN TYPE D'ASSOCIATIONS (Transf. 7.3)

Dans cette transformation, la contrainte d'intégrité explicite du schéma de gauche est vérifiée si R est obligatoire pour B. On remarquera par ailleurs que les deux schémas sont en fait identiques à la permutation près de A et B.



Transfo 7.3 - Rotation simple d'un type d'associations

ROTATION MULTIPLE D'UN TYPE D'ASSOCIATIONS (Transf. 7.4)



Transfo 7.4 - Rotation multiple d'un type d'associations

Ici encore les contraintes explicites du schéma de gauche sont vérifiées si R1 et R2 sont obligatoires pour B. On notera que S est 1-N et (malheureusement) pas N-N comme dans la transformation 7.3.

Remarque : Les transformations 7.3 et 7.4 ne sont valables que moyennant une contrainte d'intégrité dynamique supplémentaire : si l'on change l'identifiant d'un B, il faut aussi changer les A, A1, A2 associés aux C de ce B. .P/JP Cette contrainte constitue l'un des problèmes posés par l'approche relationnelle, car les SGBD ne prennent généralement pas en charge sa vérification, quand bien même ils assurent la gestion des contraintes référentielles (qui constituent un cas particulier des contraintes d'inclusion vues en 5.4.2). Une manière extrême d'en garantir la validité est de figer les associations R et R1,R2 concernant chaque élément de B. Certains SGBD offrent des possibilités en cette matière, par exemple le mode de rétention "FIXED" des SGBD CODASYL 78 et au delà.

7.3.3 DEDUCTION DE CONTRAINTES D'INTEGRITE

Les quatre transformations ci-dessus sont données sous la forme de schémas généraux. Il convient, après leur application, d'exprimer dans le schéma final toutes les contraintes d'intégrité qui n'auraient pas été prises en charge par la transformation. Ainsi en est-il des contraintes d'existence.

7.3.4 EQUIVALENCE D'ACCES

Les transformations ci-dessus assurent une équivalence sémantique. Il faut ensuite orienter les types d'associations de manière à ce que les accès du schéma initial soient présents dans le schéma final. Dans certaines situations, cette spécification peut s'avérer délicate pour la transformation 7.4 (par exemple si A1 et A2 sont des types d'articles). La détermination des accès relève alors de procédés propres à la démarche de conception (8.4.4 et 8.4.5).

Lorsqu'un type d'associations est doté d'un inverse, il est possible de procéder différemment selon que l'on considère ces deux types globalement (perception sémantique) ou séparément (perception d'accès). Lorsqu'on les considère globalement, on transforme le type d'associations, sans distinguer les deux sens, puis on spécifie l'orientation de manière à obtenir l'équivalence d'accès. Lorsqu'on les considère séparément, chacun des types inverses est transformé indépendamment de l'autre, éventuellement selon des techniques différentes. La redondance initiale (les deux accès inverses ont même sémantique) est en général rendue explicite. Il conviendra de l'exprimer par une contrainte d'intégrité. On trouvera dans l'exemple 7.6 ci-dessous une application de ces deux optiques.

7.3.5 QUELQUES EXEMPLES DE TRANSFORMATIONS

On présentera quelques applications utiles de ces principes. Le lecteur déterminera aisément les transformations qui y sont utilisées. On fera remarquer que l'élimination de certaines constructions peut réclamer plusieurs étapes de transformation. Par exemple, l'élimination d'un item répétitif en relationnel passera d'abord par la création d'un type d'articles et d'un type de chemins, puis se clôturera par l'élimination de ce dernier.

ELIMINATION D'UN ITEM REPETITIF

Ce problème se pose en général quand la répétitivité est illimitée, lorsque le nombre maximum de valeurs est élevé par rapport à sa moyenne, ou dans le cas de bases de données relationnelles.

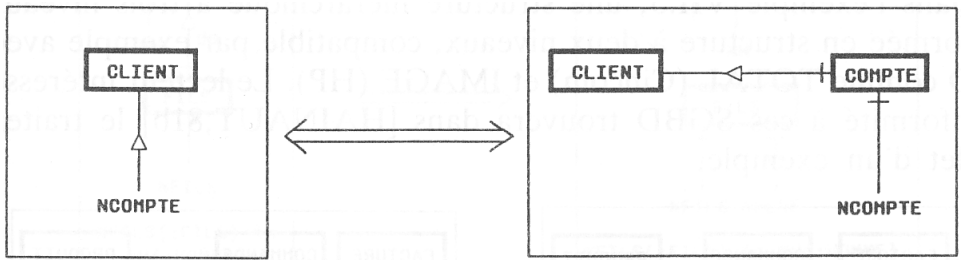


Figure 7.2 - Item répétitif

ELIMINATION D'UNE CLE D'ACCES NON IDENTIFIANTE

Le procédé permet également de respecter la contrainte "une seule clé par type d'articles".

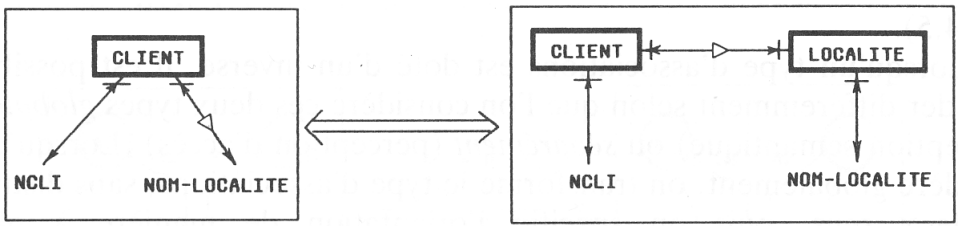


Figure 7.3 - Clés d'accès

ELIMINATION DE TYPES DE CHEMINS

L'application immédiate est évidemment la production de schémas relationnels ou COBOL. On peut aussi de cette manière obtenir un schéma en réseau à partir d'une spécification relationnelle.

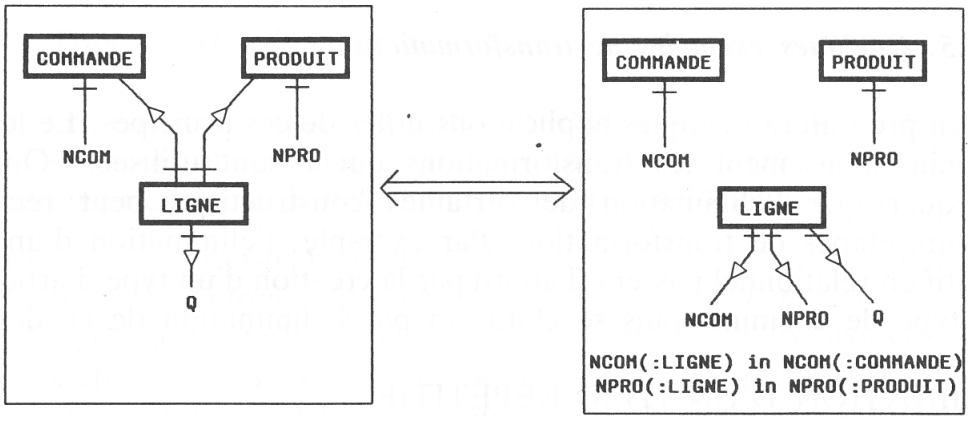


Figure 7.4 - Types de chemins

REDUCTION D'UNE HIERARCHIE

Dans l'exemple 7.5, une structure hiérarchique à trois niveaux est transformée en structure à deux niveaux, compatible par exemple avec des SGBD comme TOTAL (Cincom) et IMAGE (HP). Le lecteur intéressé par la conformité à ces SGBD trouvera dans (HAINAUT,81) le traitement complet d'un exemple.

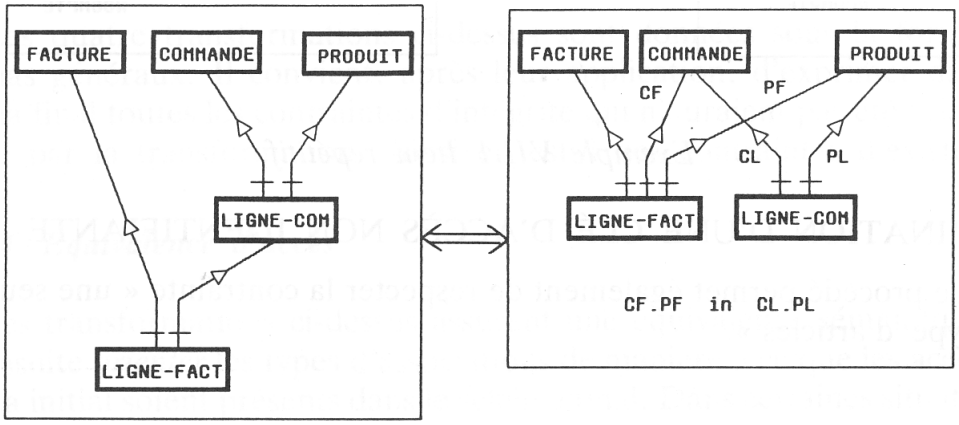


Figure 7.5 - Réduction d'une hiérarchie

ELIMINATION DE TYPES DE CHEMINS RECURSIFS

Ce problème se pose dans la plupart des SGBD. Les six schémas dérivés ci-dessous sont sémantiquement équivalents au premier et constituent les schémas les plus communément proposés. Ils sont ici obtenus systématiquement.

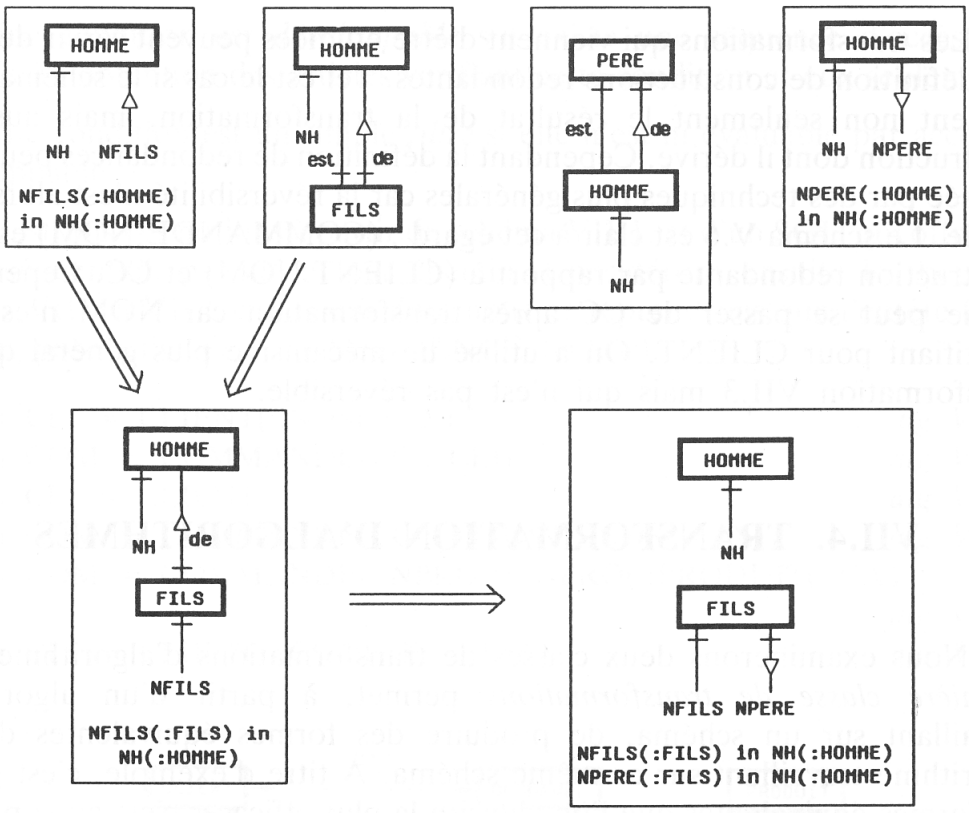


Figure 7.6 - Types de chemins récursifs

TRANSFORMATION DE TYPES DE CHEMINS INVERSES

Les principes évoqués en 7.3.4 sont appliqués ici aux deux types de chemins inverses STOCK du schéma 5.3. Dans l'exemple 7.7, le schéma de gauche est obtenu par une transformation globale et le schéma de droite par des transformations indépendantes. Dans ce dernier schéma, on a utilisé volontairement des techniques différentes. On notera que NPRO est devenu une clé d'accès à PRODUIT afin d'assurer l'équivalence sémantique. Ces principes seront également d'application pour traiter une clé d'accès, puisqu'il y correspond deux accès inverses (cfr schéma 5.5).

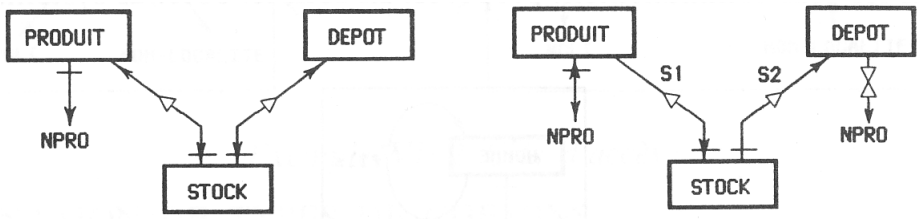


Figure 7.7 - Transformation de STOCK du schéma 5.3

7.3.6 TRANSFORMATIONS REDONDANTES

Les transformations qui viennent d'être étudiées peuvent servir de base à la définition de constructions redondantes. Tel est le cas si le schéma final contient non seulement le résultat de la transformation, mais aussi la construction dont il dérive. Cependant la définition de redondances peut être réalisée par des techniques plus générales car la réversibilité n'est alors plus exigée. Le schéma 5.6 est clair à cet égard : (COMMANDE,NOM) est une construction redondante par rapport à (CLIENT,NOM) et CC. Cependant on ne peut se passer de CC après transformation car NOM n'est pas identifiant pour CLIENT. On a utilisé un mécanisme plus général que la transformation 7.3 mais qui n'est pas réversible.

7.4 TRANSFORMATION D'ALGORITHMES

Nous examinerons deux classes de transformations d'algorithmes. La première classe de transformations permet, à partir d'un algorithme travaillant sur un schéma, de produire des formes équivalentes de cet algorithme travaillant sur ce même schéma. A titre d'exemple, c'est parmi ces formes équivalentes que l'on choisira la plus efficace; c'est aussi par ces transformations que l'on obtiendra un algorithme conforme à un SGD si l'algorithme initial ne l'était pas. La seconde classe de transformations permet, à partir d'un algorithme travaillant sur un schéma, de produire une forme équivalente de cet algorithme, travaillant sur un autre schéma, équivalent au premier. C'est, par exemple, par de telles transformations que l'on adaptera un algorithme suite à la transformation conforme à un SGD du schéma sur lequel il travaille.

Nous n'aborderons donc que les problèmes liés aux manipulations des données d'une base. Nous nous limiterons enfin à quelques transformations essentielles des conditions de sélection dans une boucle énumérative d'accès à des articles. Les autres cas s'en déduiront ou se traiteront d'une manière similaire. on se reportera en outre au paragraphe 6.3.10 qui propose quelques transformations algorithmes générales.

7.4.1 ALGORITHMES EQUIVALENTS SUR UN MEME SCHEMA

Une condition de sélection est en toute généralité une expression booléenne de conditions simples. Nous ne considérerons d'abord que les conditions formées d'une condition simple. Considérons un SGD, réel ou abstrait (voir 8.7), caractérisé notamment par les primitives d'accès qu'il offre. Relativement à ce SGD, on peut définir trois types de conditions simples.

- Une condition est évaluable par accès s'il existe une primitive du SGD qui fournit les articles vérifiant cette condition, et eux seulement.
- Une condition est développable s'il existe une séquence d'accès qui conduit à l'obtention des articles vérifiant cette condition et eux seulement.
- Une condition est un simple filtre si elle n'est ni évaluable par accès ni développable.

Illustrons ces notions à partir du schéma 7.1. et d'un SGD qui offre les accès séquentiel, par clé et par chemin. Considérons ensuite les boucles d'accès 7.1 à 7.6 ci-dessous dont nous ne reprendrons que l'entête (que l'on suppose suivie de : do SEQUENCE endfor).

for CLI := CLIENT(:NOM = X) alg 7.1
for COM := COMMANDE(CC : CLI) alg 7.2
for CLI := CLIENT alg 7.3
for COM := COMMANDE(CC : CLIENT(:NOM = X)) alg 7.4
for COM := COMMANDE(:NPRO = NPRO(:PRODUIT(:CAT = X))) alg 7.5
for CLI := CLIENT(:COMPTE < 1000) alg 7.6

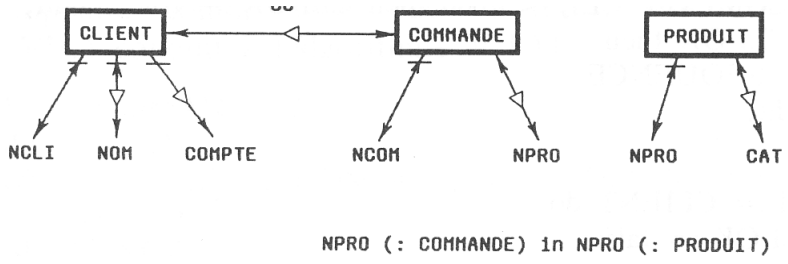


Figure 7.8 -

Aux algorithmes 7.1, 7.2 et 7.3 correspondent des conditions évaluables par accès (par clé pour 7.1, par chemin pour 7.2 et séquentiel pour 7.3).

Les algorithmes 7.4 et 7.5 contiennent des conditions développables (pour 7.4, on accèdera à CLIENT (:NOM = X) puis de là à COMMANDE via CC; pour 7.5, on accèdera à PRODUIT (:CAT = X), puis de là à COMMANDE par la clé NPRO).

Quant à la condition de l'algorithme 7.6, elle est un simple filtre.

Le but ultime des transformations qui vont être proposées est la production d'algorithmes exécutables par le SGD considéré, c'est-à-dire dont les boucles d'accès spécifient des conditions évaluables par accès.

TRANSFORMATION D'UNE BOUCLE A FILTRE

La condition est enlevée de l'entête de boucle et vérifiée explicitement. On obtient ainsi un accès séquentiel et une alternative pilotant explicitement les itérations. L'algorithme 7.6 est ainsi transformé dans l'un des algorithmes 7.7.

```
for CLI := CLIENT do
     if CLI(:COMPTE < 1000) then
          SEQUENCE
     endif;
endfor;

for CLI := CLIENT do
     if not CLI(:COMPTE < 1000) then next; endif;
     SEQUENCE
endfor;
```

alg 7.7

Cependant, afin de résoudre de manière uniforme les formes de conditions plus complexes, nous adopterons également les structures générales suivantes, qui isolent l'évaluation de la condition. Les algorithmes 7.8 illustrent cette forme. On y a également transformé la condition, qui porte à présent sur une valeur d'item et non plus sur un article.

```
for CLI := CLIENT do
     CLI-OK := false;
     if COMPTE(:CLI) < 1000 then CLI-OK := true; endif;
     if CLI-OK then
          SEQUENCE
     endif;
endfor;

for CLI := CLIENT do
     CLI-OK := false;
     if COMPTE(:CLI) < 1000 then CLI-OK := true; endif;
     if not CLI-OK then next; endif;
     SEQUENCE
endfor;
```

alg. 7.8.

TRANSFORMATION D'UNE BOUCLE EVALULABLE PAR ACCES

La condition étant évaluable, la boucle n'a pas en principe à être transformée. On peut cependant la transformer comme on l'aurait fait d'une simple boucle à filtre. En d'autres termes, on substitue un accès séquentiel à l'accès spécifié par la condition. Citons comme application le cas où un accès séquentiel serait moins coûteux qu'un accès par clé.

TRANSFORMATION D'UNE BOUCLE DEVELOPPABLE

Une condition développable spécifie explicitement ou non l'association à une collection d'objets. Le développement consiste à définir une boucle d'accès à cette collection, puis à y insérer la boucle que l'on transforme. La condition est remplacée par un accès à partir de chaque objet de la collection. L'algorithme 7.9 constitue le développement de l'algorithme 7.4.

```
for CLI := CLIENT(:NOM = X) do
     for COM := COMMANDE(CC : CLI) do
          SEQUENCE
     endfor;
endfor;
```

Alg 7.9.

De même, l'algorithme 7.10 est obtenu par développement de l'algorithme 7.5.

```
for P := PRODUIT(:CAT = X) do
     for COM := COMMANDE(:NPRO = NPRO(:P)) do
          SEQUENCE
     endfor;
endfor;
```

Alg 7.10

La nouvelle boucle ainsi définie devra à son tour être traitée selon la classe de la condition éventuelle qui la caractérise.

De même que pour les boucles évaluables par accès, on peut traiter une boucle développable comme une boucle à filtre. Traité de la sorte, l'algorithme 7.4 conduit à l'algorithme 7.11.


```
for COM := COMMANDE do
     COM-OK := false;
     if COM(CC : CLIENT(:NOM = X)) then COM-OK := true; endif;
     if COM-OK then
          SEQUENCE
     endif;
endfor;
```

Alg 7.11

TRANSFORMATION D'UNE BOUCLE A CONDITION COMPLEXE

Lorsque la condition est constituée de plusieurs conditions simples unies par "and", et que l'une d'entre elles au moins n'est pas un filtre, on décomposera cette condition en deux parties. L'une est évaluable par accès ou développable tandis que l'autre est (ou est considérée comme) un filtre. Chaque partie sera traitée selon les principes énoncés ci-dessus. S'il y a conflit (plus d'une condition évaluable par accès et/ou développable), on choisira une condition comme évaluable ou développable tandis que les autres seront traitées comme filtres. On en dérivera donc plusieurs algorithmes équivalents. Leurs coûts seraient cependant généralement différents, ce qui constitue un critère de choix (voir 8.4.4).

Lorsque la condition est constituée de conditions simples unies par des "or", elle est un filtre dès que l'une des conditions simples est un filtre. Si toutes les conditions simples portent sur la même clé d'accès, on définira une itération sur les valeurs spécifiées de cette clé. Une autre technique applicable lorsqu'aucune des conditions simples n'est un filtre, consiste à construire une liste de références sans doubles par des boucles d'accès indépendantes, basées chacune sur une condition simple. Le corps de la boucle initiale est alors exécuté pour chaque article référencé par la liste.

DEVELOPPEMENT DE LA CONDITION D'UNE ALTERNATIVE

Pour la plupart des SGD, la condition qui pilote une alternative ou une boucle WHILE ne peut qu'être très élémentaire, portant par exemple sur des valeurs de variables ou sur des valeurs d'items d'un article auquel on vient d'accéder (c'est-à-dire, en LDA, désigné par une variable d'article). Pour être évaluables, des conditions plus complexes, portant par exemple, sur des ensembles d'articles qualifiés, doivent être développées. Ce développement est analogue à celui des boucles d'accès. Considérons par exemple la condition de l'algorithme 7.11, qui est trop complexe pour être évaluable selon nos hypothèses. On proposera ci-dessous deux développements admissibles de cette condition. Le résultat de l'évaluation est placé dans le booléen CONDITION.

```
CONDITION := false;
```

```
for CLI := CLIENT(CC: COM) do
     if NOM(:CLI) = X then CONDITION := true; endif;
endfor;
```

Alg 7.12

```
CONDITION := false;
for CLI := CLIENT(:NOM = X) do
     for CCOM := COMMANDE(CC: CLI) do
          if CCOM = COM then CONDITION := true;
               exit CLI;
          endif;
     endfor;
endfor;
```

alg 7.13

PROBLEME DES ACCES MULTIPLES DANS UNE BOUCLE DEVELOPPEE

Considérons l'algorithme 7.14 qui accède aux CLIENT ayant passé au moins une COMMANDE dont NPRO = X.

```
for CLI := CLIENT(CC: COMMANDE(: NPRO = X)) do
     SEQUENCE
endfor;
```

Alg 7.14

Cet algorithme peut être développé selon l'algorithme 7.15

```
for COM := COMMANDE(: NPRO = X) do
     for CLI := CLIENT(CC: COM) do
          SEQUENCE
     endfor;
endfor;
```

Alg 7.15

En fait, ces algorithmes ne sont pas équivalents. Si par exemple, un CLIENT est associé à deux COMMANDES dont NPRO = X, ce CLIENT sera traité une seule fois selon l'algorithme 7.14 et deux fois selon l'algorithme 7.15. D'une manière générale, le problème des accès multiples (indésirables) peut surgir lorsque dans la chaîne des accès, il existe un type d'objets pour lequel on accède à plusieurs occurrences (ici COMMANDE) et duquel part un accès N-1 ou N-N (ici CC).

On résoudra ce problème en mémorisant dans une liste les références des objets auxquels on a déjà accédé. On peut alors traiter cette liste de deux manières :

Lors de l'accès à un objet, on teste l'appartenance de sa référence à la liste. Si elle s'y trouve, on ne traite pas l'objet; si elle ne s'y trouve pas, on l'y range et on traite l'objet.

Lors de l'accès à un objet, on range sa référence dans la liste. Lorsque la liste est complète, on en élimine les doubles puis on accède à nouveau aux objets restants afin de les traiter.

On peut alors proposer les algorithmes corrects 7.16 et 7.17.

```
newlist(LC);
for COM := COMMANDE(:NPRO = X) do
     for CLI := CLIENT(CC : COM) do
          if CLI not-in LC then
               addlist(LC,CLI);
               SEQUENCE
          endif;
     endfor;
endfor;
```

Alg 7.16

```
newlist(LC);
for COM := COMMANDE(:NPRO = X) do
     for CLI := CLIENT(CC : COM) do
          addlist(LC, CLI);
     endfor;
endfor;
unique(LC);
for CLI := LC do
     SEQUENCE
endfor;
```

Alg 7.17

7.4.2 ALGORITHMES EQUIVALENTS SUR SCHEMAS EQUIVA-

LENTS

Cette seconde classe de transformations permet de "faire suivre" un algorithme lorsque le schéma sur lequel il travaille subit des transformations comme celles qui sont proposées en 7.3. Nous définirons les transformations pour quelques formes syntaxiques LDA selon les principales transformations de schémas. Toutes ces transformations sont réversibles.

TRANSFORMATIONS SUITE A LA CREATION D'UN TYPE D'ARTICLES (Transf 7.1. et 7.2)

Désignons par CB une condition quelconque (éventuellement vide) portant sur B, et CA une condition portant sur A. De l'équivalence entre R d'une part et RAB, RA, RB d'autre part, on établit :

1. que la forme

```
for a := A (R: B (CB)) do
```

est équivalente à

```
for a := A (RA: RAB (RB: B(CB))) do
```

2. et que la forme

```
for a := A (CA) do
   for b := B ((R: a) and CB) do
```

est équivalente à

```
for a := A (CA) do
   for b := B ((RB: RAB (RA: a) do
```

qui peut se développer en

```
for a := A (CA) do
  for ab := RAB (RA: a) do
     for b := B ((RB: ab) and CB) do
```

TRANSFORMATIONS SUITE A LA ROTATION SIMPLE D'UN TYPE D'ASSOCIATIONS (Transf 7.3)

Désignons par CB une condition quelconque portant sur B, et CC une condition portant sur C. De l'équivalence entre R, S d'une part et R,T d'autre part, on établit :

3. que la forme `for b := B (S: C (CC)) do`

est équivalente en toute généralité à

```
for b := B (R: A (T: C (CC))) do
```

ou, si A est un item, à

```
for b := B (R: A in A (T: C (CC))) do
```

4. et que la forme

```
for b := B (CB) do
  for c := C ((S: b) and CC) do
```

est équivalente en toute généralité à

```
for b := B (CB) do
  for c := C ((T: A (R: b)) and CC) do
```

qui peut se développer comme suit

```
for b := B (CB) do
  for a := A(R: b) do
     for c := C ((T: a) and CC) do
```

Si A est un item, on peut écrire

```
for b := B (CB) do
  for c := C ((T: A = A (R: b)) and CC) do
```

TRANSFORMATIONS SUITE à LA ROTATION MULTIPLE D'UN TYPE D'ASSOCIATIONS (Transf. 7.4)

CB et CC étant définis comme ci-dessus, de l'équivalence entre S, R1, R2 d'une part, et T1, T2, R1, R2 d'autre part, on établit :

5. que la forme

```
for b := B (CB) do
   for c := C (S: b) do
```

est équivalente à

```
for b := B (CB) do
  for c := C ((T1: A1 (R1: b)) and (T2: A2 (R2: b))) do
```

qui n'est généralement pas évaluable par accès mais qui par contre pourrait être développée.

6. La forme "for c := C (S: B (CB)) do" est développable dans la forme ci-dessus. Par contre elle ne peut être transformée, sauf si CB est identifiante, dans l'algorithme suivant :

```
for c := C((T1 : A1(R1 : B(CB))) and (T2 : A2(R2 : B(CB)))) do
```

7. On établit également

que la forme

```
for c := C(CC) do
   for b := B(S : c) do
```

est équivalente à

```
for c := C (CC) do
  for b := B ((R1: A1 (T1: c)) and (R2: A2 (T2: c))) do
```

qui n'est généralement pas évaluable par accès mais qui pourrait être développée.

8. La forme "for b := B (S: C (CC) do" est développable dans la forme ci- dessus. Par contre elle ne peut être transformée, sauf si CC est identifiante), de la manière suivante :

```
for b := B ((R1: A1 (T1: C (CC))) and (R2: A2 (T2: C (CC)))) do
```

QUELQUES EXEMPLES

1. A partir de la transformation de schémas de l'exemple 7.2, l'expression,


```
for CLI := CLIENT (: NOM-LOCALITE = X) do
```

est transformée en

```
for CLI := CLIENT (: LOCALITE (: NOM-LOCALITE = X)) do
```

qui est développable en

```
for LOC := LOCALITE (: NON-LOCALITE = X)) do
  for CLI := CLIENT (: LOC) do
```

2. A partir de la transformation de schémas de l'exemple 7.3, l'expression

```
for L := LIGNE (: COMMANDE (: NCOM = X)) do
```

sur le schéma de gauche, est transformée, sur le schéma de droite, en

```
for L := LIGNE (: NCOM = NCOM (: COMMANDE (: NCOM = X))) do
```

puis, par simplification selon la règle 5.3.10, en

```
for L := LIGNE (: NCOM = X) do
```

Remarquons que cette simplification n'est valide que grâce à la première contrainte d'intégrité du schéma de droite.

7.5 TRADUCTION D'UN ALGORITHME LDA EN PROGRAMME

Il s'agit de la phase finale de production concernant les traitements. Chaque algorithme ADL, attaché à un module, est traduit en une unité de programme exécutable, rédigée dans un langage de programmation et dans un langage de manipulation de données (LMD), lorsque les deux sont distincts. Nous intéressant essentiellement aux problèmes liés aux données, nous n'aborderons les points de programmation que dans cet esprit.

Nous ne parlerons pas en particulier de la traduction des constructions classiques, qui ne posent guère de problèmes dans les langages de programmation habituels, et dont des formes primitives sont suggérées en 6.3.10.

Si l'on part d'un algorithme LDA conforme au SGD, la traduction sera assez systématique. En ce qui concerne la manipulation des données, nous proposerons en 8.7. un mécanisme général de réalisation sous la forme de module de gestion de données). Outre ce problème, il faudra résoudre l'expression des concepts comme ceux de variable d'article et de variable liste.

Une variable d'article contient l'identifiant d'un article (ou une valeur représentant son absence). Cet identifiant sera fonction du SGD. On choisira un stockage explicite de la database-key en CODASYL 71/73, ou une KEEPLIST en CODASYL 78. En séquentiel indexé COBOL, on pourra adopter la valeur de la clé principale et dans un fichier séquentiel, le numéro d'ordre de l'article.

Selon la taille prévue d'une variable liste, on représente celle-ci par un tableau ou par un fichier. Dans ce dernier cas, on choisira un fichier séquentiel indexé ou séquentiel, selon qu'il est ou non prévu d'y accéder aléatoirement (test d'appartenance par exemple). Certains SGBD offrent des structures dynamiques bien adaptées à cette notion, telles les tables temporaires dans les SGBD relationnels, les DYNAMIC SETS en CODASYL 71/73 ou les KEEP-LISTs en CODASYL 78 et suivants.

Chapitre 8 : DEMARCHE DE CONCEPTION D'UNE BASE DE DONNEES

8.1 INTRODUCTION

La démarche qui va être développée consiste en un ensemble de processus de transformation. A partir d'une description (d'une partie) de la solution, chaque processus produit une nouvelle description, équivalente à la précédente, et qui intègre des choix faits par rapport à un critère de décision. Chaque processus est caractérisé par un critère, ou, en toute généralité, par un petit nombre de critères homogénes. Ces critères sont, autant que possible indépendants les uns des autres, ce qui rend les processus correspondants individualisables. Des critères comme : accès strictement nécessaires et suffisants, efficacité des accès logiques, conformité d'un schéma à un SGD, conformité d'un algorithme à un SGD, parallélisme maximum, peuvent souvent être envisagés indépendamment les uns des autres. Ces processus, par nature hiérarchisés, ne constituent pas à proprement parler une méthode de conception, mais plutôt un cadre de référence pour définir une méthode, pour analyser une méthode existante ou encore pour construire progressivement un atelier logiciel modulaire.

Toute méthode complète devra, explicitement ou non, mettre en oeuvre les processus évoqués. Ils pourront cependant être regroupés ou simplifiés au point de ne plus réclamer de décision. D'autre part, certaines méthodes s'arrêteront à un niveau élevé, le relais étant pris alors par des outils opérationnels tels que les langages de 4ème génération. D'autres encore démarreront à un niveau intermédiaire, à partir de descriptions élaborées par ailleurs.

Telle qu'elle est proposée, la démarche offre une modularité qui lui permet de s'adapter à des contingences particulières : choix d'un autre modèle conceptuel, indépendance par rapport au SGD, indépendance par rapport aux langages de programmation, abandon de certains processus estimés inutiles, ajout de processus spécifiques non encore mentionnés, etc. A ces propriétés de la modularité, déjà bien connues dans le domaine du génie logiciel s'ajoute celle, tout aussi connue dans le domaine méthodologique, de la parcellisation et donc de la maîtrise de la complexité. .J0

8.2 ORGANISATION GENERALE DE LA DEMARCHE

La démarche de conception est décomposée en niveaux s'inspirant directement des niveaux de description d'une base de données qui ont été discutés au chapitre 4 (paragraphe 4.6, sché-

ma 4.2). Elle consiste en trois parties consécutives : l'analyse conceptuelle (développée dans (BOD-PIGN,83)), la conception logique et la conception physique (voir schéma 8.1). Ces deux dernières parties constituent la phase de mise-en-oeuvre du Système d'Information et plus particulièrement de sa base de données; elles sont l'objet essentiel du présent volume. Cette phase de mise-en-oeuvre a été conçue sur base des objectifs suivants :

- Produire de façon systématique une solution (dont la base de données) qui respecte la spécification conceptuelle et qui soit efficace vis-à-vis des applications prévues.
- Localiser et réduire les dépendances vis-à-vis du SGD et des langages de programmation, en particulier en retardant le plus possible les décisions concernant ces composants; il en résulte une minimisation de l'effort à consentir lors d'un changement de SGD ou des caractéristiques physiques de la base de données.
- Permettre la conception de bases de données gérées tant par des logiciels spécialisés (SGBD) que par de simples systèmes de gestion de fichiers (SGF).
- Servir de base à une programmation orientée bases de données.

La décomposition de la phase de mise-en-oeuvre en ses deux parties de conception logique et de conception physique dérive immédiatement de ces objectifs. La conception logique, en effet, produit une solution technique, efficace, mais indépendante des outils de réalisation du système d'information opérationnel. La conception physique est la traduction de cette solution sur une machine réelle caractérisée par ses composants matériels et logiciels (système d'exploitation, langages de programmation, systèmes de gestion de données, gestionnaires d'écran, de communication, de processus, etc.).

8.3 L'ANALYSE CONCEPTUELLE

L'analyse conceptuelle conduit à élaborer une description complète du Système d'Information qui soit indépendante de la notion même d'outil informatique. Cette description constitue la spécification de la solution retenue. Elle est constituée d'un schéma conceptuel des données, d'un schéma conceptuel des traitements ainsi que du relevé des quantifications. Ces schémas traduisent une perception de la solution au travers de modèles appropriés. Modèles et analyse conceptuels sont développés dans (BOD-PIGN,83) et résumés dans le chapitre 3 du présent volume.

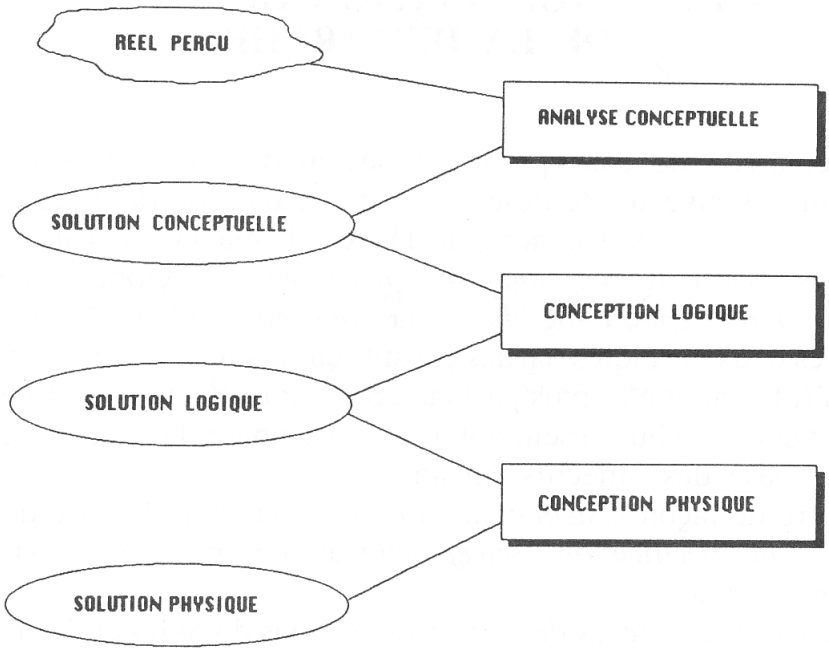


Figure 8.1 - Schéma général de la démarche

8.4 LA CONCEPTION LOGIQUE

8.4.1 OBJECTIFS ET SCHEMA GENERAL

Cette phase conduit, à partir de la solution conceptuelle, à la définition d'une solution qui soit "exécutable" par une machine abstraite, strictement indépendante des machines réelles. La solution logique est est dotée des trois propriétés suivantes : elle est correcte, efficace et indépendante de la machine réelle.

La solution logique est correcte : les algorithmes satisfont à la spécification de leur module et le schéma des données exprime toute la sémantique (y compris les contraintes d'intégrité) du schéma conceptuel.

La solution logique est efficace : l'accès aux données que réalisent les algorithmes est optimisé; le schéma de données ne reprend que les mécanismes d'accès strictement nécessaires aux algorithmes des modules.

La solution logique est indépendante de la machine réelle. La machine abstraite à laquelle s'adresse la solution est constituée d'un SGD virtuel du type MAG et d'un processeur LDA.

La solution s'exprime sous la forme d'un schéma MAG et par une architecture de modules (développée dans le 3ème volume de l'ouvrage) dotés chacun d'un algorithme LDA.

La conception logique est décomposée en quatre phases, dont l'organisation est mise en évidence dans le schéma 8.2.

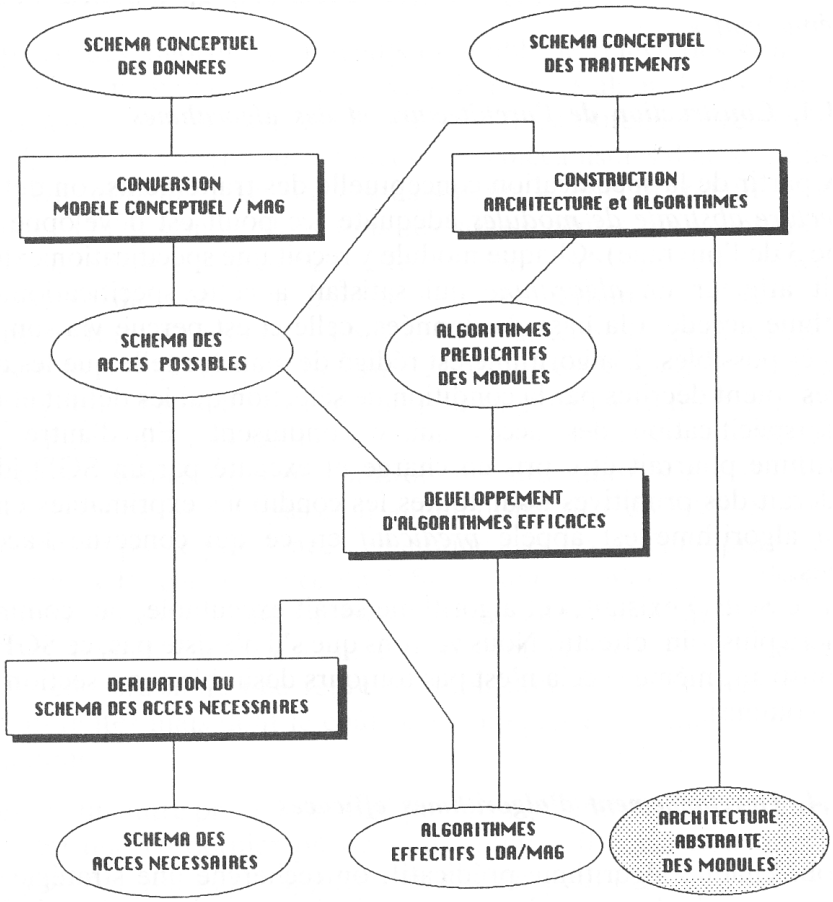


Figure 8.2 - Phases de la conception logique

8.4.2 PRODUCTION D'UN PREMIER SCHEMA D'ACCES (SCHEMA DES ACCES POSSIBLES)

Le Schéma des Accès Possibles (SAP) exprime dans le MAG une structure de données qui est dotée d'une seule propriété : elle est une représentation correcte aussi directe que possible du schéma conceptuel. Il ne contient en particulier aucun aspect lié aux différentes sortes de

performances (sinon la complétude et la non-redondance sémantiques). Son objectif est d'offrir une expression des données dans un formalisme adapté aux processus de conception logique.

La dérivation du SAP à partir du schéma conceptuel s'effectue systématiquement selon les règles définies dans la section 7.2. A partir du relevé des quantifications conceptuelles, on élabore la description statistique du Schéma des Accès Possibles tant du point de vue des nombres d'éléments que de leur évolution (taux de croissance). Il est clair que la prise en charge de schémas conceptuels exprimés dans un autre formalisme que le modèle E/A réclame simplement un autre jeu de règles de dérivation.

La notion de SAP porte, dans (ROBINSON,81) le nom évocateur de "Schéma naïf".

8.4.3 CONSTRUCTION DE L'ARCHITECTURE ET DES ALGORITHMES

A partir de la spécification conceptuelle des traitements, on définit une architecture abstraite de modules adéquate (ce point est développé dans le volume 3 de l'ouvrage). Chaque module y reçoit une spécification externe, et se voit affecter un algorithme qui satisfait à cette spécification. Si cet algorithme accède à la base de données, celle-ci est perçue via son schéma des accès possibles. L'algorithme est rédigé de manière telle que les données utilisées soient décrites par la condition de sélection qui les définit et non pas par la spécification des accès qui y conduisent. En d'autre termes, l'algorithme pourrait être pris en charge et exécuté par un SGD idéal qui possèderait des primitives pour tous les filtres exprimables en LDA. Un tel algorithme est appelé prédicatif en ce qui concerne l'accès aux données.

Si ce SGBD existait, cet algorithme serait exécutable, ou, comme on le désignera plus loin, effectif. Nous verrons que s'il n'existe pas, ce SGBD peut être construit, même si cela n'est pas toujours désirable (voir section 8.7. de ce volume).

8.4.4 DEVELOPPEMENT D'ALGORITHMES EFFICACES

Pour chaque algorithme prédicatif, on recherche une stratégie d'accès qui soit optimale du point de vue (principalement) des accès à la base de données. Dans cette recherche, on suppose que tout type de chemins est doté d'un inverse, et que toute combinaison d'items est une clé d'accès potentielle.

Le seul critère de performance envisageable à ce stade est le nombre d'opérations logiques (accès à des articles). Il n'est pas suffisant pour affirmer avec certitude qu'un algorithme sera, lors de son exécution, meilleur qu'un autre. Les décisions prises ici devront être réévaluées lorsque les caractéristiques physiques des données seront précisées (elles constituent le schéma interne); c'est en effet à ce moment seulement qu'une évaluation précise, en termes d'accès physiques, pourra être réalisée.

En toute généralité, cette phase peut être décomposée en trois processus. Le premier est la production d'algorithmes évaluables par accès à partir de l'algorithme prédicatif. Ce concept

ainsi que les règles de production ont été étudiés dans la section 7.4.1 (algorithmes équivalents sur un même schéma). Le deuxième est l'évaluation des coûts probables de chacun des algorithmes retenus; les résultats pourront s'exprimer comme décrit en 8.4.5. Le troisième est le choix du meilleur algorithme, sur base de critères dont l'un des principaux est le coût probable minimum. Cet algorithme choisi (on peut d'ailleurs en retenir plus d'un) sera appelé l'algorithme effectif du module.

8.4.5 DERIVATION DU SCHEMA DES ACCES NECESSAIRES.

A partir des algorithmes effectifs on établit le relevé des structures de données strictement nécessaires à l'exécution de ces algorithmes. On obtient ainsi le schéma des accès nécessaires (SAN). Il est obtenu en relevant, pour chaque algorithme effectif, les données et les mécanismes d'accès utilisés par celui-ci. On obtient ainsi un "sous-schéma" effectif relatif à chaque algorithme. On fusionne alors tous les sous-schémas ainsi obtenus, pour produire le schéma "global" effectif. Le SAN est en principe un sous-ensemble du SAP. On pourra admettre certaines structures redondantes permettant de diminuer les accès aux articles. Il sera prudent de ne pas abuser de ce type d'optimisation à ce niveau et de les reporter plutôt au niveau physique, car son efficacité ne pourra généralement être confirmée, voire infirmée, que lors de l'utilisation des particularités du SGD réel.

Certaines données peuvent n'avoir pas été utilisées par les fonctions analysées. Elles ne doivent pas pour autant disparaître du SAN. Si tel était le cas, il n'y aurait plus équivalence sémantique avec les schémas précédents, en particulier avec le schéma conceptuel. Si une donnée n'est pas utilisée, il faut soit la retirer de tous les schémas en amont (on admet alors qu'il y a eu erreur de perception ou de contexte au niveau conceptuel), soit la conserver dans le SAN. On choisira alors une représentation dont les performances d'accès sont indifférentes mais dont le coût de maintenance (création, suppression) est minimum.

Il existe un autre procédé analytique, plus général que la construction du SAN, et tel que ce dernier puisse s'en déduire. Il consiste à relever, par module, puis par phase, par application, et enfin pour le projet, les primitives de manipulation de données utilisées par les algorithmes correspondants. En ce qui concerne l'accès, une primitive sera simplement décrite par l'expression de la séquence des articles. La base de données est alors un type de données qui est manipulable par le jeu de primitives ainsi constitué. Cette définition est proche de celle d'un type abstrait. Le SAN est construit comme l'ensemble des structures utilisées par ces primitives, enrichi des éléments non utilisés, comme discuté ci-dessus.

Ce schéma étant élaboré, on en définira la quantification statique (taille des populations d'objets, etc), issue directement de la description statistique du SAP (voir 8.4.2.). On définira aussi sa quantification dynamique, qui correspond à la consolidation, pour tous les algorithmes effectifs d'une phase et d'une application, de leur coût probable (voir 8.4.4.). On procédera par exemple de la manière suivante.

A la liste des primitives de l'algorithme d'un module, on associera :

- le nombre moyen (éventuellement minimum et maximum) d'activations de la primitive lors d'une activation du module; - le nombre moyen d'éléments (articles le plus généralement) concernés par une activation de la primitive; on indiquera par exemple le nombre moyen d'articles de la séquence caractéristique d'une primitive d'accès; - le produit de ces deux valeurs, représentant le nombre moyen d'éléments (articles) concernés par une activation du module.

  On établit la fréquence d'activation du module, sur base d'une estimation qui peut être réalisée au niveau conceptuel. Ces fréquences peuvent aussi provenir de résultats de simulation réalisés au niveau conceptuel (voir volume 1), traduits pour l'architecture des modules qui aura été retenue. On peut alors calculer, pour le module :
- le nombre moyen, par unité de temps (par jour par exemple), d'activations de chaque primitive;
- le nombre moyen d'éléments (articles) traités par unité de temps.

On calcule alors ces deux dernières grandeurs pour l'ensemble des modules, ce qui constitue la quantification dynamique du SAN.

En ne considérant que les accès on établit une carte du "trafic" dans le schéma montrant clairement les sections critiques où l'effort d'optimisation se concentrera.

Il faut être conscient que ce processus conduit à un schéma de base de données qui est lié aux programmes qui utilisent cette base. Ce résultat peut sembler contradictoire avec l'approche "Base de Données" dont la présente démarche se réclame. Ce résultat est cependant inévitable à ce niveau si l'on désire optimiser les performances globales. Les inconvénients généralement admis de cette dépendance (notamment rigidité vis-à-vis des besoins ultérieurs) ont une importance qui est fonction du SGD - les SGBD relationnels sont très souples sur ce plan -, du schéma de la base de données et des techniques de programmation utilisées (voir 8.5.9. et 8.7). Ajoutons encore que l'on assurera généralement une plus grande stabilité des structures d'accès en mettant en oeuvre des mécanismes d'accès non utilisés par les algorithmes actuels, mais qui seront nécessaires dans l'avenir; on enrichit donc le SAN à titre prévisionnel.

8.5 LA CONCEPTION PHYSIQUE

8.5.1 OBJECTIFS ET SCHEMA GENERAL

Le résultat de la phase de conception logique consiste principalement en un schéma des accès nécessaires, accompagné de sa quantification statique et dynamique, d'une architecture abstraite de modules et des algorithmes effectifs des modules.

Ceci va servir de spécification pour la phase de conception physique qui est chargée de produire une solution physique qui soit correcte, efficace et exécutable par une machine réelle.

La solution physique est correcte : les programmes et procédures traduisent précisément les algorithmes effectifs et respectent leur spécification. L'architecture concrète de ces program-

mes est une représentation correcte de l'architecture abstraite. La structure de la base de données et/ou des fichiers exprime toute la sémantique et les mécanismes d'accès du schéma des accès nécessaires.

La solution physique est efficace : l'efficacité logique doit être réalisée par le respect des algorithmes effectifs et des structures d'accès proposés au niveau logique. D'autre part des optimisations spécifiques à une machine réelle seront recherchées.

La solution physique est exécutable : elle s'exprime par des programmes et procédures rédigés dans des langages pour lesquels la machine réelle dispose de processeurs (langages de programmation traditionnels, langage de manipulation de données d'un SGBD, modules de gestion de données, langage de 4ème génération ou de développement d'application, etc.), ainsi que par des schémas de données interprétables par des logiciels de gestion de données (SGBD ou SGF). La solution exécutable fera aussi appel à des composants tels que des gestionnaires d'écran, des systèmes de communication entre machines (accès réseau par exemple). Nous ne les envisageons pas dans ce volume.

La conception logique est décomposée en phases dont les principales sont présentées dans le schéma 8.3.

On observe que la démarche distingue deux sous-couches physiques. L'une, dite abstraite, spécifie une solution conforme à la machine réelle, mais encore exprimée en termes du MAG et du LDA. La seconde, dite concrète, conduit à la construction de la solution exécutable à partir de cette spécification. Cette découpe, qui peut apparaître comme d'un raffinement excessif, est importante sur deux plans. D'une part, la définition de la solution conforme abstraite utilise les modèles et les outils généraux de transformation du niveau supérieur, ce qui entraîne une économie de concepts, de moyens, et une plus grande facilité d'automatisation. D'autre part, cette étape intermédiaire correspond à un niveau de programmation souvent utilisé en pratique (voir programmes du niveau 2 dans la section 8.7).

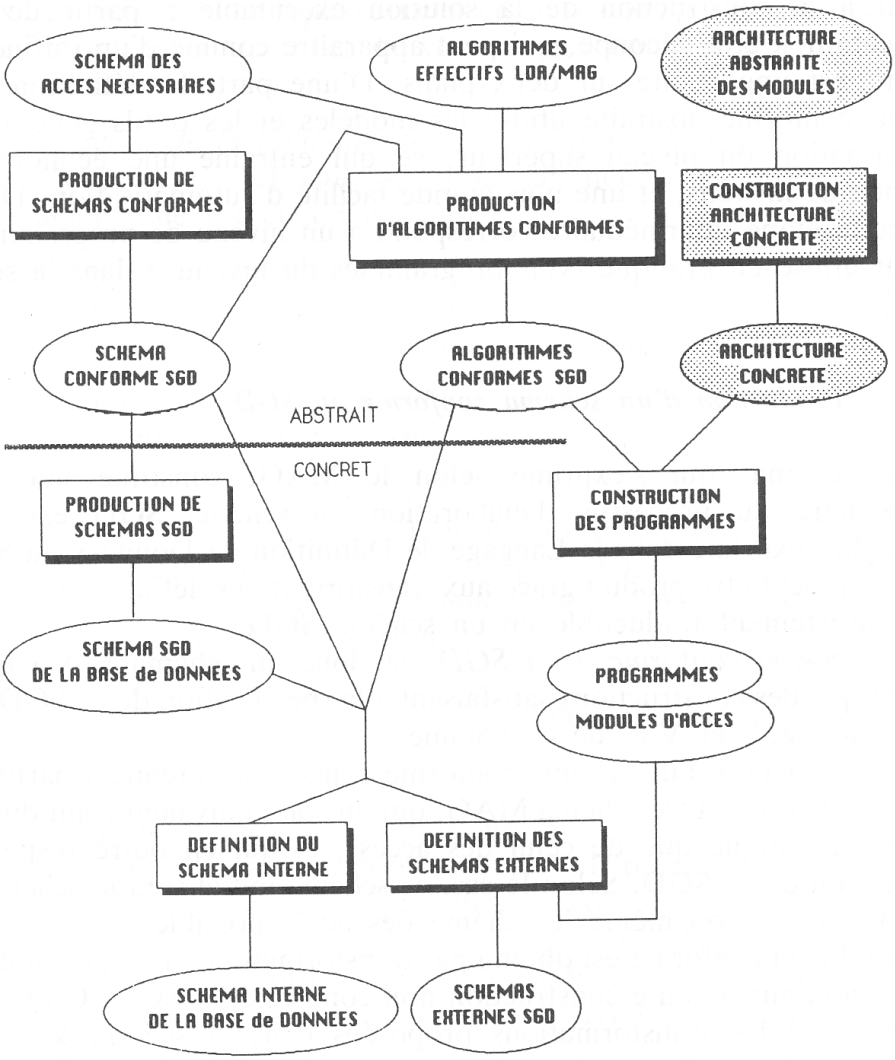


Figure 8.3 - Phases de la conception physique

8.5.2 PRODUCTION D'UN SCHEMA CONFORME AU SGD

Ce schéma, qui s'exprime selon le MAG, constitue une étape intermédiaire du processus d'élaboration du schéma SGD exécutable (c'est-à-dire exprimé dans le Langage de Définition de Données du SGD). Ce schéma peut être produit grâce aux transformations définies sur le MAG et est directement traductible en un schéma SGD.

Un schéma conforme à un SGD est donc un schéma MAG qui ne contient que des constructions satisfaisant à la spécification de ce SGD (voir les sections 5.7. et 5.8. de ce volume).

La production d'un schéma conforme consiste à obtenir, à partir d'un schéma MAG, un autre schéma MAG, qui lui soit équivalent (tant du point de vue sémantique que de celui des accès), et qui en outre respecte la spécification de ce SGD. Selon le cas, ce schéma initial sera le schéma des accès nécessaires ou même le schéma des accès possibles.

Le schéma conforme est obtenu par transformation du schéma initial de manière à éliminer toute construction non conforme au SGD. On utilisera pour ce faire les transformations proposées dans la section 7.3. qui garantissent l'équivalence sémantique et d'accès.

D'autres transformations, non indispensables sur le plan formel (respect des contraintes du SBD), pourront être adoptées de manière à satisfaire des critères tels que les suivants :

- respect de contraintes plus strictes que celles du SGD; ex : pas plus de 3 niveaux en IMS, pas plus d'une clé d'accès par fichier COBOL, etc;
- minimisation des temps d'accès, de mise-à-jour; ex : redondances;
- modularisation des ensembles de données; ex : éclatement d'un type d'articles en deux pour diminuer le volume des données en ligne;
- maximisation du parallélisme; ex : éclatement d'un type d'articles en deux pour permettre à deux modules exclusifs de travailler simultanément;
- optimisation du plan de réorganisation des données; ex : éclatement d'un type d'articles en deux selon le taux de mise-à-jour des items pour diminuer le volume des données à réorganisation fréquente;
- optimisation de la politique de reprise sur incident; ex : éclatement ... (cfr ci-dessus) pour diminuer le volume des données réclamant une protection coûteuse;
- redondances de sécurité; ex : pour un item copié, l'une des valeurs peut être détruite accidentellement sans perte définitive de l'intégrité des données;
- confidentialité; ex : éclatement d'un type d'articles selon le degré de confidentialité des items.

C'est aussi à ce niveau que l'on introduire, lorsque le choix est possible, une première découpe des structures de données en fichiers. Le choix des fichiers est lié à des considérations relatives, par exemple :

- à l'homogénéité des opérations de mise-à-jour,
- au parallélisme,
- à la modularisation des ensembles de données,
- à la réorganisation,
- à la reprise sur incident,
- aux performances des applications.

Ces choix sont liés aux propriétés que le SGD reconnait au fichier. Dans de nombreux cas, en effet, les principes de concurrence, de montage en ligne, de réorganisation, de reprise sur incident (journalisation), de copie, de verrouillage, de paramètres physiques statiques (taille de

pages) et dynamiques (taille de tampons, gestion de ceux-ci), etc. sont directement liés à la notion de fichier.

On fera remarquer que cette phase n'est, en toute généralité, pas entièrement automatisable. Il existe en effet, pour toute construction non conforme, plusieurs constructions équivalentes conformes. Le choix de celle qui sera retenue dépend de critères parfois quantitatifs (volume, temps d'accès) mais parfois aussi, comme cela apparait clairement ci-dessus, qualitatifs (parallélisme, modularisation, etc.).

8.5.3 PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES AU SGD

Ce processus est, dans ses objectifs, parallèle au précédent. Il consiste à donner, des algorithmes opérant sur le schéma non conforme, des versions LDA équivalentes, mais qui soient conformes au SGD et qui opèrent sur le schéma conforme. La notion d'algorithme conforme a été définie dans la section 6.6. Cette conformité est obtenue en partie par les transformations présentées en 7.4.2., qui sont induites formellement par les transformations de schémas. La conformité sera totale si en outre l'algorithme respecte le troisième principe de la définition de conformité (enchaînement conforme des primitives, section 6.6.).

C'est aussi à ce niveau que l'on définira les principes de régulation de la concurrence et de protection contre les incidents en fonction des possibilités du SGBD. Ces principes sont à définir en fonction des objectifs à atteindre en cette matière. Ils se répercutent non seulement dans les algorithmes mais aussi dans l'architecture concrète des modules et dans la structure de données (cfr 8.5.2.). Ces principes trouveront leur traduction dans les programmes (8.5.6.) et dans le schéma interne (8.5.7.). On les retrouvera également lors de l'établissement du plan d'exploitation (8.5.9.).

8.5.4 PRODUCTION DU SCHEMA SGD

Cette phase consiste à produire, à partir d'un schéma MAG conforme au SGD, un texte décrivant ces structures de données rédigé dans le Langage de Description de Données de ce SGD. Cette production est pour l'essentiel systématique, donc automatisable. Cependant certaines définitions devront être élaborées manuellement, du moins si l'on désire exploiter les possibilités d'optimisation du SGD, et ce pour deux raisons principales. D'abord, à un concept du MAG peuvent correspondre dans le SGD plusieurs constructions distinctes, dont la meilleure dans chaque cas ne peut pas être déterminée aisément de manière automatique. En outre, certains SGD possèdent des concepts qui doivent être spécifiés dans un schéma, mais dont l'expression n'est pas prise en charge par le MAG (ce modèle ne couvre pas toutes les constructions de tous les SGD; cfr section 5.1.). Si par contre, l'on accepte des solutions non nécessairement optimales, on peut fixer par défaut ces choix particuliers, et ainsi rendre ce processus entièrement automatisable.

On notera que, souvent, le schéma SGD ainsi produit est encore incomplet en ce qui concerne le SGD. Ceci découle de ce que la notion de niveaux de perception des données (voir chapitre

4) ne correspond pas exactement aux différents schémas d'un SGD, la mise sur le marché de ce dernier étant souvent antérieure à la mise en évidence de cette notion de niveaux. En particulier, un schéma CODASYL 71/73 exprime, dans un même texte, des notions conceptuelles (notion d'identifiant), d'accès logiques (clé d'accès, chemins d'accès), de spécifications internes (pointeurs arrière) et de modalités d'usage, typiques d'un schéma externe (LOCATION MODE et SET SELECTION)

8.5.5 DEFINITION DES SCHEMAS EXTERNES

Un schéma externe est la perception des données que l'on offre à une classe homogène d'utilisateurs (voir chapitre 4). Ce schéma ne reprend généralement que les types de données nécessaires et suffisants pour cette classe, et selon un format adapté aux besoins de celle-ci.

Dans le cadre de cette démarche une classe d'utilisateurs se traduit notamment par un ensemble de modules constitué selon des critères qui pourraient être par exemple l'homogénéité temporelle ou l'homogénéité spatiale. Dans cette optique, l'ensemble des modules d'une phase pourrait constituer un critère réaliste.

Le relevé des types de données décrits dans un schéma externe se fera par intégration progressive des "sous-schémas" effectifs conformes des algorithmes des modules concernés, selon le même processus qu'en 8.4.5. Ce schéma externe MAG doit alors être traduit en forme exécutable. Cette forme dépend essentiellement du SGD et des techniques de programmation utilisées. Elle consistera en un "subschema" en CODASYL, en "view" dans certains SGBD relationnels, en un PSB en IMS, ou encore en un module de gestion de données, ou modules d'accès, comme on le suggérera en 8.7. Elle spécifiera, si nécessaire, les fichiers à monter en ligne.

Nous ferons deux remarques générales tendant à nuancer l'apparente simplicité du processus. D'abord, la forte liaison entre programme et schéma externe qui existe dans certains SGD peut se traduire par le fait que la frontière entre description des données et algorithmique devient très imprécise. Citons deux exemples : les clauses LOCATION MODE et SET SELECTION en CODASYL 71/73 constituent une ingérence de procédures algorithmiques dans une description de données. La définition d'une "VIEW" relationnelle sur base de jointures, projections et sélections en est un autre exemple, ces opérations pouvant être spécifiées soit dans l'algorithme soit dans la définition de la "VIEW".

Une deuxième remarque importante est que les types de données explicitement cités dans l'algorithme d'un module sont certes nécessaires, mais pas toujours suffisants. Par exemple, la suppression d'un article CLIENT concerne également les types COMMANDE ET LIGNE, dont des articles seront immanquablement supprimés. Ces extensions "cachées" sont liées soit à des contraintes d'intégrité, soit à une disposition physique particulière des données (chaînes d'articles de plusieurs types, dont un seul est exploité, ou traversant "inopinément" un fichier dont on n'a pas besoin).

L'utilisation de modules de gestion de données permet d'éviter ces problèmes, ou en tout cas de las pallier en en cachant les effets d'une manière centralisée (voir 8.7.).

8.5.6 CONSTRUCTION DES PROGRAMMES

Cette phase doit être considérée comme un processus de codage systématique, toute activité de programmation proprement dite étant terminée à ce stade. Pour être systématique, et donc automatisable, ce processus n'est cependant pas exempt de choix, en particulier en ce qui concerne l'accès aux fichiers et bases de données. La localisation de ces accès et la perception des données qui sera offerte au programmeur (qu'il soit humain ou automate) constituent les choix principaux. Les accès peuvent être codés dans le corps même du programme du module ou au contraire, être rassemblés en un module de gestion de données. La description des données peut-être celle du SGD, mais elle peut aussi être de beaucoup plus haut niveau (conceptuel par exemple). Ces points, que l'on considérera comme fort importants seront traités spécifiquement dans la section 8.7. L'approche qui y sera proposée consiste essentiellement à découpler l'algorithme de ses accès aux données selon une architecture qui distingue programme d'application, matérialisant l'algorithme, et module de gestion de données, matérialisant les accès aux données.

Remarque : l'approche développée ici semble relever de l'informatique d'application dite lourde, caractérisée par une chaîne de production complexe et coûteuse. On peut en fait y intégrer des solutions beaucoup plus souples, relevant par exemple de l'aide à la décision ou des Centres d'Information (Info-Centre). En particulier, un algorithme LDA peut être traduit non pas par un programme mais simplement par une requête d'un langage d'interrogation. Nous ne nous attacherons cependant pas à cet aspect des choses ainsi que cela avait été spécifié dans le premier volume, I.1.1.

8.5.7 DEFINITION DU SCHEMA INTERNE DES DONNEES

Ce dernier processus consiste à compléter la description de la base de données en précisant les paramètres techniques de celle-ci. Le schéma interne (voir chapitre 4) regroupe les différents composants de cette description. Dans la pratique, surtout pour les SGD non récents, les informations du schéma interne sont dispersées à plusieurs endroits. Tel sera le cas par exemple pour les SGBD CODASYL 71/73 où elles sont contenues surtout dans le schéma DMCL ou SSL (qui est une réalisation très claire de cette notion de schéma interne) mais malheureusement aussi dans le schéma global (Set mode, linked to prior, linked to owner, location mode). La situation s'éclaircira à partir des propositions CODASYL DMCL 1978 (DMCL,78). Dans d'autres systèmes ces concepts sont bien plus mélangés encore.

Le contenu du schéma interne est fortement dépendant du SGD et il est difficile de donner sur ce point des directives générales. Selon les SGD, il faudra préciser l'organisation des fichiers, les paramètres physiques statiques et dynamiques de ceux-ci, leur correspondance avec les fichiers externes, les techniques des types de chemins, les paramètres des clés d'accès calculées, les paramètres et l'emplacement des index, la constitution d'agrégats (clustering), la taille et la politique de gestion des tampons, les principes de journalisation, etc.

D'une manière générale, les paramètres physiques seront choisis de manière à optimiser le fonctionnement global du Système d'Information. On cherchera notamment à minimiser les

temps d'accès sur base de la carte des volumes de "trafic" élaborés lors de la construction du schéma des accès nécessaires (adapté au schéma conforme SGD). On rappelle que cette carte doit mettre en évidence les zones critiques qui influencent le plus fortement la consommation.

La minimisation des temps d'accès n'est pas le seul objectif. On veillera par exemple à maximiser le parallélisme (souvent lié à la découpe en espaces physiques et à la répartition des types d'articles), à minimiser le temps de reprise après incident, à retarder les réorganisations, etc.

8.5.8 EVALUATION DES CONSOMMATIONS DE RESSOURCES

Il est à présent possible de quantifier de manière plus précise les coûts afférant au fonctionnement du Système d'Information, du moins en ce qui concerne les données.

A partir de la quantification statique relative au Schéma des Accès Nécessaires (Section 8.4.5.), on détermine les tailles des populations des types du schéma SGD (nombre d'articles de chaque type, nombre d'articles par valeur de clé, nombre de cibles par chemin, etc.). Du schéma interne, on déduit la taille de chaque objet (page, article de chaque type, fichier, etc.). On détermine ainsi les volumes des fichiers et/ou des bases de données ainsi que les taux de remplissage. Sur base de l'évolution des populations, on détermine également celle des volumes et des taux de remplissage. Dans certains SGBD cependant, ce calcul doit être fait préalablement à la fixation de certains paramètres physiques; il est alors reporté à la définition du schéma interne.

Ceci établi, on calculera le temps moyen d'exécution de chaque opération élémentaire ou primitive sur la base de données. Ce calcul fera intervenir selon les techniques d'accès employées, des paramètres comme la taille des pages, celle des articles, la longueur des clés, leur distribution, les taux de remplissage, la taille des agrégats (p. ex. articles LIGNE groupés autour de leur article COMMANDE) l'âge du fichier depuis sa dernière réorganisation, etc. On consultera notamment (WIEDERHOLD,77), (TEOREY- FRY,82), (WATERS,75), (SEVERENCE,76), (INFOTECH,77) et particulièrement (RUSTIN,77).

On peut alors calculer par module, par phase, par application et pour le projet, les temps d'accès aux données et de leur mise-à-jour par unité de temps. Ce calcul est simplement l'actualisation et la totalisation du tableau des coûts élaboré en 8.4.5. en fonction des valeurs physiques obtenues ci-dessus. Développons brièvement deux exemples.

1. Un module réalise en moyenne 0,5 accès (1 accès avec une probabilité de 0,5) à une séquence à laquelle correspond une clé non identifiante. Chaque séquence contient en moyenne 8 articles. Cette clé étant organisée en calculé avec un taux de remplissage correct (de 0,8 à 0,9 par exemple), il lui correspond en moyenne 1,5 accès physiques pour obtenir le premier article et 0,2 accès physique pour retrouver chacun des autres dans une séquence. On peut donc, en simplifiant (le raisonnement est quelque peu spécieux du point de vue probabiliste), imputer au module 0,5x(1,5+7x0,2) soit 1.45 accès physiques, ou encore 58 msec d'accès au disque, si l'on admet 40 msec par accès physique sur un disque fort partagé.

2. Autre exemple : un module accède 1 fois aux articles LIGNE cibles d'un chemin CL. Un tel chemin contient en moyenne 3 cibles. On décide de créer des agrégats constitués de l'article COMMANDE et des cibles LIGNE de chaque chemin. On calcule que chaque agrégat a une taille moyenne de 260 caractères. Dans des pages de 2500 caractères chargées à 80%, il y a 7,5 agrégats. Pour simplifier, sur 8 agrégats, 7 sont entièrement contenus dans une page tandis que 1 chevauche deux pages. Dans 1 agrégat sur 8, l'une des 3 LIGNES requiert 1 accès physique, toutes les autres n'en consommant aucun, soit de l'ordre de 0,04 accès physique par LIGNE d'une COMMANDE, qu'il s'agisse de la première ou d'une suivante. Le module effectue en moyenne 1x(0,04+2x0,04) soit 0,12 accès physiques, que l'on évalue à 5 msec.

Une évaluation précise des coûts permet de valider (ou au contraire d'invalider) les choix qui ont été faits jusqu'à présent dans un objectif de maximisation des performances. On pense en particulier aux algorithmes effectifs, qui minimisent les accès logiques (du type MAG). Il est clair que l'on ne reposera la question que pour quelques algorithmes critiques. On admettra également qu'une contradiction logique/physique n'apparaîtra que rarement puisque la démarche, à tout instant, ne fait que renforcer les choix optimisants décidés antérieurement. On citera comme exemple la définition du schéma interne qui tente de raffiner sur le plan physique une solution voulue optimale sur le plan logique. Ajoutons enfin que seule une automatisation poussée peut permettre une telle rétroaction. Par contre, une révision des paramètres physiques pourra intervenir plus aisément à ce stade.

Certains SGBD offrent par ailleurs une aide à la conception notamment sous la forme de simulation ou de prototypage du futur système d'information dans ses aspects Base de Données (DBPROTOTYPE d'IMS par exemple).

REMARQUES

1. On sera souvent amené à négliger le temps de traitement (CPU) consommé par un module. Ceci se justifie par la prédominance traditionnellement reconnue des temps d'Entrée/Sortie sur le temps de traitement ainsi que par la difficulté d'évaluer ce dernier. Il faut cependant se souvenir qu'un SGBD est un système complexe dont la consommation en temps de traitement peut être considérable, non seulement pour les SGBD récents (relationnels par exemple, ce qui se conçoit aisément) mais aussi pour certains parmi les plus anciens. Elle est en tout cas d'autant plus grande que les services demandés sont plus nombreux. A titre indicatif, on admet qu'un accès logique n'entraînant qu'un seul accès physique consomme de 1000 à 3000 instructions élémentaires, soit de 1 à 3 msec sur une machine de 1 mips.
2. Le coût en Entrées/Sorties doit tenir compte de l'activité liée à la gestion de la concurrence et à celle de la reprise sur incident. En effet, la tenue d'un journal des images avant et/ou après modification, la création de points de synchronisation (avec recopie forcée du contenu des tampons) ou la gestion de verrous engendrent eux-mêmes un trafic non négligeable avec les mémoires secondaires. En régime de parallélisme serré (transac-

tions très courtes) et à haute sécurité (double journal), on arrivera rapidement à un triplement des accès physiques pour une mise-à-jour des données. Ces situations seront soigneusement analysées et évaluées.

3. Temps de traitement et temps d'Entrées/Sorties constituent des consommations essentielles d'un module. Il en est d'autres que nous nous contenterons de citer : activité du moniteur de télétraitement (notamment gestion des files d'attente de messages), du gestionnaire d'écrans, des modules de synchronisation, du système de gestion des E/S (gestion des files de demandes physiques).

8.5.9 PLAN D'INSTALLATION

Nous nous limiterons ici à la mise en route de la base de données elle-même, en ignorant celle des traitements, traitée spécifiquement dans le troisième volume. Le plan d'installation définit les tâches, ainsi que leur calendrier et les ressources à y affecter, qui conduisent à la mise de la base de données à disposition des utilisateurs. Cette mise à disposition (qui suivrait la phase de réception, selon une optique d'ingéniérie classique) constitue une frontière toute théorique entre installation et exploitation en raison à la fois du caractère fortement évolutif de la plupart des systèmes d'information et de l'urgence de la disponibilité des données. Il est rare qu'une base de données ne soit exploitée qu'après complète installation; il n'est pas rare qu'une base de données ne voit jamais son installation clôturée (l'installation n'est alors que l'amorce de la maintenance).

D'autre part, la définition de ce plan et la mise en route de certaines phases peuvent débuter avant même la réalisation de certaines phases décrites précédemment. En particulier, la collecte et la mise en forme des données de chargement peuvent commencer dès la clôture de l'analyse conceptuelle.

Nous décrirons brièvement les phases importantes de l'installation, en ignorant délibérément les problèmes de choix et de test de SGBD, de changements organisationnels (nouvelles fonctions liées à la base de données, nouvelles méthodes de travail) ou de formation.

- DEFINITION DE LA BASE DE DONNEES. Elle consiste à réserver les ressources en mémoire, à compiler les schémas de définition et à initialiser les espaces de données. La base de données existe mais est vide.
- COLLECTE DES DONNEES. Il s'agit d'une phase qui peut être longue, difficile et délicate. Elle consiste à répertorier, analyser et collecter les données qui vont être chargées dans la base. Les difficultés de la collecte peuvent être de nature politique : en cas de sources multiples d'une même donnée, une seule sera généralement choisie. Elles peuvent être de nature sémantique : les sources peuvent être incohérentes entre elles, mais aussi avec la sémantique de la base de données. Le format des données sera différent de celui qui est spécifié dans la base de données, non seulement en ce qui concerne les données élémentaires (items) mais aussi les macro-structures (types d'articles, associations). Certaines données sont déjà sur support informatique (pour celles-ci, l'installation correspond surtout à une conversion), d'autres sont encore sur papier, quand elles ne relèvent

pas tout simplement de la tradition orale. On élaborera alors des procédures d'extraction et d'encodage. Pour l'essentiel donc, la collecte comporte des tâches relevant de l'analyse conceptuelle.

- PRE-VALIDATION ET MISE EN FORME DES DONNEES. Cette phase consiste à valider les données à charger, et à donner à celles-ci une forme qui facilite ce chargement. Une validation (généralement partielle car il n'y a pas encore d'intégration) sera moins coûteuse à ce stade qu'après chargement. La mise en forme aura pour but d'homogénéiser les données d'origines différentes et surtout d'optimiser le chargement car cette dernière phase peut être très coûteuse (le chargement d'une grosse base de données peut prendre plusieurs semaines). En particulier, si un SGBD n'offre pour garnir une base que les primitives traditionnelles de mise-à-jour, des séquences différentes d'insertion d'articles peuvent entraîner des coûts dans un rapport de 1 à 100 (que l'on songe simplement au garnissage d'un fichier séquentiel indexé par utilitaire de chargement, par insertion aléatoire, par insertion triée directe, par insertion triée inverse). On cherchera autant que possible à permettre un chargement par processus séquentiels tant en lecture des données à charger qu'à l'écriture dans la base. Il est clair que toute validation de contrainte d'intégrité lors du chargement ne pourra que freiner celui-ci. Il y a donc intérêt, sur le plan des performances à soigner la pré-validation, qui n'usera que de techniques légères (parcours séquentiels fichiers simples, tris, fusions, etc). On acceptera par conséquent aussi que le chargement laisse la base de données dans un état potentiellement incohérent.
- CHARGEMENT DES DONNEES. Le garnissage de la base se fera en une seule fois ou de manière incrémentale, de manière à accélérer la mise à disposition de certaines parties de la base. Ce chargement se fera soit par des utilitaires du SGD, soit si ceux-ci n'existent pas ou là où ils ne conviennent pas, par des programmes de chargement spécifiques. On fera remarquer que l'effort d'analyse et de programmation consenti lors du chargement ne se limite pas à cette phase mais qu'il sera également exploitable lors de la maintenance, en particulier lors des phases de réorganisation et de restructuration.
- VALIDATION DE LA BASE DE DONNEES. La phase de chargement ne prenant généralement pas en charge la validation de toutes les contraintes d'intégrité, on procédera à des tests de cohérence et à des corrections. On réalisera également des tests de validation vis-à-vis des utilisateurs. On leur fournira par exemple, sous forme de rapports, des extraits de la base de données, ou on leur permettra de procéder à l'interrogation directe des données, afin qu'ils vérifient la validité de celles-ci.
- TESTS DE PERFORMANCES ET REGLAGES FINALS. Ces tests permettent de vérifier les prévisions antérieures, d'effectuer des corrections de paramètres physiques, d'ajuster les paramètres d'exploitation et de préparer le plan d'exploitation. On réalisera soigneusement l'étude du comportement du SGBD en régime de concurrence, en environnement surchargé et lors d'un incident. Cette étude met fréquemment en lumière des surprises désagréables. C'est pour cette raison qu'il sera prudent de prévoir très tôt (en parallèle avec la conception logique) des tests de comportement sur bases de données prototypes, dans un environnement et des conditions d'exploitation proches de ceux de l'exploitation définitive. En effet, il est rare qu'une approche analytique ou par simulation

permette de prévoir avec une garantie suffisante le comportement d'un système d'une telle complexité.

8.5.10 PLAN D'EXPLOITATION

Ici encore, nous nous limiterons aux problèmes liés à la base de données, dont nous ne retiendrons que les aspects suivants : le suivi et le réglage des performances, la protection contre les incidents et l'évolution qualitative de la base de données.

LES PERFORMANCES

Elles doivent faire l'objet d'une surveillance régulière sous forme du contrôle de l'évolution de quelques paramètres clés que l'on juge représentatifs : taux de remplissage, proportion d'articles en débordement, nombre de niveaux dans les index, taux de défauts de page dans les tampons (ou son complémentaire : la proportion d'accès aux tampons sans transfert physique), tailles moyenne et extrême des files d'attente de ressources ainsi que temps d'attente des processus, équilibre de l'activité des unités physiques, localité des références sur les supports, fréquence des interblocages, fréquence des reprises arrière (rollback), etc. Lorsqu'une situation anormale se produit ou se prolonge, on cherchera un retour à la normale par un ajustement des paramètres physiques de la base de données ou de son exploitation : taille des tampons, verrouillage de parties de ceux-ci, espace en mémoire centrale assigné au SGBD, meilleure répartition des applications dans la journée, modification de la politique de régulation de la concurrence et des techniques de reprise sur incident, meilleure répartition des fichiers sur les unités.

Si ces ajustements simples ne suffisent pas, on procédera à une réorganisation de certaines parties de la base. Cette opération réalise une "remise en ordre" physique des données sans en modifier les structures d'accès, mais en ajustant éventuellement certains paramètres physiques comme la taille des pages et celle des fichiers.

Cette opération est typique des fichiers à accès calculé pour lesquels on augmentera l'espace d'adressage et pour les fichiers indexés. On fera remarquer que pour ces derniers, les techniques dites à auto-réorganisation (basées sur les structures de B-Tree comme VSAM d'IBM, ainsi que la plupart des organisations séquentielles indexées actuelles) n'éliminent en rien la nécessité de réorganiser régulièrement. On montre en effet que quel que soit le taux de remplissage au moment du chargement et la taille des pages, le taux oscille rapidement autour de 70% si l'on ne fait que des additions aléatoires et tend vers 0 pour 50% d'additions et 50% de suppressions d'articles. Une réorganisation peut être coûteuse, surtout dans les bases de données contenant des chemins d'accès inter-articles. Elle est cependant prévisible lorsque le comportement des applications est connu et régulier (on connait donc les taux de mise-à-jour des données) et que les principes algorithmiques du SGBD sont également connus. Le plan de réorganisation peut ainsi être préétabli, sur base d'un calcul économique (coût de la dégradation des performances contre coût de la réorganisation) et de l'opportunité par rapport au calendrier (on ne réorganise pas lors de la clôture des budgets).

Il faut ici faire une remarque importante sur l'aptitude de certains SGBD à admettre des ajustements de la structure physique des données beaucoup plus profonds que ceux qui ont été cités jusqu'ici. Tel est le cas des SGBD dits à fichiers inverses auxquels on joindra la plupart des SGBD relationnels qui au niveau physique du moins, leur sont souvent apparentés. Ces systèmes offrent la possibilité d'ajouter et de retirer une clé d'accès à un type d'articles sans pour autant modifier ces articles ni même les retirer de la circulation durant l'opération. Cette qualité est le propre des clés d'accès réalisées par fichier inverse.

LA PROTECTION CONTRE LES INCIDENTS

On décidera des mesures à prendre et des techniques à adopter en cours d'exploitation et lorsqu'un incident se produit, de manière à minimiser l'impact de ce dernier sur le fonctionnement du Système d'Information. Mesures de protection et incidents entraînent des coûts qu'il convient d'évaluer et de comparer. Les précautions vont du simple planning d'exploitation (p. ex. le traditionnel "batch de mise-à-jour la nuit, consultation pure le jour") jusqu'aux procédures ad hoc (gestion explicite de journaux, utilisation de fichiers différentiels), en passant par les procédures et techniques fournies par le SGBD ou le système d'exploitation (journaux des images avant et après, mises-à-jour retardées, fichiers différentiels, prise de copie incrémentale ou globale). L'étude du plan de protection consiste à établir le relevé des incidents possibles et des traitements qu'ils peuvent perturber, puis à fixer pour chaque traitement et chaque incident la perturbation admise, selon son coût économique direct, son coût indirect (dégradation du service à l'extérieur), son coût psychologique. On élaborera alors les procédures et on choisira les techniques qui permettent de ne pas dépasser ces seuils de perturbation. Les éléments de départ de ce plan doivent en fait être étudiés très tôt, car ce plan est lié aux décisions qui seront prises au plus tard lors de la conception physique (voir 8.5.2, 8.5.3 et 8.5.7 par exemple).

La gestion des journaux constitue également un point important. On calculera le volume créé par unité de temps (plusieurs centaines de millions d'octets par jour n'est pas une quantité exceptionnelle). On déterminera les supports de stockage (par exemple enregistrement en temps réel sur disque, puis déchargement sur bandes). On établira si nécessaire les procédures de fusion et de compactage des journaux, de manière à ne conserver qu'un seul journal ne contenant que la dernière copie de chaque objet modifié. On prévoira l'actualisation de la copie complète de la base de données de manière à disposer lors d'un incident d'une copie moins ancienne, et de diminuer le volume du journal. On veillera également à protéger les journaux eux-mêmes contre les incidents.

EVOLUTION DE LA BASE DE DONNEES

Les Systèmes d'Information sont par nature susceptibles d'évoluer, non seulement du point de vue technique, mais également du point de vue sémantique.

L'évolution technique est bien sûr liée aux changements technologiques (nouvelle version du SGBD, changement de SGBD, nouveau compilateur ou Système d'exploitation), qui doivent être absorbés au moindre coût. Cette évolution est aussi liée aux changements des besoins en

performance, qui vont réclamer non plus de simples ajustements de paramètres physiques, mais des modifications plus profondes qui vont affecter le schéma des accès : ajoût/suppression d'une clé d'accès, d'un type de chemins. Dans certains SGBD (relationnels par exemple) de telles modifications n'affectent en général ni les données telles qu'elles sont stockées, ni les programmes d'application (parfois à une recompilation près). D'autres SGBD (à fichiers inverses par exemple) exigeront des modifications du texte source des applications car les structures modifiées y sont explicitement référencées. Pour d'autres SGBD enfin, il sera nécessaire de reconstruire entièrement la base de données (le coût est celui d'un chargement) selon un processus que l'on nomme restructuration; il sera en outre souvent nécessaire de modifier le texte source de certains programmes d'application. On notera que la technique des modules de gestion de données (section 8.7.) permet d'isoler les programmes de tels changements bien plus que ne le font la plupart des SGBD.

L'évolution sémantique est plus fondamentale car elle est inhérente à la nature même des Systèmes d'Information et des organisations auxquelles ils sont associés. Cette évolution découle de celle de l'organisation elle-même ou de celle de l'objectif assigné au Système d'Information. Elle se traduit par des modifications de la description conceptuelle de la solution. Le problème posé est de minimiser l'impact d'une modification conceptuelle sur la base de données et les traitements. L'indépendance des traitements sur ce plan est l'un des objectifs premiers des méthodes modernes de construction de programmes, qui proposent des solutions en termes de critères de modularisation. Le problème est moins avancé en ce qui concerne les structures de données et l'indépendance des programmes par rapport à celles-ci. Nous aborderons brièvement ces deux sous-problèmes.

1. La possibilité d'évoluer d'une base de données est d'abord liée aux structures de données qui l'organisent ainsi qu'au SGBD qui les gère. Ce dernier est en particulier caractérisé par son aptitude à absorber des extensions du schéma sans exiger un rechargement coûteux des données et ni une modification du texte source des programmes. A titre d'exemple, des fichiers COBOL sont tout-à-fait figés sur ce plan, car on ne peut qu'ajouter un fichier à une "base de données Cobol"; par contre la plupart des SGBD relationnels admettent l'ajout d'une colonne à une table, l'ajout et la suppression d'une table; certains SGBD CODASYL admettent l'ajout d'un item à un type d'articles ainsi que l'ajout de types d'articles et de types de Sets (types de chemins) s'ils n'ont aucun lien avec les anciens. L'approche relationnelle pourra servir de modèle dans la conception d'un schéma SGBD, même si ce SGBD n'est pas relationnel. En effet tout ajout d'un item, d'un type d'articles ou d'un type de chemins peut se traduire par la définition d'un nouveau type d'articles muni des index adéquats. La pauvreté des structures de données conduit à une plus grande souplesse dans l'évolution future.

   Considérons à titre d'exemple les schémas de l'exemple 7.3 et supposons que l'on désire ajouter les types d'articles FACTURE et LIGNE-FACT. On obtient les schémas 8.4, dont on a ignoré les contraintes d'intégrité supplémentaires. Ces deux schémas sont compatibles avec CODASYL.

   Cependant le premier aura exigé une restructuration (avec rechargement) généralement

très coûteuse. Tandis que l'adoption du second permettra de telles extensions à coût immédiat minimum. Il y a cependant des coûts permanents auxquels il faudra consentir : temps d'accès et de mise-à-jour augmentés, complexité accrue des programmes d'accès et de mise-à-jour (en particulier à cause des contraintes d'intégrité référentielles).

2. Le problème de l'indépendance des programmes par rapport aux modifications des structures de la base de données trouvera des solutions qui seront fonction du SGBD. Les SGBD relationnels sont souvent excellents sur ce plan (mécanismes des "vues" et éclatement des structures comme on l'a montré ci-dessus); les SGBD plus traditionnels (tels CODASYL) sont nettement plus faibles (sub-schemas en CODASYL) alors que les fichiers COBOL sont nuls sur ce plan. Une solution générale ne pourra être trouvée que par des techniques de programmation spécifiques telle que celles des modules de gestion de données (voir 8.7). Cette dernière approche permet en effet d'absorber tous les types de modification de la base de données, depuis le changement des techniques physiques jusqu'aux modifications conceptuelles en passant par le changement de SGBD.

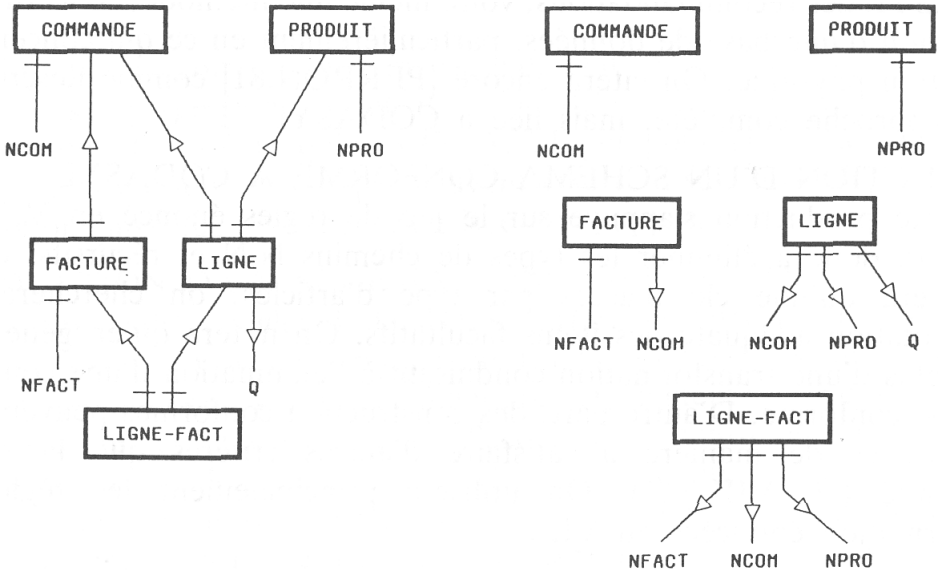


Figure 8.4 - Extension des schémas 7.3

8.6 PARTICULARISATION DE LA DEMARCHE

8.6.1 GENERALITES ET PRINCIPES

La démarche de conception logique et physique exposée en 8.4 et 8.5 fait abstraction du SGD cible. Les phases et les produits qui y sont définis et leur enchaînement ont été présentés d'une manière abstraite. Le but de cette section est de proposer une adaptation de cette démarche à trois approches de gestion des données que l'on considère généralement comme autogonistes : CODASYL, relationnelle et fichiers COBOL simples. On serait alors autorisé à parler d'une méthode de conception d'une base de données CODASYL (ou relationnelle ou COBOL). On constatera que même si les démarches particularisées apparaissent comme différentes, toutes les phases de la démarche générale y jouent un rôle important.

Il va sans dire que l'analyse conceptuelle échappe à cette particularisation. Elle sera donc ignorée ici.

8.6.2 CONCEPTION D'UNE BASE DE DONNEES CODASYL

La conception logique telle qu'elle est développée en 8.4, est ici d'application stricte. La raison essentielle en est que la machine réelle CODASYL n'est pas plus puissante que la machine abstraite LDA/MAG qui définit la couche de conception logique. La conception physique d'une base de données CODASYL reprendra pour l'essentiel les différentes phases décrites en 8.5. On trouvera chez chaque fournisseur de SGBD CODASYL des recommandations, voire même une méthode, concernant la conception d'une base de données, particulièrement en ce qui concerne la conception physique. On citera encore (PERRON,81) comme description d'une approche complète, mais liée à CODASYL 71/73.

PRODUCTION D'UN SCHEMA CONFORME A CODASYL

Cette production s'appuye sur le jeu de règles énoncé en 5.7. On cherchera donc à éliminer les types de chemins N-N et récursifs; on ne conservera qu'une clé d'accès par type d'articles, on cherchera une représentation adéquate des items facultatifs. On notera qu'en général, il existe plus d'une transformation conduisant à l'élimination d'une construction non conforme. D'autre part, des construction conformes peuvent être transformées de manière à satisfaire d'autres critères que la simple conformité (cfr 8.5.2.). On utilisera principalement les règles de transformation énoncées en 7.3.

PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES A CODASYL

Cette phase ne pose pas de problème particulier en CODASYL. On respectera les règles spécifiées en 5.7.

PRODUCTION DU SCHEMA CODASYL

La traduction est assez immédiate. On veillera cependant à laisser de côté la spécification des paramètres techniques (CALC PROCEDURES, LOCATION MODE VIA, LINKED TO OWNER, LINKED TO PRIOR, SET MODE). Les clauses SET SELECTION qui correspondent à une propriété d'usage relèveront des schémas externes. Dans l'optique d'une programmation automatique ou assistée on proposera de la fixer d'autorité à THRU CURRENT, ce qui correspond d'ailleurs à une pratique habituelle.

DEFINITION DES SCHEMAS EXTERNES CODASYL

Cette notion correspond assez clairement aux subschemas CODASYL qui sont, en première approximation, de simples sous-ensembles du schéma. On notera cependant que l'on peut également utiliser à cet effet des modules de gestion de données (section 8.7).

CONSTRUCTION DES PROGRAMMES

Ce point sera traité en 8.7. Des règles systématiques de traduction de LDA en COBOL-DML sont proposées dans (HAINAUT,81).

DEFINITION DU SCHEMA INTERNE DES DONNEES

En CODASYL 71/73, la notion stricte de schéma interne est couverte par certaines clauses du schéma (essentiellement SET MODE, LINKED TO PRIOR, LINKED TO OWNER, LOCATION MODE VIA, LOCATION MODE DIRECT) ainsi que par un schéma physique autonome, rédigé dans un langage appelé Device Media Control Language, Storage Schema Language ou Data Storage Description Language (OLLE,78), (DMCL,78). Jusqu'en 1978, ces langages étaient spécifiques à chaque SGBD. Nous développerons précisément les paramètres physiques pour un SGBD CODASYL 73 au chapitre 9.

8.6.3 CONCEPTION D'UNE BASE DE DONNEES COBOL

Nous désignerons par le terme Base de Données COBOL, la traduction en fichiers, types d'articles et items COBOL d'une base de données conceptuelle. De même qu'en CODASYL, et pour la même raison, les phases de la conception logique peuvent être reprises sans adaptation. La conception physique sera particularisée comme suit.

PRODUCTION D'UN SCHEMA CONFORME A COBOL

En utilisant les règles de la section 7.3, on dérivera du schéma des accès possibles un schéma respectant les règles relatives aux structures COBOL spécifiées en 5.7. On éliminera principalement les types de chemins si l'on accepte des fichiers multi-clés d'accès. Si par contre on veut (ou si l'on doit) se limiter à des fichiers mono-clés, on poursuivra la transformation .P/JP selon la technique illustrée dans l'exemple 7.2, suivie d'une élimination du type de chemins ainsi créé. Les fichiers COBOL à clé d'accès (RELATIVE ou INDEXED SEQUENTIAL) ont la particularité que l'une des clés est un identifiant (voir 5.7.) on sera donc parfois amené

à traduire une proposition de clé non identifiante par une clé identifiante en lui adjoignant un autre item, au besoin artificiel et conçu dans ce seul but. On se souviendra à ce sujet que si une clé d'accès est une clé de tri, tout préfixe de cette clé est aussi une clé d'accès.

PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES A COBOL

Cette phase ne pose pas de problème particulier en COBOL.

PRODUCTION DU SCHEMA COBOL

La spécification des fichiers comporte une composante en ENVIRONMENT DIVISION et une en DATA DIVISION. La première composante spécifiera :

- le nom du fichier COBOL,
- l'organisation : SEQUENTIAL si pas de clé ni d'identifiant, RELATIVE si une clé identifiante dont le domaine des valeurs s'y prête (ce qui est rare en pratique), INDEXED dans les autres cas. On spécifiera dans le dernier cas la RECORD KEY et les ALTERNATE KEYS éventuelles, assorties ou non de WITH DUPLICATE.

La deuxième composante décrira les types d'articles et leurs items.

DEFINITION DES SCHEMAS EXTERNES

COBOL offre peu de possibilités sur ce plan. Il est possible de ne déclarer dans un programme qu'un sous-ensemble des fichiers, d'ignorer certains items (filler) de renommer types d'articles et items. L'usage de module de gestion de données, dont la pratique n'est d'ailleurs pas rare en programmation d'application COBOL, est ici particulièrement indiqué. On notera cependant que COBOL demande que l'on complète la description ci-dessus par deux propriétés d'usage, liées au programme : le mode d'accès et le nom du FILE STATUS.

Dans une approche de programmation manuelle, et sans utiliser de modules de gestion de données, on constituera par schéma externe, deux fichiers contenant les descriptions Cobol. Ces descriptions seront insérées dans les programmes par des instructions COPY.

CONSTRUCTION DES PROGRAMMES

Ce point sera traité en 8.7.

DEFINITION DU SCHEMA INTERNE DES DONNEES

On complétera la description des fichiers par la spécification physique des fichiers et de leur usage : clauses ASSIGN et RESERVE AREA ainsi que la clause BLOCK CONTAINS, là où elle est encore exigée. Les autres caractéristiques physiques seront fixées par utilisation d'utilitaires spécialisés (chargement d'un fichier séquentiel indexé), ou via le système d'exploitation (taille des pages, allocation initiale, incrément d'allocation, etc).

8.6.4 CONCEPTION D'UNE BASE DE DONNEES RELATIONNELLE

Afin de lever d'emblée toute ambigu~ité, précisons que nous n'aborderons pas les problèmes et les techniques de normalisation, de synthèse et de décomposition, habituellement liés aux Base de Données Relationnelles, car ceux-ci sont de nature conceptuelle. Le schéma conceptuel est la description d'une solution soit normalisée, soit non normalisée, mais dont les dépendances "parasites" sont documentées sous la forme de contraintes d'intégrité. Dans le cadre limité de cet ouvrage nous ignorerons de telles contraintes d'intégrité.

SPECIFICITE DES SGBD RELATIONNELS

L'une des propriétés remarquables de la plupart des SGBD relationnels est qu'ils sont à même d'assurer une certaine optimisation des accès relatifs à une requête relationnelle. Cette aptitude existe tant dans le langage interactif que dans le langage de programmation d'application. Elle se traduit, selon les concepts du MAG, par le fait que le SGBD réel offre des primitives d'accès aux données dont la condition de sélection évaluable par accès est considérée comme un filtre quelconque par la plupart des SGD. Le programmeur peut donc souvent désigner les données requises par une condition exprimée de façon naturelle, en ignorant les chemins et clés d'accès réellement disponibles. Selon la démarche générale, l'algorithme prédicatif, rendu conforme au SGBD, peut être considéré par le programmeur comme un algorithme affectif puisque son expression pourra être prise en charge par le SGBD. En principe donc, la démarche générale se simplifie pour le programmeur.

Dans cette optique, l'optimisation des accès est confiée au SGBD. Les programmes peuvent être, aux réserves près qui seront énoncées ci-dessous, plus simples, plus sûrs, plus stables et de maintenance plus aisée. Cette approche a cependant ses limites, qui conduiront parfois à préférer une optimisation explicite (par le programmeur) des accès.

Une première raison pourrait être l'extrême diversité des SGBD du point de vue des conditions de sélection admises et des possibilités d'optimisation. Ceci est vrai non seulement de par la variété des langages (citons simplement les langages dérivés de SEQUEL (DATE,81), d'INGRES (DATE,81) ainsi que les langages semi-algorithmiques tels RDB de DEC), mais aussi, pour un même langage, de stratégies d'optimisation très différentes allant des plus raffinées aux plus grossières et qui fournissent donc des solutions qui ne seront pas toujours optimales. On serait donc amené à construire autant de méthodes de conception qu'il y a de SGBD relationnel lorsque l'optimisation est un point crucial. L'optimisation explicite résoud tous ces problèmes de variété et de non fiabilité de l'optimisation par un "nivellement par le bas".

Une deuxième raison que l'on pourrait invoquer est l'indépendance des dossiers de conception par rapport au SGBD. Ainsi, si l'on envisage une gestion de la base de données par plusieurs SGD, ou encore un changement de SGD, même dans une même famille, il est important que les dossiers de la conception logique soient indépendants du SGD, afin de minimiser l'effort d'adaptation et de conversion.

Une troisième raison est qu'aucun SGBD n'effectue une optimisation globale d'un programme d'application qui contiendrait plusieurs commandes d'accès aux données. Si le programmeur peut se reposer sur le SGBD pour l'optimisation locale à une commande d'accès isolée, il doit par contre assurer entièrement celle de l'enchaînement des commandes (boucles emboitées par exemple); on sait en effet que dans ce domaine, l'optimisation globale n'est pas toujours réalisée par l'ensemble des optimisations locales. Ici encore, les parts d'optimisation par le SGBD et d'optimisation explicite dépendent du SGBD.

Une quatrième raison est liée aux contraintes de certains SGBD, qui obligent le programmeur à définir la construction de larges tables résultats par jointure de nombreuses tables de base. Il est alors nécessaire de rééclater algorithmiquement les données obtenues de manière à retrouver leur hiérarchie naturelle. Ceci se réalise par un contrôle explicite de ruptures à plusieurs niveaux, ce qui ne manquera pas de ravir le programmeur COBOL chevronné, mais qui nous éloigne de la transparence des algorithmes à laquelle nous avait habitués les SGBD traditionnels.

Il est donc des situations où l'optimisation explicite pourra être préférée, sinon imposée, et d'autres où la pleine puissance des SGBD relationnels sera utilisée. Dans le cadre de cet ouvrage, nous définirons deux approches selon la puissance que l'on reconnait au SGBD relationnel.

1. Si l'on s'engage dans l'optimisation explicite, on admet que les seules conditions évaluables par accès correspondent à l'accès séquentiel et à l'accès par clé à un type d'articles. On rattacherait à l'optimisation explicite l'approche qui admettrait comme évaluable toute condition portant sur les valeurs des items d'un type d'articles, qu'il y corresponde ou non un accès par clé.
2. L'optimisation implicite quant à elle admet comme évaluable, toute condition LDA qui correspond à une condition comprise par le SGBD. Il est clair que cette notion est fonction du SGBD.

Nous développerons séparément les deux approches. Cependant, deux phases leur sont communes : la construction du schéma MAG conforme au relationnel et sa traduction dans le langage relationnel. Ces schémas sont en effet indépendants des accès effectifs et donc de toute considération d'optimisation. Ceci demanderait en fait à être nuancé, car des transformations de schéma pourraient être proposées pour des raisons d'optimisation d'accès, de modularisation, de lisibilité, etc (par exemple duplications d'items, fusions ou éclatements de types d'articles).

La conception d'une base de données relationnelle a, comme on s'en doute, fait l'objet de nombreuses études. Nous citerons simplement les ouvrages (DATE,81), (DELO-ADI,83) et (BEN-ROL,79) auxquels on se reportera.

CONCEPTION BASEE SUR UNE OPTIMISATION IMPLICITE

PRODUCTION D'UN SCHEMA CONFORME AU RELATIONNEL

Ce schéma qui sera utilisé par les programmeurs ne spécifie pas les accès nécessaires. Il peut donc dériver directement du schéma des accès possibles. On y éliminera notamment les types de chemins, les items répétitifs et les items décomposables (cfr 5.7.). On pourra à cette occasion rencontrer le problème du type d'articles qui n'a pas d'identifiant et qui doit être référencé par un autre type d'articles (de telles situations sont créées par les transformations 7.3. et 7.4. qui permettent d'éliminer des types de chemins). On sera alors amené à attacher à ce type d'articles un identifiant artificiel. Ce problème, et quelques autres sont abordés dans (DATE,83), chapitre 5. Il n'y a en principe dans ce schéma conforme aucune clé d'accès, sinon parfois celles qui jouent le rôle de support d'identifiant. Par contre, on veillera à relever soigneusement les contraintes d'intégrité (dites "référentielles" selon la terminologie relationnelle) engendrées par les transformations 7.3. et 7.4.

PRODUCTION DU SCHEMA RELATIONNEL

La traduction est en général immédiate pour l'essentiel : définition des tables (ou relations) et de leurs colonnes (ou attributs). Lorsque le SGBD le permet, on indiquera les attributs admettant la valeur NULL et on précisera l'interprétation de celle-ci (valeur inconnue, valeur absente, attribut non pertinent, etc.). On consultera à nouveau (DATE,83) sur ce point. On spécifiera les contraintes dites référentielles (certain SGBD les prennent en charge) ainsi que les identifiants. Dans plusieurs SGBD, les identifiants sont déclarés comme caractéristiques d'une clé d'accès. On sera donc forcé de déclarer de telles clés si l'on désire que le SGBD assure l'unicité des valeurs. Les contraintes d'intégrité non prises en charge par le SGBD doivent être assurées par les programmes de mise-à-jour ou, plus judicieusement par les modules de gestion de données.

PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES AU RELATIONNEL

L'optimisation des accès étant assurée par le SGBD, la phase de production des algorithmes effectifs (8.4.4.) ne se justifie pas lorsque ceux-ci sont simples. Les algorithmes conformes dérivent donc alors immédiatement des algorithmes prédicatifs. On sera cependant amené à décomposer un algorithme prédicatif lorsqu'il fait usage de constructions non acceptées par le SGBD. Cette transformation pourra dans certains cas être .P/JP assez profonde, notamment lorsque les restrictions du SGBD sont assez sévères (par exemple, une seule séquence d'accès active à la fois). Un autre type de transformation consistera à regrouper des boucles d'accès emboitées de manière à préparer leur traduction sous la forme d'une jointure relationnelle. Celle-ci est en effet dans certains SGBD la forme privilégiée de désignation d'ensembles de données qualifiés par une condition d'association (INGRES, RDB, DATACOM/DB par exemple). Cette transformation particulière est généralement inutile dans les SGBD admettant les requêtes emboitées (du type SQL et assimilés). On rappelle une fois de plus qu'il s'agit de transformations de conformité et non d'optimisation.

DEFINITION DES SCHEMAS EXTERNES RELATIONNELS

La plupart des SGBD relationnels offrent des mécanismes de définition de "vues" qui peuvent spécifier soit de simples sous-ensembles du schéma global soit de nouvelles relations virtuelles construites à partir de celles du schéma selon des expressions plus ou moins complexes. Une vue peut prendre en charge une part algorithmique non négligeable. Sa définition peut donc interagir avec la phase suivante. Dans une optique plus large, on utilisera également des modules de gestion de données.

CONSTRUCTION DES PROGRAMMES

Ce point sera traité en 8.7. Signalons cependant que dans des situations simples, un module peut se traduire par une requête rédigée dans le langage interactif.

DEFINITION DU SCHEMA INTERNE DES DONNEES

Le schéma interne sera déterminé principalement par le choix des clés d'accès à attribuer à chaque table ou relation. Ce choix est lié aux algorithmes optimaux que le SGBD associera à chaque condition de sélection de l'algorithme conforme. Nous adopterons le principe qui consiste à déterminer un jeu initial de clés d'accès qui conduit à des performances raisonnables de l'ensemble des algorithmes. Toute erreur sera aisément corrigée (clé inutile, trop coûteuse ou manquante) en cours d'exploitation si le SGBD admet, comme c'est souvent le cas, l'ajout et le retrait a posteriori d'une clé d'accès sans perturbation majeure des programmes.

Pour déterminer ce jeu de clés d'accès, nous choisirons pour chaque algorithme prédicatif un algorithme effectif (non conforme au relationnel) et nous en déduirons le schéma des accès nécessaires. En rendant ce dernier schéma conforme en relationnel, on obtient une spécification des clés d'accès parmi lesquelles on choisit celles qui sont susceptibles d'être le plus utiles (fréquence d'utilisation, taille des tables). L'hypothèse posée est que les procédures d'optimisation du SGBD choisiront, à défaut de trouver mieux, d'utiliser les index que l'on a retenu. Cette procédure étant identique à celle de l'optimisation explicite, nous ne la détaillerons pas plus longuement. Rappelons cependant qu'ici, l'optimisation des algorithmes et l'élaboration du schéma des accès nécessaires n'ont en principe aucun impact sur la programmation.

Outre la présence et les composants des clés d'accès, on précisera également, selon les SGBD, les techniques de réalisation de ces clés, les espaces physiques de stockage des données, la taille des incréments d'extension, le mode rangement des lignes d'une table (en vrac, par agrégats selon un index, par agrégats selon une expression de jointure entre deux tables, taux de remplissage initial, etc.).

CONCEPTION BASEE SUR UNE OPTIMISATION EXPLICITE

Dans cette approche, la démarche est strictement similaire à celle qui est adoptée pour les SGD traditionnels. On y trouvera donc toutes les phases de la conception logique car le SGBD relationnel n'est pas ici considéré comme plus puissant que la machine abstraite LDA/MAG. Nous décrirons brièvement les phases de conception physique et leur enchaînement.

PRODUCTION D'UN SCHEMA CONFORME OU RELATIONNEL

Cette phase est celle de 8.6.4.2., mais elle conduit en outre à la spécification des clés d'accès.

PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES AU RELATIONNEL

Ces algorithmes se déduisent des algorithmes effectifs sur machine abstraite LDA/MAG. Les transformations principales sont induites par l'élimination des types de chemins lors de l'obtention du schéma conforme. Il se peut cependant que des transformations plus profondes interviennent, de manière à produire un enchaînement de primitives conforme au SGBD. Tel sera le cas lorsque l'algorithme propose deux parcours emboités d'une même table et que le SGBD n'admet pas cette structure.

PRODUCTION DU SCHEMA RELATIONNEL

Cette phase est celle de 8.6.4.2.

DEFINITION DES SCHEMAS EXTERNES RELATIONNELS

Sur les principes, cette définition ne diffère pas de celle de 8.6.4.2.

CONSTRUCTION DES PROGRAMMES

Ce point sera traité plus spécifiquement en 8.7. Signalons cependant que certains SGBD relationnels offrent un langage de manipulation de Données (LMD) doté d'une structure de boucle d'accès sur des séquences de données. Ceci conduit souvent à une simplification de la traduction.

DEFINITION DU SCHEMA INTERNE DES DONNEES

Pour l'essentiel, ce schéma précisera les clés d'accès qui paraissent utiles parmi celles qui sont définies dans le schéma conforme ou relationnel. On fixera également les différents paramètres physiques proposés par le SGBD (cfr. 8.6.4.2.).

8.7 LA PROGRAMMATION SUR BASE DE DONNEES

8.7.1 INTRODUCTION

Nous aborderons dans cette section quelques aspects fondamentaux propres à la programmation d'application sur bases de données (au sens large). Ces aspects seront présentés d'une manière générale, puis resitués dans le cadre de la démarche qui fait l'objet de ce chapitre.

L'un des critères essentiels sur lesquels sont basées les architectures de programmes est l'indépendance de ceux-ci par rapport à la base de données. Même restreinte au domaine des don-

nées, la notion de dépendance d'un programme peut présenter plusieurs facettes. On admettra qu'un programme (ou une procédure) dépend d'un aspect spécifique des données si un changement dans l'organisation des données selon cet aspect peut entraîner une modification du texte source de ce programme. Le programme est par conséquent indépendant selon cet aspect si ce changement n'a pas d'impact. Cette indépendance sera réalisée par le SGBD ou par la programmation elle-même. Examinons quelques exemples d'une manière progressive.

Au niveau le plus bas se situe l'Indépendance Physique, selon laquelle les paramètres techniques de représentation des données, de leur stockage et de leur accès ne doivent pas être connus pour rédiger un programme de manipulation des données. Ce type d'indépendance est en général assuré par les SGBD actuels.

On peut également requérir l'indépendance par rapport au schéma SGBD des données. Elle est assurée par le SGBD via des mécanismes de "subschema" (CODASYL) de "PCB" (IMS) ou de "views" (systèmes relationnels). Sauf dans le dernier exemple cité, le SGBD ne peut protéger les programmes que vis-à-vis de classes très limitées de modifications du schéma.

Un programme peut être lié à une classe de SGBD tout en étant indépendant de chacun de ses membres. C'est ainsi que l'on écrira un programme travaillant sur une base de données relationnelle quel qu'en soit le SGBD. En l'absence de standardisation officielle cette indépendance ainsi que les suivantes, ne pourront être assurées que par une programmation adéquate.

On peut concevoir un programme de manière telle qu'il ne soit lié à aucun SGBD. Tel est le cas d'un programme qui ne connait de la base de données que son schéma des accès selon le MAG, avant transformation pour conformité.

Un programme peut encore être indépendant des accès disponibles et donc n'être lié qu'au schéma conceptuel des données (ou, selon l'approche décrite, du schéma des accès possibles).

L'indépendance par rapport au schéma conceptuel est assurée si, suite à une modification de ce schéma, il est possible de définir une correspondance ("mapping" en anglais) qui permette de présenter au programme les données sous leur forme antérieure. Cette vision stable des données est assurée par la notion désormais traditionnelle de Schéma Externe (voir chapitre 4). Il y correspond le concept d'Indépendance Logique.

On peut encore définir un autre type d'indépendance que l'on estimera équivalent ou même supérieur au précédent. Dans cette approche, les données ne sont pas décrites par des modèles généraux et ne sont pas manipulables par des primitives standard et générales offertes par un SGBD. Elles sont plutôt définies par un module (au sens moderne du terme) dont la spécification est liée au programme ou à une classe de programmes. Ces principes sont aussi ceux d'une approche par les types abstraits. Il s'agit du type d'interface avec les données que propose M. Jackson (JACKSON,83). Cet auteur ne propose cependant aucune recommandation concernant la structuration des données et la construction des procédures de gestion et d'accès.

Il ressort clairement de ces exemples que les SGBD actuels ne garantissent qu'à un faible degré l'indépendance des programmes par rapport aux données. Quand bien même il se limiterait à des valeurs traditionnelles comme les SGBD CODASYL ou les fichiers COBOL, le

programmeur se heurterait à des problèmes parfois complexes de conversion, dus non seulement aux différences entre standard ou rapports successifs (un seul exemple mais d'importance : le rapport CODASYL DBTG 1971 stipule que l'identifiant d'un record type ne l'est que dans un même fichier tandis qu'en 1973 sa portée est étendue à toute la base de données), aux interprétation variées d'un même standard par les différents fournisseurs, et aussi aux différentes versions et révisions d'un même standard chez un même fournisseur.

La situation pourrait cependant évoluer favorablement quant à la stabilisation et à l'adoption généralisée de normes concernant le modèle et les langages de bases de données. Citons par exemple les propositions de l'ANSI (ANSI,85) concernant NDL (Network Data Languages) et RDL (Relational Data Language) actuellement un développement. Cependant, on ne peut en attendre de résultats effectifs sur le plan commercial avant plusieurs années. En outre, l'adoption de telles normes ne suffit pas à résoudre les problèmes d'indépendance liés à l'évolution de la base de données. Par conséquent, dans la plupart des situations, l'indépendance est à charge du programmeur.

C'est dans ce contexte que nous introduirons la notion de module de gestion de données ou module d'accès.

8.7.2 NOTION DE MODULE DE GESTION DE DONNEES ou MODULE D'ACCES

Un module de gestion de données est une entité exécutable à laquelle le programmeur peut faire appel lorsqu'il désire effectuer une opération sur une base de données. Un module est défini par :

- la structure des données qu'il gère,
- les primitives qu'il offre pour gérer ces données et pour y accéder,
- les régles d'enchaînement de ces primitives.

Ces propriétés étant aussi celles que l'on reconnait généralement aux SGBD, on admettra qu'un tel module constitue un SGBD virtuel, ainsi qu'on le verra plus loin. Concrètement, un module se présentera le plus souvent sous la forme d'une procédure ou sous-programme indépendant. Il pourra également être réalisé par remplacement de texte : les références aux primitives se trouvant dans le texte source du programme sont remplacées par le développement de la primitive ou par une séquence d'appels à un sous-programme gérant la base de données. Ce remplacement peut être réalisé par un simple éditeur de textes (en particulier ceux qui offrent des possibilités de macro-instructions), par un préprocesseur spécialisé, ou par un macro-générateur généralisé.

Un cas particulier de cette méthode sera parfois utilisé en COBOL : chaque invocation d'une primitive est exprimée par un PERFORM, l'ensemble des sections concernées étant inséré à la compilation par un COPY. Le langage PL/1 offre lui aussi des possibilités de macro-instructions parfaitement adaptées dans ce cas. Une autre technique enfin consisterait à réaliser le module sous la forme d'un processus autonome unique, qui communiquerait avec les dif-

férents processus (programmes) utilisateurs des données. Cette technique, plus complexe, permet une centralisation de la gestion.

Citons, à titre d'illustration, quelques rôles que l'on peut assigner à un module d'accès.

- Un module d'accès est un moyen de réaliser un schéma externe d'une base de données. En d'autres termes, il permet d'offrir à ses utilisateurs une vue de la base de données qui est un sous-ensemble, et une transformation du schéma réel de la base. Il peut, par exemple, fournir des données calculées à partir des données de la base, ou émuler des structures de données inconnues du SGBD réel.
- Un module d'accès, quand on peut obliger une classe d'utilisateurs à passer par lui, permet d'assurer la confidentialité et le contrôle d'accès aux données.
- Il est aisé de faire vérifier par un module d'accès des contraintes d'intégrité que le SGBD ne peut assurer lui-même. On augmente ainsi considérablement la garantie d'intégrité des données.
- Un module d'accès peut inclure des techniques de reprise sur incident et la gestion des données y afférente de manière à pallier les lacunes du SGBD sur ce plan.
- Dans le même domaine, il assurera la réalisation de transactions dans un environnement multi-ressources (p. ex. une base de données et communications avec un terminal, plusieurs SGBD distincts). On consultera sur ce point (GRAY,78) et (DATE,83).
- De la même manière, un module d'accès peut inclure des techniques de régulation de la concurrence d'accès aux données.
- On pourra inclure dans un module d'accès des points de mesure de manière à récolter des statistiques sur l'usage des données et les performances du SGBD. Ces mesures peuvent être retirées sans dommage pour les programmes d'application.
- Un module d'accès pourra tenir une comptabilité des ressources consommées par chaque programme utilisateur.
- Il pourra homogénéiser l'accès à une base de données gérée par plusieurs SGBD (p.ex. CODASYL + fichiers COBOL, SQL + fichiers VSAM).
- Il isolera les programmes utilisateurs des changements de SGBD.
- Il isolera les programmes utilisateurs des modifications du schéma SGBD, des modifications du schéma d'accès (p. ex. clé d'accès disparue mais simulée par recherche séquentielle), et même des modifications du schéma conceptuel, à condition que les données anciennes soient dérivables à partir des nouvelles. C'est ainsi que les programmes peuvent être insensibles au fait que deux types d'articles ont fusionné, qu'un type d'articles a éclaté, que les nom, longueur, type, position d'un item ont été modifiés.

Cette notion de module assurant l'indépendance présente par ailleurs le même intérêt dans tous les domaines où la technologie ou parfois même les concepts fondamentaux sont susceptibles d'évoluer ou ne se conforment pas à des règles, un modèle ou un standard imposés. Il en sera ainsi des communications entre processus (dans une machine et entre machines), des accès à des périphériques, de manipulation graphique, etc. A chacun de ces domaines on peut associer (et l'on associe effectivement) des modules assurant l'indépendance des programmes

d'application par rapport aux particularités syntaxiques et sémantiques des sous-systèmes utilisés. On fera enfin remarquer que les dernières propositions ANSI en matière de langages de bases de données (ANSI,85)) incluent la notion de modules d'accès dans lesquels sont cachées les instructions de manipulation de données. Cette notion est cependant plus restreinte que celle qui va être développée dans ces pages.

De manière à limiter l'analyse, nous n'aborderons que le problème des modules définissant un modèle de données et offrant à l'utilisateur (programmeur) un ensemble de primitives permettant d'accéder aux données et de les gérer.

La spécification d'un tel module se compose du modèle de données, des primitives offertes et des enchaînements de primitives qui sont admis. C'est donc naturellement dans le cadre de la démarche de conception, qui définit quatre niveaux de schémas, que nous trouverons des critères de construction de modules d'accès liés aux modèles de données.

8.7.3 LES MODULES D'ACCES DANS LA DEMARCHE DE CONCEPTION DE SYSTEMES D'INFORMATION

Rappelons brièvement les étapes importantes de la conception logique et de la conception physique.

- Le schéma des accès possibles : il s'agit d'une expression systématique du schéma conceptuel des données en MAG. Il est indépendant de toute contrainte technique ou de performance.
- Les algorithmes prédicatifs : un tel algorithme dérive directement de la spécification d'un module. En ce qui concerne la désignation des données auxquelles le module accède dans la base de données, l'algorithme utilise une description qui précise les propriétés de ces données (par une condition de sélection ou prédicat) et non la manière d'y accéder.
- Les algorithmes effectifs LDA/MAG : l'algorithme effectif est un développement efficace de l'algorithme prédicatif. On y spécifie les accès qui devront être utilisés pour obtenir dans la base de données les données nécessaires à l'algorithme. Le critère essentiel est ici de minimiser le nombre des accès logiques à la base de données. Cet algorithme est fonction des primitives d'accès disponibles. On admettra en l'occurrence que l'on dispose des primitives de base du MAG, c'est-à-dire celles qui sont relatives aux accès séquentiel, par clé et par chemin.
- Le schéma des accès nécessaires : ce schéma ne reprend du schéma des accès nécessaires que les structures de données et d'accès qui sont utilisées par les algorithmes effectifs. Certaines redondances d'optimisation peuvent apparaître.
- Le schéma conforme au SGD : ce schéma est obtenu par une transformation du schéma des accès nécessaires de manière à le rendre conforme au SGBD.
- Les algorithmes conformes au SGD : l'algorithme effectif LDA/MAG est adapté aux particularités du SGD cible. Cette adaptation a une double origine. D'une part la transformation du schéma des accès nécessaires en schéma des accès SGD, qui rend le premier conforme aux structures de données du SGD cible, entraîne une transformation parallèle

des algorithmes qui utilisaient les constructions disparues. D'autre part, l'algorithme doit être également conforme au SGD cible en ce qui concerne les primitives auxquelles il fait appel.

- Le schéma SGD : il est obtenu par l'expression du schéma MAG conforme dans le langage de définition de données du SGD.
- Les programmes.

Considérons un algorithme prédicatif quelconque, que nous noterons A4. Il lui correspond un algorithme effectif (noté A3), auquel on associe un algorithme conforme (noté A2).

Ce qui distingue fondamentalement l'algorithme A2 des algorithmes A3 et A4, c'est sa conformité au SGD réel; les deux autres en sont indépendants, et donc ne lui sont généralement pas conformes. Rappelons qu'un algorithme est conforme à un SGD si aux accès qu'il spécifie ne sont associées que des conditions de sélection directement évaluables par les primitives du SGD, si les commandes de mise-à-jour sont celles du SGD, et si la structure du programme est, du point de vue des accès, conforme à ce que permet le SGD.

On peut donc aisément traduire l'algorithme A2 en un programme P1 dans lequel les accès ont été traduits en invocations des primitives du SGD (généralement via un Langage de Manipulation de Données). Le programme P1 sera dit de niveau 1. Il est "non indépendant" du SGD.

Supposons à présent qu'il existe un SGBD tel que l'algorithme A4 lui soit conforme. Dans une telle hypothèse il serait donc possible de traduire immédiatement chaque accès de A4 en invocation des primitives d'accès de ce SGBD. On obtiendrait ainsi un programme de niveau 4 (noté P4). Pour chaque séquence apparaissant dans une expression de désignation de données (par exemple une boucle FOR) le SGBD offrirait les primitives d'accès ou de mise-à-jour (on se limitera ici aux primitives PREMIER et SUIVANT). Cette situation est illustrée dans le schéma 8.5.

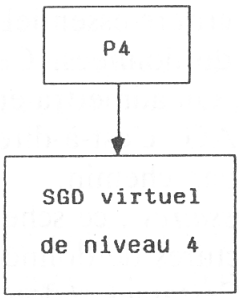


Figure 8.5 - Architecture de niveau 4

Bien qu'un tel SGBD n'existe pas à l'état naturel, il est possible de le construire en interposant entre le programme P4 et le SGD réel un module dont le rôle est de convertir les appels de P4

en appels au SGD réel. Nous appellerons ce module un MODULE D'ACCES. La mise en oeuvre de cette architecture se présente comme indiqué dans le schéma 8.6.

On définirait de même un SGBD virtuel de niveau 3 pour l'algorithme effectif MAG (A3) en créant un module d'accès de niveau 3. Ce SGBD offrirait des services plus simples et banalisés que le module de niveau 4, puisqu'il ne définirait rien d'autre qu'un SGBD du type MAG, fournissant les primitives de base traditionnelles (principalement accès séquentiel, par clé et par chemin, mise-à-jour ponctuelles). On serait ainsi amené à écrire un programme de niveau 3 (noté P3), qui ferait appel à ce module. .J0/P

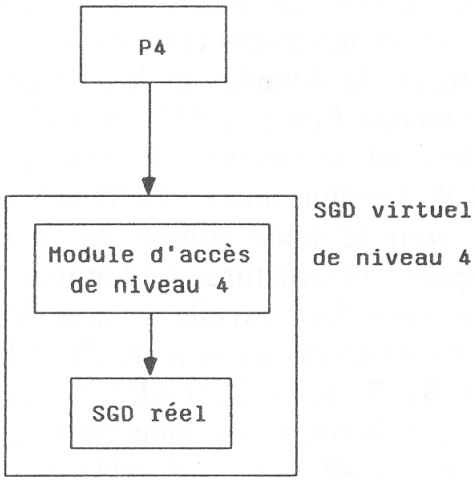


Figure 8.6 - Architecture de niveau 4 - détail

Ce principe d'architecture peut également être adopté pour un algorithme effectif SGBD (A2). Dans ce cas, le SGBD virtuel de niveau 2 aurait les mêmes fonctionnalités que le SGD réel. Le programme de niveau 2 (noté P2) qui en dériverait aurait même structure que le programme de niveau 1, à ceci près que les appels au SGD réel y seraient remplacés par des appels au module d'accès. Ce module d'accès (de niveau 2) serait un simple habillage des primitives du SGD réel.

En appelant MA4, MA3, MA2 les modules d'accès des niveaux 4, 3 et 2, respectivement, on schématisera les différentes architectures possibles comme indiqué dans le schéma 8.7.

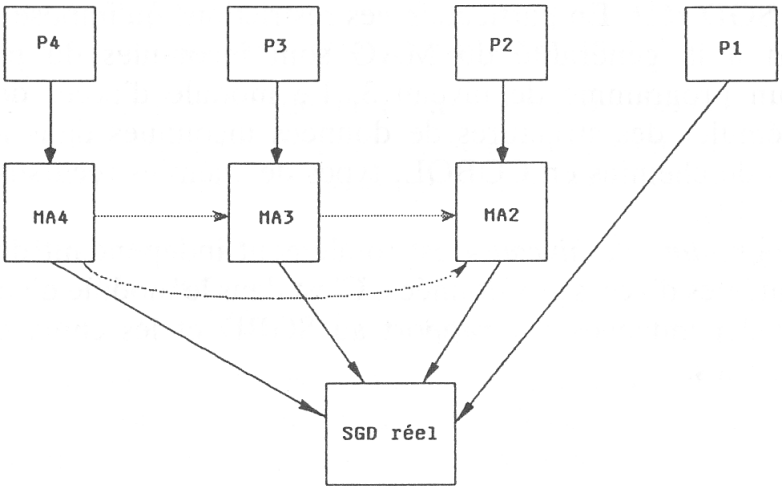


Figure 8.7 - Synthèse des architectures

On y a représenté en traits interrompus des architectures dérivées, dans lesquelles coexistent des modules d'accès de plusieurs niveaux, MA4 pouvant faire appel à MA3 ou MA2, et MA3 pouvant faire appel à MA2. Les modules d'accès eux-mêmes peuvent ainsi jouir de l'indépendance qui est accordée aux programmes P3 et P2.

Analysons à présent, le rôle que jouent les modules d'accès du point de vue de l'indépendance des programmes (P4, P3, P2, et P1 en l'occurrence) par rapport aux structures de données et à leur gestion.

Un programme de niveau 1 est par définition totalement dépendant du SGD. Toute modification, même mineure, de la syntaxe ou de la sémantique du DML (changement de version, autre SGBD de même type) entraîne la nécessité de mettre à jour le texte source de ce programme. Quant à l'indépendance par rapport aux structures de données, elle est celle du SGD. L'indépendance logique sera donc excellente pour les SGBD relationnels, faible pour les SGBD CODASYL à partir de 78 et IMS (DL/I), médiocre pour les SGBD CODASYL 71 et 73, TOTAL, etc., et nulle pour les fichiers COBOL. Quant à l'indépendance physique (par rapport aux techniques physiques d'implémentation), elle est en général assez satisfaisante partout à l'heure actuelle.

Un programme de niveau 2 est indépendant des caractéristiques syntaxiques du DML du SGD. Le SGBD virtuel utilisé par le programme correspond à une restriction du MAG aux fonctionnalités du SGD réel. Le module d'accès (de niveau 2) qui y correspond offre par exemple l'indépendance par rapport à un changement de version du SGBD, ou à son remplacement par un SGBD de la même famille, ou encore par son remplacement par un SGBD d'une autre famille, mais limité aux fonctionnalités du premier (CODASYL remplaçant un

SGBD relationnel utilisé d'une manière restreinte, ce dernier remplaçant des fichiers COBOL).

Un programme de niveau 3 est totalement indépendant du SGD, mais non des stratégies d'accès. Le programme exploite efficacement les services d'un SGBD virtuel obéissant aux spécifications du MAG. Ce SGBD virtuel est simulé par le module d'accès (de niveau 3), qui cache les dépendances par rapport au SGD réel. En particulier, les restrictions qu'impose le SGD réel par rapport à la généralité du MAG sont inconnues du programmeur rédigeant un programme de niveau 3. Le module d'accès devra donc à l'occasion émuler des structures de données inconnues dans le SGD réel (p.ex. types de chemins en COBOL, types de chemins récursifs ou N-N en CODASYL 83).

Un programme de niveau 4 est totalement indépendant du SGD ainsi que des stratégies d'accès aux données. C'est dans le module d'accès que sont cachées les dépendances par rapport au SGBD et les choix de stratégies d'accès efficaces.

8.7.4 SPECIFICATION D'UN MODULE D'ACCES

Il s'agit avant tout de faire un choix quant au modèle de données, aux primitives et aux enchaînements autorisés.

LE MODELE DE DONNEES

Il peut être ou non lié au SGD réel. On définira ainsi des modules offrant des structures de données du type relationnel, hiérarchique (ou plus exactement arborescent), réseau-CODASYL, réseau-TOTAL, fichiers traditionnels mono ou multi-clés. On pourra à l'occasion simplifier ou enrichir le modèle dont on s'inspire, non seulement en ce qui concerne les structure de données (p.ex. plusieurs clés d'accès par type d'articles en CODASYL 73) mais aussi dans le domaine des contraintes d'intégrité (p.ex. vérification des contraintes référentielles dans certains systèmes relationnels et pour des fichiers simples.) Il est tout-à-fait envisageable de définir un module dont le modèle est du niveau conceptuel. Nous nous intéresserons surtout aux modules du type MAG.

Un module peut être conçu comme étant général, c'est-à-dire capable de manipuler n'importe quelle base de données selon le modèle choisi, ou au contraire comme étant lié à une base de données spécifique. La première solution est la plus lourde car le module, tout comme un SGBD, doit pour chaque exécution de primitive, consulter une description de la base de données, vérifier la validité de la demande, construire les arguments à transmettre au SGD réel, faire appel à celui-ci (à condition qu'il soit activable par une instruction CALL, ce qui est très fréquent), puis mettre en forme les résultats selon les directives de la description déjà mentionnée.

Cette approche interprétative peut être remplacée par un approche compilée dans la seconde solution. Dans ce cas, les instructions du module font référence aux objets de l'unique base de données à gérer. La description de celle-ci est en quelque sorte "cablée" dans le module. Il y aura un module par base de données, ou le plus souvent un module par schéma externe

de cette base (celui-ci étant par exemple lié à une phase, ce qui conduit à un module d'accès par phase).

LES PRIMITIVES

Bien que celles-ci soient liées au modèle, de grandes variations sont possibles. Examinons quelques critères de choix importants.

Le module peut-il créer de nouveaux types de données, c'est-à-dire modifier, même temporairement, le schéma. La chose est aisée dans certains systèmes relationnels, ou à l'autre extrême, en BASIC.

Les primitives correspondent-elles à des actions sur des ensembles (systèmes relationnels, à fichiers inverses, APL) ou sur des éléments ponctuels (systèmes CODASYL, fichiers et langages traditionnels).

Lors de l'accès à des articles dont la condition de sélection porte sur des items, n'admet-on que la spécification de clés d'accès (une ou plusieurs) ou au contraire admet-on un filtre quelconque, le module choisissant alors la stratégie d'accès.

Les primitives d'accès ponctuel sont définies par la séquence des articles et par la position de l'article demandé. On peut pour chaque séquence possible définir soit trois primitives de base (définition de la séquence, accès au premier, accès au suivant) soit deux (accès au premier d'une séquence, accès au suivant) soit une seule (accès au suivant d'une séquence; lors de la première activation on obtient le premier).

Pour activer la primitive d'accès au suivant dans une séquence, faut-il spécifier l'article duquel on veut obtenir le suivant, ou cet article est-il implicitement le dernier auquel on a accédé dans la séquence. Dans le premier cas (le plus puissant) le courant de la séquence est géré par le progamme utilisateur. Il peut exister un nombre quelconque de courants. Dans le second (le plus simple) il existe un courant implicite géré par le module. Il est plus difficile de définir plusieurs courants autres que ceux qui sont implicites. Les CURSOR de SQL tombent dans le premier cas, tandis que les CURRENTS de CODASYL relèvent du second.

Les valeurs d'items sont-elle obtenues par une primitive spécifique ou à l'occasion de l'accès à un article. Faut-il spécifier les items concernés ou obtient-on les valeurs de tous les items.

Le module réalise-t-il d'autres primitives annexes : gestion de transaction, gestion de la concurrence. Réalise-t-il d'autres fonctions : audit, prise de mesure, gestion de la sécurité.

Quelle est la forme concrète des paramètres. Comment désigner (code, nom explicite) un type d'articles, un item, un type de chemins. Comment désigner un article. Comment transmettre et recevoir des valeurs d'item.

Quels sont les événements, naturels (article trouvé ou non trouvé) ou exceptionnels (type d'article inexistant, incident système), dont le programme utilisateur est avisé. Un interblocage est-il résolu automatiquement ou signalé au programme. Quels sont les codes de diagnostic qui sont transmis au programme.

Les paramètres sont-ils vérifiés par le module ou celui-ci suppose-t-il que ces paramètres sont corrects (acceptable dans le cadre d'une génération automatique des programmes).

Gère-t-il la notion de version. Le module est-il conservé lorsque le schéma de la base de données est étendu.

Définit-on des primitives de gestion de listes de références d'articles.

LES ENCHAINEMENTS DE PRIMITIVES AUTORISES

La solution la plus simple pour la construction du module est évidemment de transmettre simplement les contraintes du SGD réel (modules du niveau 2). Il est cependant possible de simplifier ou d'étendre ces règles. Citons quelques choix à faire :

Faut-il "ouvrir" une base de données, un fichier, un type d'articles, avant d'y accéder ou au contraire le premier accès provoque-t-il une ouverture automatique. Peut-on ouvrir plusieurs fois un même objet (ceci est important si plusieurs modules indépendants accèdent au module). Faut-il "fermer" un objet ouvert, combien de fois.

Peut-on réaliser deux parcours simultanés de chemins du même type (par exemple pour vérifier que deux COMMANDES distinctes ont des LIGNES semblables). Peut-on emboiter deux parcours aux articles du même type (si le type PERSONNE a pour items NUM, PERE et MERE appartenant au même domaine, et si NUM est clé d'accès, peut-on accéder aux PERSONNES d'une LOCALITE, et pour chacune accéder à ses parents).

Dans une séquence d'articles, est-on forcé de demander le suivant du dernier obtenu ou peut-on perturber cet ordre.

Peut-on redemander un article auquel on vient d'accéder.

Peut-on alterner les accès à un type d'articles par deux clés d'accès distinctes.

Peut-on accéder à l'article que l'on vient de créer, de modifier.

8.7.5 CONSTRUCTION D'UN MODULE D'ACCES

Nous examinerons brièvement quelques problèmes spécifiques liés à ce type de modules.

CHOIX DU SGD

Le SGD auquel le module d'accès fait appel sera le plus souvent le SGD réel. Il peut cependant faire appel à un module d'accès de niveau inférieur. Cette architecture a été discutée en 8.7.3., schéma 8.7.

GESTION DES COURANTS

On appelle généralement COURANT en matière de base de données le dernier article manipulé dans une séquence déterminée. A ce courant correspond un registre qui en contient la référence. Cette notion existe dans tous les SGD qui admettent l'accès ponctuel (current en CODASYL; associé à la notion de cursor en SQL/DS d'IBM; un seul, implicite, par fichier COBOL ouvert). Il existe dans notre contexte trois types de courants, ceux du SGD réel, ceux

connus du programme utilisateur lorsqu'il communique avec le module d'accès, et les courants internes au module d'accès. Ces courants peuvent être identiques (dans certains modules du niveau 2). mais ils seront souvent différents.

Les registres courants du SGD (c'est-à-dire les registres qu'il connait et dont il assure automatiquement la gestion) sont soit imposés (CODASYL), soit définis par l'utilisateur (SQL), soit implicites (fichiers COBOL, INGRES).

Un registre courant d'un module d'accès contient la référence d'un article. La représentation que l'on choisira pour cette dernière est fonction des possibilités offertes par le SGD pour désigner univoquement un article. Lorsque la chose est possible, on utilisera l'identifiant interne qu'assignent certains SGBD à chaque article (database key en CODASYL 71/73, RELATIVE KEY en COBOL). On utilisera aussi la valeur de l'identifiant explicite si le type d'articles en possède. Dans d'autres cas, le module construira lui-même cette référence. Tel est le cas d'un fichier séquentiel pour lequel la référence au courant est le rang de cet article dans le fichier. Le module connait ainsi les positions courantes dans toutes les séquences d'articles actives. Dans certains cas difficiles, on peut être amené à ajouter à un type d'articles un item technique compact jouant le rôle d'identifiant. Cet item ne sera connu que du module d'accès. La cas d'un SGBD CODASYL 71/73 montre clairement ce que peut être la distinction entre les courants du SGBD et ceux du module d'accès. CODASYL connait un seul courant par SET TYPE (grossièrement un type de chemins), alors que le module d'accès, comme tout programme d'application, peut repérer plusieurs courants pour ce SET TYPE par stockage de leur database-keys dans des variables adéquates (MOVE CURRENCY STATUS). Le repositionnement dans le SET TYPE se commandera par l'instruction FIND USING. Dans les SGBD CODASYL 78, la valeur d'une database-key n'est plus accessible, mais peut se manipuler indirectement via le mécanisme des KEEP-LIST.

Les courants du programme utilisateur sont généralement associés aux boucles d'accès et correspondent aux variables de ces boucles. La synchronisation entre courants du programme et courants du SGD est assurée par le module d'accès, via ses propres courants. Cette synchronisation peut être complexe dans les modules de niveau 3 et 4, si l'on n'admet aucune restriction sur les enchaînements de primitives.

STRUCTURE GENERALE D'UN MODULE D'ACCES

Un module sera généralement constitué d'une procédure (ou section de code) par primitive, et pour chacune, d'une procédure par type d'objet concerné.

STRUCTURES DES PRIMITIVES D'ACCES A UNE SEQUENCE D'ARTICLES

Ces primitives seront généralement l'accès au premier article et l'accès à l'article suivant. La gestion des courants mise à part, le codage des primitives ne pose pas de problèmes particuliers pour les modules de niveau 2 et souvent également de niveau 3. Par contre la construction d'un module de niveau 4, et parfois de niveau 3 peut être plus délicate. Illustrons cette construction par un exemple réduit, basé sur l'algorithme prédicatif suivant :


```
for COM := COMMANDE (CC: CLIENT(: NOM=X))
     print DATE(:COM);
endfor;
```

Admettons encore qu'il y corresponde l'algorithme effectif ci-dessous, obtenu par simple développement :

```
for CLI := CLIENT(: NOM=X)
     for COM := COMMANDE (CC: CLI)
          print DATE(:COM);
     endfor;
endfor;
```

Développons l'algorithme prédicatif de manière à n'utiliser que des accès ponctuels (le format des arguments est ici purement arbitraire) :

```
PREMIERE-COM("COMMANDE(CC:CLIENT(:NOM=X))",COM,TROUVE);
While TROUVE do
     LIRE-DATE(COM,DATE); print DATE;
     COM-SUIVANTE("COMMANDE(CC:CLIENT(:NOM=X))",COM,TROUVE);
endwhile;
```

On met ainsi en évidence les primitives PREMIERE-COM et COM-SUIVANTE, ainsi que LIRE-DATE (que nous négligerons). La construction d'un module d'accès offrant ces primitives s'appuye sur le développement de la boucle d'accès qui a conduit à l'algorithme effectif. Ce dernier nous donne donc la structure du module, qui se présente comme suit :

```
PREMIERE-COM : goto FIRST;
COM-SUIVANTE : goto NEXT;
LIRE-DATE    : goto DATE;
FIRST :for CLI:=CLIENT(:NOM=X)
           for COM:=COMMANDE(CC:CLI)
                TROUVE:=true;
                return;
NEXT  :
           endfor;
     endfor;
TROUVE:=false;
return;
```

On y observe que l'activation du module par COM-SUIVANTE s'effectue à partir de l'endroit où la dernière activation s'est terminée, comme si le module "appelait" le programme utilisateur pour lui faire traiter l'article livré. Programme et module se comportent donc en coroutines. Ce que traduit la structure des branchements du module.

On y observe également que la livraison d'un article obtenu et la poursuite des accès (NEXT) s'effectuent à l'intérieur de la boucle la plus emboitée. Une telle structure suppose que lorsque l'on appelle le module pour effectuer la primitive COM-SUIVANTE, celui-ci soit exactement dans l'état où il était avant le dernier retour. Cette condition risque de n'être pas toujours réalisée en raison de l'usage illicite des boucles qui est fait dans cet algorithme (quitter une boucle ou une branche d'un if et y entrer par un branchement direct). Nous transformerons donc le code du module de façon à y éliminer les structures de contrôle complexes dont la gestion est basée sur des variables d'état qui risquent d'être perturbées lors des entrées et des retours du module. Développons d'abord systématiquement les boucles FOR en supposant l'existence des primitives de niveau inférieur PREM-CLI, COM-DE-CLI, CLI-SUIV :

```
PREMIERE-COM : goto FIRST;
COM-SUIVANTE : goto NEXT;
LIRE-DATE    : ...

FIRST :    PREM-CLI("CLIENT(:NOM=X)",CLI,CLI-TROUVE);
           while CLI-TROUVE do
                PREM-COM-DE-CLI("COMMANDE(CC:CLI)",COM,COM-TROUVEE);
                while COM-TROUVEE then
                     TROUVE:=true;
                     goto SORTIE;
NEXT  :
                     COM-DE-CLI-SUIV("COMMANDE(CC:CLI)",
                                     COM,COM-TROUVEE);
                endwhile;
                CLI-SUIV("CLIENT(:NOM=X)",CLI,CLI-TROUVE);
           endwhile;
           TROUVE:=false;
SORTIE:    return;
```

Remplaçons enfin les boucles WHILE (et les alternatives complexes) par des structures n'employant que l'instruction IF-GOTO.

```
PREMIERE-COM : goto FIRST;
COM-SUIVANTE : goto NEXT;
LIRE-DATE    : ...

FIRST :    PREM-CLI("CLIENT(:NOM-X)",CLI,CLI-TROUVE);
TEST-CLI : if not CLI-TROUVE then goto FIN-CLI; endif;
```



```
          PREM-COM-DE-CLI("COMMANDE(CC:CLI)",COM,COM-TROUVEE);
TEST-COM : if not COM-TROUVEE then goto FIN-COM; endif;
               TROUVE :=true;
               goto SORTIE;
NEXT :
               COM-DE-CLI-SUIV("COMMANDE(CC:CLI)",
                                    COM,COM-TROUVEE);
               goto TEST-COM;
FIN-COM :  CLI-SUIV("CLIENT(:NOM=X)",CLI,CLI-TROUVE);
           goto TEST-CLI;
FIN-CLI :  TROUVE := false;
SORTIE :   return;
```

Par transformation systématique de la structure d'accès de l'algorithme effectif, nous obtenons ainsi un algorithme qui n'exige aucune mémorisation d'états liés aux structures de contrôle. La seule exigence sur le langage dans lequel cet algorithme sera traduit est qu'il offre des possibilités de variables rémanentes. On trouvera dans (HAINAUT,84) un développement concernant ce type de transformations algorithmiques.

Tout au long de cet exemple, nous avons choisi la nature et le format des arguments d'un module de manière à rendre simple et claire l'utilisation de celui-ci, mais sans autre souci de réalisme. En pratique, un module pourrait se présenter sous la forme d'une procédure dotée des paramètres suivants :

Données :

- code de la primitive (p. ex. : accès au suivant par clé),
- type d'articles concerné (p. ex. : CLIENT);
- mécanisme utilisé (p.ex. : clé NOM);
- valeur de clé (p.ex. : valeur de X);
- référence de l'article dont on désire le suivant;

Résultats :

- référence de l'article trouvé
- diagnostic (code de retour)

Ainsi, l'appel à CLI-SUIV s'exprimerait comme suit :

```
MODULE3 ("SUIVANT-CLE", "CLIENT", "NOM", X, CLI, CLI, RETOUR)
```

Chapitre 9 : ETUDE DE CAS

9.1 INTRODUCTION

Nous développerons dans ce chapitre la solution d'un cas selon deux hypothèses. La première hypothèse est celle d'une forte simplification, pour des raisons évidentes liées aux limites de cet ouvrage. En particulier, nous réduirons le nombre des fonctions de traitement à un échantillon minimal, ainsi que la complexité du schéma des données et celle des fonctions. D'autre part, certains aspects dont le principe a été abordé dans la partie théorique seront ignorés; tel sera le cas des problèmes de reprise sur incident, de concurrence, d'installation et d'exploitation. Par contre, nous conduirons la solution jusqu'à son terme opérationnel, sous la forme de schémas et de programmes compilables. La deuxième hypothèse est celle d'une implantation multi-cible, qui nous conduira à définir des solutions en termes de fichiers COBOL (COBOL ANS-74) , d'une base de données CODASYL DBTG 71/73 (DBMS-20 de DEC) ainsi que d'une base de données relationnelle (SQL/DS d'IBM). On couvre ainsi les solutions traditionnelles simples, les solutions Base de Données traditionnelles, ainsi que les solutions Base de Données récentes. Signalons encore, comme le lecteur averti ne manquera pas de le remarquer à l'occasion, que nous avons parfois plus visé la démonstration méthodologique que l'efficacité absolue des solutions.

BREVE DESCRIPTION DU SYSTEME REEL

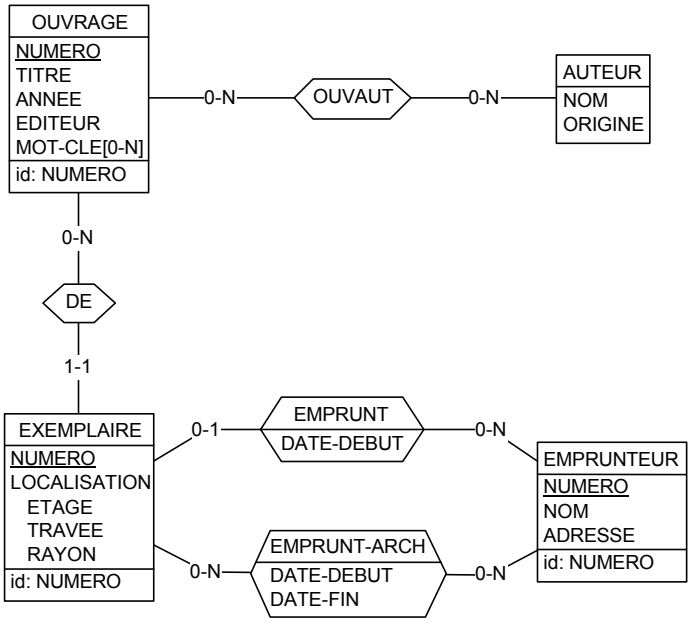
Une bibliothèque gère le stockage, l'indexation et l'emprunt d'un ensemble de livres. Chaque livre est un exemplaire d'un ouvrage, auquel peuvent correspondre un nombre quelconque d'exemplaires. Un ouvrage est caractérisé par un numéro identifiant, un titre, ses auteurs éventuels, l'année de parution, l'éditeur et les mots-clés qui le décrivent. On désire connaître, si possible, l'origine de chaque auteur. Chaque exemplaire est caractérisé par un code identifiant et sa localisation (étage, travée, rayon). D'un emprunteur on connaît le numéro identifiant, le nom, l'adresse, les exemplaires en cours d'emprunt accompagnés de la date d'emprunt, ainsi que tous les emprunts clôturés, avec leurs dates de début et de fin. La bibliothèque assure les fonctions et services traditionnels relatifs aux activités mentionnées ci-dessus.

9.2 LES RESULTATS DE L'ANALYSE CONCEPTUELLE

Nous retiendrons ici le schéma conceptuel des données, la spécification statique de trois fonctions (élémentaires) ainsi que la quantification de données et des traitements.

9.2.1 LE SCHEMA CONCEPTUEL DES DONNEES

Sa représentation graphique (schéma 9.1) est complétée des contraintes d'intégrité supplémentaires. On y relève les types d'entités OUVRAGE, EXEMPLAIRE, EMPRUNTEUR et AUTEUR et les types d'associations correspondants. La notion d'emprunt a été éclatée en emprunt en cours (EMPRUNT) et emprunt archivé (EMPRUNT-ARCH) en raison de la différence de leurs contraintes d'intégrité et de leur rôle dans le système. Nous n'en dirons pas plus quant à la description sémantique des données.



Contraintes supplémentaires :

1. Une association EMPRUNT-ARCH est identifiée par EXEMPLAIRE et DATE-DEBUT
2. Pour toute entité EXEMPLAIRE, la DATE-DEBUT de l'EMPRUNT éventuel n'est pas inférieur aux DATE-FIN de ses EMPRUNT-ARCH.

Figure 9.1 - Schéma conceptuel de la BD BIBLIO

Les caractéristiques des attributs sont les suivantes :

```
NOM d'AUTEUR          : alpha 35
ORIGINE d'AUTEUR      : alpha 50
NUMERO d'OUVRAGE      : numer 6
TITRE d'OUVRAGE       : alpha 80
```

```
ANNEE d'OUVRAGE      : date
EDITEUR d'OUVRAGE    : alpha 45
MOT-CLE d'OUVRAGE    : alpha 20
NUMERO d'EXEMPLAIRE  : numer 8
 ETAGE d'EXEMPLAIRE  : alpha 4
 TRAVEE d'EXEMPLAIRE : alpha 4
 RAYON d'EXEMPLAIRE  : alpha 4
 NUMERO d'EMPRUNTEUR : numer 8
 NOM d'EMPRUNTEUR    : alpha 35
 ADRESSE d'EMPRUNTEUR:alpha 60
 DATE-DEBUT d'EMPRUNT: date
 DATE-DEBUT d'EMPRUNT-ARCH : date
 DATE-FIN d'EMPRUNT-ARCH   : date
```

9.2.2 SPECIFICATION DES TRAITEMENTS

Dans le cadre de l'application de gestion des emprunts, nous ne retiendrons que trois fonctions

- Fonction 1 : étant donné le titre, l'année d'édition et le nom d'un auteur d'un ouvrage, fournir le numéro d'un exemplaire disponible.
- Fonction 2 : enregistrer la restitution d'un exemplaire emprunté de numéro donné.

- Fonction 3 : établir un relevé des numéros des exemplaires en retard de restitution (c'est-à-dire empruntés depuis plus de 14 jours) ainsi que du nom de l'emprunteur.

Nous serons cependant amenés, dans l'une ou l'autre phase ultérieure de conception, à faire intervenir des caractéristiques d'autres processus, de manière à rendre le cas plus réaliste.

9.2.3 ELEMENTS DE QUANTIFICATION

LES DONNEES

1. Il y a 160.000 OUVRAGES, 60.000 AUTEURS, 420.000 EXEMPLAIRES, 2000 EMPRUNTEURS.

2. Il y a de 0 à 8 AUTEURS par OUVRAGE, avec une moyenne de 1,7. On recense 15% d'OUVRAGES sans AUTEUR. Il y a donc en moyenne 2 AUTEURS par OUVRAGE ayant au moins 1 AUTEUR.

3. Il y a de 0 à 15 OUVRAGES par AUTEUR. De 2. on déduit qu'il y a une moyenne de 4,5 OUVRAGES par AUTEUR. Il y a également 10% d'AUTEURS sans OUVRAGE, et donc une moyenne de 5 OUVRAGES par AUTEUR ayant au moins un OUVRAGE.

4. Il y a de 0 à 20 EXEMPLAIRES par OUVRAGE, avec une moyenne de 2,7. Sachant que l'on compte 90% d'OUVRAGES d'au moins 1 EXEMPLAIRE, on déduit que ceux-ci ont aussi 3 EXEMPLAIRES en moyenne.

5. 50.000 exemplaires ont été empruntés, de 1 à 15 fois, avec une moyenne de 4. Il y a une moyenne de 2000 EMPRUNTS en cours, concernant 1000 EMPRUNTEURS. 10% sont de plus de 14 jours; durée moyenne : 1 semaine.

6. Il y a de 0 à 10 valeurs de MOT-CLE par OUVRAGE, avec une moyenne de 2. On admet que 1/3 des OUVRAGES n'ont pas encore reçu de MOT-CLE. Il y a en moyenne 16 OUVRAGES par MOT-CLE.

7. Il y a en moyenne 3 AUTEURS par NOM présent dans la base de données.

LES TRAITEMENTS

Fonction 1 : 200 activations par jour dont 10% échouent parce que l'ouvrage spécifié n'est pas trouvé.

Fonction 2 : 400 par jour.

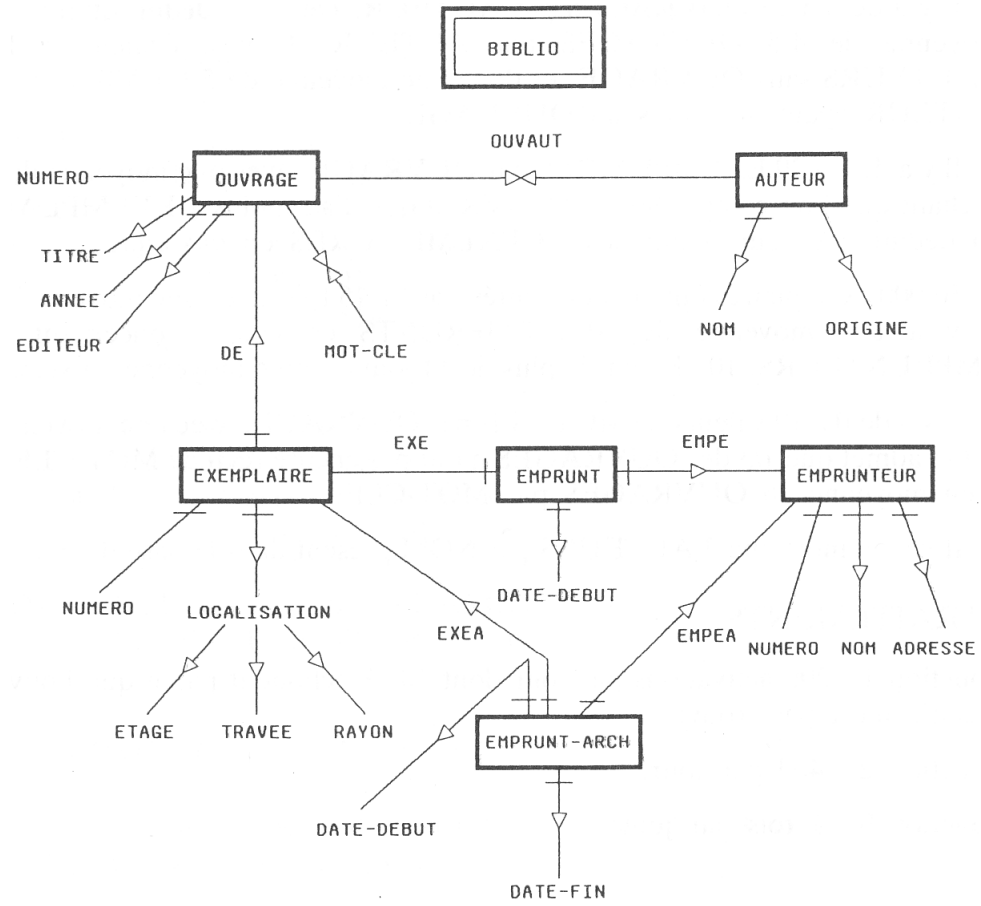
Fonction 3 : 1 fois par jour.

9.3 LA CONCEPTION LOGIQUE

Nous donnerons ici une solution standard telle que décrite en 8.4. Celle-ci pourra être revue et adaptée pour certains SGD (relationnels notamment).

9.3.1 CONSTRUCTION DU SCHEMA DES ACCES POSSIBLES

Le résultat est représenté dans le schéma 9.2. On a choisi de représenter le type d'associations EMPRUNT par le type d'articles EMPRUNT et les types de chemins associés. Il eut été possible d'associer DATE-DEBUT à EXEMPLAIRE comme item facultatif et de définir un type de chemins 1-N entre EMPRUNTEUR et EXEMPLAIRE (remarquons que par les transformations 7.4, on peut s'assurer que ces deux solutions sont équivalentes).



Contrainte supplémentaire :

Pour tout e d'EXEMPLAIRE et pour tout a d'EMPRUNT-ARCH(EXEA:e) :

DATE-DEBUT(:EMPRUNT(EXE:e)) >= DATE-FIN(:a)

Figure 9.2 - Schéma des accès possibles de la BD BIBLIO

QUANTIFICATION DES DONNEES

160.000 OUVRAGES (+ 4400 par an)

60.000 AUTEURS (+ 3300 par an)

420.000 EXEMPLAIRES (+ 12320, -220 par an)

2.000 EMPRUNTEURS (+ 220 par an)

2.000 EMPRUNTS (+ 400, - 400 par jour)

200.000 EMPRUNT-ARCH (+ 400 par jour)

OUVAUT : 0 à 8 AUTEURS/OUVRAGE, moyenne 1,7
15% d'OUVRAGES sans AUTEUR
0 à 15 OUVRAGES/AUTEUR, moyenne 4,5
10% d'AUTEURS sans OUVRAGE.

DE : 0 à 20 EXEMPLAIRES/OUVRAGE, moyenne 2,7
10% d'OUVRAGES sans EXEMPLAIRE

EXE : 0 à 1 EMPRUNT/EXEMPLAIRE, moyenne 0,005
99,5% d'EXEMPLAIRES sans EMPRUNT

EMPE : moyenne de 1 EMPRUNT/EMPRUNTEUR
50% d'EMPRUNTEURS sans EMPRUNT

EXEA : de 0 à 15 EMPRUNT-ARCH/EXEMPLAIRE, moyenne 0,47
88% d'EXEMPLAIRES sans EMPRUNT-ARCHI

EMPEA : moyenne de 100 EMPRUNT-ARCH/EMPRUNTEUR

MOT-CLE par OUVRAGE : de 0 à 10, 2 en moyenne. 33% d'OUVRAGES sans MOT-CLE.
En moyenne 16 OUVRAGES par MOT-CLE présent dans la base de données.

Distribution des EMPRUNTS par DATE-DEBUT : cfr 9.2.3.

9.3.2 ELABORATION DES ALGORITHMES PREDICATIFS

Nous associerons un et un seul module à chaque fonction de traitement. Les aspects du dialogue avec l'environnement et de validation sont réduits à leur plus simple expression sous la forme de procédures d'acquisition "input" et d'affichage "print". On associe à chacun de ces modules les algorithmes 9.1, 9.2 et 9.3.

```
for B := BIBLIO do
     input(XT, XAN, XAUT);
     for EX := #1 EXEMPLAIRE ((EXE: 0 EMPRUNT) and
                              (DE: OUVRAGE ((: TITRE = XT)  and
                                            (: ANNEE = XAN) and
                                  (OUVAUT: AUTEUR (:NOM = XAUT)))) do
          print NUMERO(:EX);
     endfor;
endfor;
```

Alg 9.1. Algorithme pr{dicatif du module 1

```
for B := BIBLIO do
     input(XEX);
     for E := EMPRUNT (EXE: EXEMPLAIRE (: NUMERO = XEX)) do
```

```
          create EA := EMPRUNT-ARCH ( (EXEA:  EXEMPLAIRE (EXE: E)) and
                (EMPEA: EMPRUNTEUR (EMPE: E)) and
                (: DATE-DEBUT = DATE-DEBUT(:E)) and
                (: DATE-FIN = DATE-DU-JOUR));
          delete E;
     endfor;
endfor;
```

Alg 9.2. Algorithme pr{dicatif du module 2

```
for B := BIBLIO do
     for E := EMPRUNT (:DATE-DEBUT < MOINS(DATE-DU-JOUR,14) ) do
          print NUMERO(: EXEMPLAIRE(EXE: E)),
                NOM(: EMPRUNTEUR(EMPE: E));
     endfor;
endfor;
```

Alg 9.3. Algorithme pr{dicatif du module 3

9.3.3 DEVELOPPEMENT DES ALGORITHMES EFFECTIFS

ALGORITHME EFFECTIF DU MODULE 1

La condition de sélection de la boucle "for EX" n'est pas évaluable par accès. Il convient donc d'en effectuer le développement. Plusieurs solutions étant possibles, elles sont déduites du sous-schéma de la condition (schéma 9.3), qui reprend le sous-ensemble des éléments du SAN cités dans cette dernière, indépendamment du sens des accès, pour l'instant indéterminé.

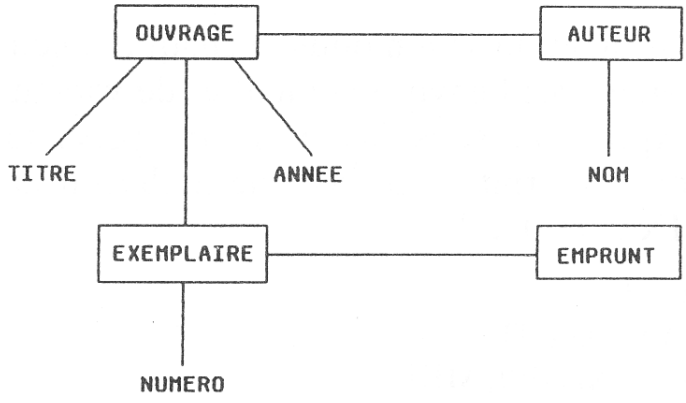


Figure 9.3 - Sous-schéma de la condition de module 1

Cette condition se prête à un développement sous forme d'arborescence (un point d'entrée, puis accès aux noeuds associés). Les racines possibles sont EXEMPLAIRE, OUVRAGE, TI-

TRE, ANNEE, AUTEUR et NOM (EMPRUNT n'en est pas une à cause de la forme de la condition où il intervient). Nous examinerons brièvement les algorithmes correspondant à chacun de ces six points d'entrée.

1. L'accès à partir d'EXEMPLAIRE (la condition devient un filtre) constitue un accès séquentiel avec évaluation de la condition pour chaque article.

   Sur les 420.000 articles on en parcourera en moyenne la moitié (210.000). L'évaluation de la condition impliquera presque toujours un accès à l'OUVRAGE car seuls 2000 exemplaires sont empruntés (210.000 accès), et rarement des accès à AUTEUR. Soit une moyenne de 420.000 accès, ce qui est intuitivement prohibitif. Cet algorithme est abandonné.

2. L'accès à partir d'OUVRAGE représentera en moyenne 80.000 accès séquentiels ce qui est également prohibitif.

3. L'accès à partir de TITRE (clé d'accès) est manifestement à retenir car le TITRE est "presqu'identifiant". On notera cependant que les valeurs de cet item sont de longues chaînes de caractères et qu'il est malaisé de donner le titre exact d'un ouvrage. Il faut donc utiliser des techniques d'accès adéquates, par exemple utilisant un codage phonétique du type SOUNDEX qui tolère certaines erreurs et imprécisions. L'accès fournit alors les OUVRAGES dont le TITRE ressemble à la valeur fournie. La sélectivité est évidemment plus faible. Il en résultera aussi que l'algorithme peut obtenir plusieurs EXEMPLAIRES desquels seul l'opérateur pourra, par examen visuel, déterminer celui qui est retenu. L'algorithme effectif devra donc prévoir ce dialogue, ce que nous ignorerons ici. On retiendra cette solution pour examen ultérieur.

4. L'accès à partir de l'ANNEE (clé d'accès) ne peut être évalué exactement que si l'on connaît la répartition du nombre d'OUVRAGE par ANNEE. Ainsi, si la plupart des OUVRAGES concernent les 40 dernières années, on obtient en moyenne 4.000 OUVRAGES par ANNEE (cette valeur est supérieure dans notre algorithme si l'on admet qu'une ANNEE est d'autant plus demandée qu'il lui correspond plus d'OUVRAGES). Cette solution peut être écartée sans autre analyse.

5. L'accès à partir de l'AUTEUR (accès séquentiel) implique en moyenne 30.000 accès à AUTEUR, ce qui nous fait rejeter la solution.

6. L'accès à partir de NOM d'AUTEUR (accès par clé) paraît favorable : 3 AUTEURS par NOM et 4,5 OUVRAGES par AUTEUR. L'algorithme est retenu.

Cette analyse sommaire conduit à retenir comme candidats l'algorithme qui part de TITRE d'OUVRAGE et celui qui part de NOM d'AUTEUR. Nous en ferons une étude plus approfondie afin de les départager. Leurs accès sont illustrés dans les schémas 9.4. Chacun de ces schémas est un arbre construit par orientation des arcs du schéma 9.3. Cet arbre représente le "squelette" de l'algorithme du point de vue des accès. On y a classé les arcs en deux catégories : ceux qui conduisent, directement ou non, aux données cherchées et qui se représentent en traits doubles, et ceux qui ne conduisent à des données que pour vérification de conditions, et qui se représentent en traits simples. La première catégorie représente des ac-

cès soit présents initialement dans l'algorithme prédicatif, soit mis en évidence suite au développement d'une condition, et qui se traduiront généralement par une boucle d'accès principale. La deuxième catégorie représente les accès nés du développement d'un filtre, et qui conduiront généralement à la valorisation d'un booléen de sélection. On y a ajouté à titre documentaire, pour chaque arc, le nombre moyen d'éléments cibles obtenus pour chaque élément origine, tel qu'on l'a établi en 9.3.1.

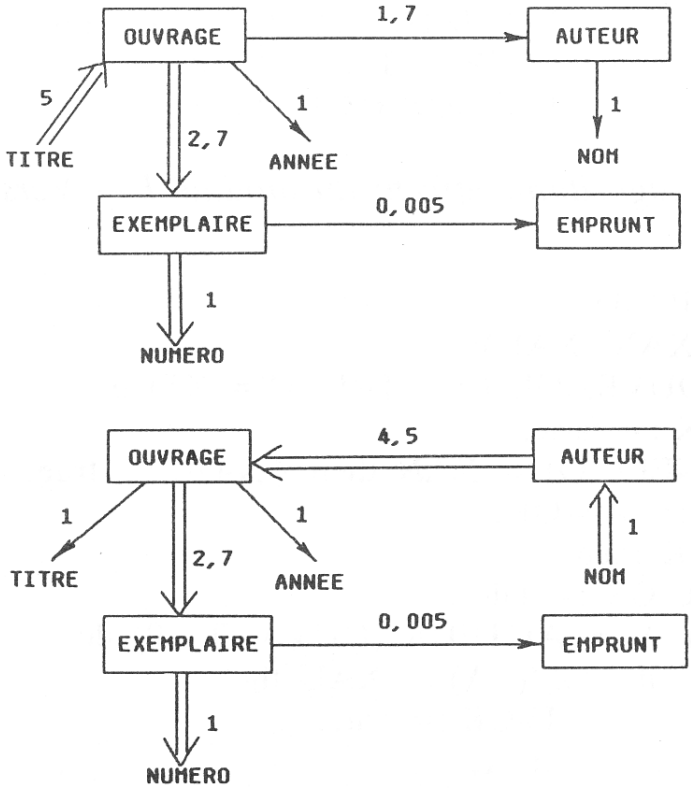


Figure 9.4 - Sous-schéma des accès des algorithmes effectifs du module 1

Nous développerons ces deux algorithmes conformément à ces arbres d'accès et nous en évaluerons le coût.

ALGORITHME EFFECTIF DU MODULE 1 - Première proposition

En séparant boucles d'accès évaluables et filtres, on obtient d'abord l'algorithme intermédiaire 9.4. On y dénote par =S= la relation de similitude déjà évoquée. Les conditions des alternatives ne sont cependant pas encore évaluables par accès. L'algorithme effectif 9.5. est obtenu suite à leur développement.


```
for B := BIBLIO do
    input(XT, XAN, XAU);
    for O := OUVRAGE (: TITRE =S= XT) do
        if O((: ANNEE = XAN) and (OUVAUT: AUTEUR(: NOM = XAU))) then
           for EX := EXEMPLAIRE (DE: O) do
               if EX(EXE: 0 EMPRUNT) then
                  print NUMERO(:EX);
                  exit O;
               endif;
           endfor EX;
        endif;
    endfor O;
endfor B;
```

Alg 9.4. Premier algorithme effectif du module 1 - Version interm{diaire

```
for B := BIBLIO do
    input(XT, XAN, XAU);
    for O := OUVRAGE (: TITRE =S= XT) do
        AN-OK := false;
        if ANNEE(:O) = XAN then AN-OK := true; endif;
        O-OK := AN-OK;
        if O-OK then
           AU-OK := false;
           for A := AUTEUR (OUVAUT: 0) do
               if NOM(:A) = XAU then
                       AU-OK := true;
                       exit A;
               endif;
           endfor;
           O-OK := AU-OK;
        endif;
        if O-OK then
           for EX := EXEMPLAIRE (DE: O) do
               EX-OK := true;
               for EM := EMPRUNT (EXE: EX) do
                   EX-OK := false;
               endfor;
               if EX-OK then
                  print NUMERO(:EX);
                  exit O;
               endif;
           endfor EX;
        endif;
```

```
    endfor O;
endfor B;
```

Alg 9.5. Premier algorithme effectif du module 1

ALGORITHME EFFECTIF DU MODULE 1 - Seconde proposition

Un premier développement, basé sur la seconde arborescence du schéma 9.4, conduit à l'algorithme intermédiaire 9.6. Par développement des filtres non évaluables par accès, on obtient l'algorithme effectif 9.7.

```
for B := BIBLIO do
    input(XT, XAN, XAU);
    for AU := AUTEUR (: NOM = XAU) do
        for O := OUVRAGE (OUVAUT: AU) do
            if O ((: TITRE =S= XT) and (: ANNEE = XAN)) then
               for EX := EXEMPLAIRE (DE: O) do
                  if EX(EXE: 0 EMPRUNT) then
                     print NUMERO(:EX);
                     exit AU;
        endif;
               endfor EX;
            endif;
        endfor O;
    endfor AU;
endfor B;
```

Alg 9.6 Second algorithme effectif du module 1 - Version intermédiaire

```
for B := BIBLIO do
    input(XT, XAN, XAU);
    for AU := AUTEUR (: NOM = XAU) do
        for O := OUVRAGE (OUVAUT: AU) do
            if (TITRE(:O) =S= XT) and (ANNEE(:O) = XAN) then
               for EX := EXEMPLAIRE (DE: O) do
                  EX-OK := true;
                  for EM := EMPRUNT (EXE: EX) do
                      EX-OK := false;
                  endfor;
                  if EX-OK then
                     print NUMERO(:EX);
                     exit AU;
                  endif;
               endfor EX;
```



```
            endif;
         endfor O;
      endfor AU;
endfor B;
```

Alg 9.7 Second algorithme effectif du module 1

EVALUATION DES DEUX ALGORITHMES CANDIDATS

PREMIER ALGORITHME (alg. 9.5.)

Supposons que pour une valeur de TITRE, l'accès phonétique produise en moyenne 15 OUVRAGES. Dans 10% des cas, (voir quantification), l'ouvrage est inconnu et l'on accède aux 15 articles. Dans 90% des cas, l'article est trouvé et l'on a réalisé en moyenne 8 accès. Par conséquent, une exécution de la boucle "for O" accède en moyenne à 8,7 articles OUVRAGE.

Examinons à présent le coût de l'évaluation de la condition sur O. La vérification de ANNEE est de coût nul et réduit à un seul la liste des 8,7 OUVRAGES sélectionnés (on a admis une sélectivité de 1/40 pour l'ANNEE, mais dans 90% des cas, l'OUVRAGE cherché est dans cette liste; ce nombre de 1 est donc réaliste). Pour cet OUVRAGE restant, on accède aux AUTEURS. Il y a en moyenne 2 AUTEURS par OUVRAGE ayant un AUTEUR, soit un parcours moyen de 1,5 AUTEUR. L'accès et le filtre laissent passer 0,9 OUVRAGE (90% de réussite).

Il y a en moyenne 2,7 EXEMPLAIRES par OUVRAGE. La probabilité d'acceptation associée au filtre sur EXEMPLAIRE est de 1-2000/420000 (elle est en fait inférieure car un exemplaire emprunté a plus de chance d'être demandé qu'un autre); en général, le premier exemplaire sera donc disponible. On comptabilisera 1 accès à EMPRUNT pour l'évaluation du filtre.

Le bilan de l'algorithme 9.5 s'établit alors comme suit :

OUVRAGE (: TITRE =S= X) : 1 séquence de 8,7 OUVRAGES
AUTEUR (OUVAUT: OUVRAGE) : 1 séquence de 1,5 AUTEUR
EXEMPLAIRE (DE: OUVRAGE) : 0,9 séquence de 1 EXEMPLAIRE
EMPRUNT (EXE: EXEMPLAIRE) : 0,9 séquence de 1 EMPRUNT

Il y a donc globalement 12 accès logiques en moyenne.

DEUXIEME ALGORITHME (alg 9.7)

On admet qu'à une valeur de NOM d'AUTEUR correspondent 3 AUTEURS en moyenne. En cas de réussite, on accédera donc en moyenne à 2 AUTEURS et en cas d'échec à 3 AUTEURS, soit, à raison de 10% d'échecs, une moyenne générale de 2,1.

En cas de réussite, il y a 5 OUVRAGES par AUTEUR. Des 2 AUTEURS sélectionnés, le premier conduit à 5 OUVRAGE et le second s'arrête à mi-chemin, soit 3. Soit une moyenne de 4 OUVRAGES par AUTEUR. En cas d'échec chaque AUTEUR conduit à 4,5 OUVRAGES, soit une moyenne générale de 2,1 séquences de 4 OUVRAGES.

L'évaluation du filtre sur ANNEE et TITRE est de coût nul et produit 0,9 OUVRAGES. La suite de l'algorithme est semblable à celle de l'algorithme précédent.

On peut alors établir le bilan de l'algorithme 9.7

AUTEUR (: NOM=X) : 1 séquence de 2,1 AUTEURS
OUVRAGE (OUVANT: AUTEUR) : 2,1 séquences de 4 OUVRAGES
EXEMPLAIRE (DE: OUVRAGE) : 0,9 séquence de 1 EXEMPLAIRE
EMPRUNT (EXE: EXEMPLAIRE) : 0,9 séquence de 1 EMPRUNT

Il y a donc globalement 12,1 accès logiques en moyenne.

CHOIX DE L'ALGORITHME EFFECTIF DU MODULE 1

L'évaluation des coûts moyens probables montre que les deux algorithmes sont sensiblement équivalents sur ce point. Le premier algorithme exige une technique d'accès phonétique qui impliquera probablement la définition d'un type d'articles supplémentaires (index phonétique), à moins que le SGBD n'offre précisément ce type de sélection (ORACLE p. ex.) et que ce dernier soit applicable ici. On l'écartera donc au profit du second, qui se contente d'une clé d'accès traditionnelle (NOM d'AUTEUR).

ALGORITHME EFFECTIF DU MODULE 2

Dans l'algorithme 9.2, ni la condition de la boucle "for E" ni les deux premières conditions du CREATE ne sont évaluables par accès. Leur développement conduit à l'algorithme 9.8, qui est le seul que nous envisagerons.

```
for B := BIBLIO do
    input(XEX);
    begin-transaction;
      for EX := EXEMPLAIRE (: NUMERO = XEX) do
          for E := EMPRUNT (EXE: EX) do
              EMP := EMPRUNTEUR (EMPE: E);
              create EA := EMPRUNT-ARCH ( (EXEA: EX) and
```



```
                                                    (EMPEA: EMP) and
                                          (: DATE-DEBUT = DATE-DEBUT(:E)) and
                                               (: DATE-FIN = DATE-DU-JOUR));
                delete E;
            endfor;
        endfor;
    close-transaction;
endfor;
```

Alg 9.8. Algorithme effectif du module 2

On notera que si la deuxième condition du CREATE a conduit au développement de l'accès explicite à EMPRUNTEUR (EMP), il n'en est pas de même pour la première, car l'EXEMPLAIRE de E est disponible à cet endroit. Il s'agit d'une optimisation globale qui se justifie par le fait que (simplification 1, en 5.3.10) :

pour tout EX de EXEMPLAIRE (EXE: EMPRUNT) :

EXEMPLAIRE (EXE: EMPRUNT (EXE : EX)) = EX

Le schéma 9.5 représente sous-schéma des accès de cet algorithme.

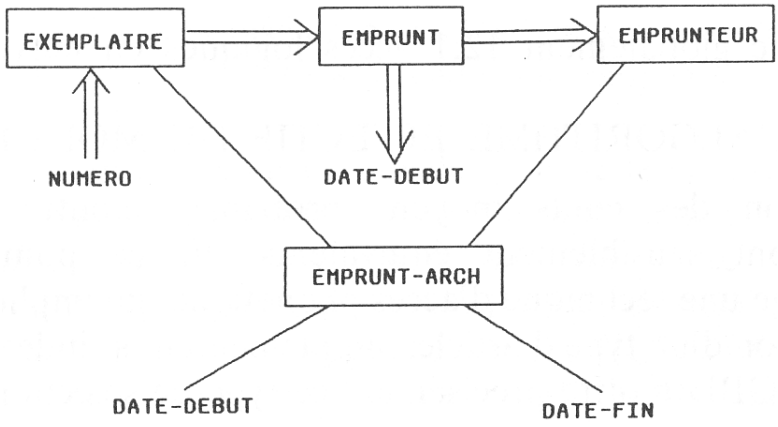


Figure 9.5 - Sous-schéma des accès de l'algorithme effectif du module 2

Evaluons le coût de cet algorithme :

EXEMPLAIRE (:NUMERO=X) : 1 séquence d'1 EXEMPLAIRE
EMPRUNT (EXE: EXEMPLAIRE) : 1 séquence d'1 EMPRUNT
EMPRUNTEUR (EMPE: EMPRUNT) : 1 séquence d'1 EMPRUNTEUR

create EMPRUNT-ARCH : 1 article
delete EMPRUNT : 1 article

Le bilan global est de 3 accès logiques, 1 création logique et 1 suppression logique.

ALGORITHME EFFECTIF DU MODULE 3

Nous proposerons un algorithme effectif (Alg 9.9) qui fait usage d'un accès par clé à EMPRUNT. On aurait pu décider de réaliser un accès séquentiel filtré étant donné le taux de 10% d'EMPRUNTS à retenir (voir quantification). Nous tenterons cependant de reporter plus loin cette décision. Le sous-schéma d'accès de l'algorithme est représenté par le schéma 9.6.

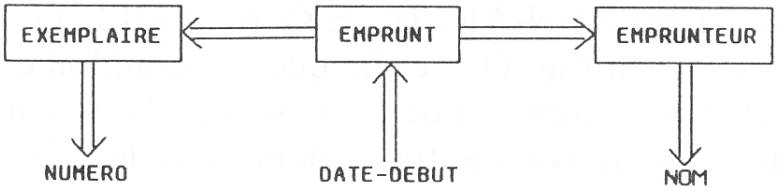


Figure 9.6 - Sous-schéma des accès de l'algorithme effectif du module 3

```
for B := BIBLIO do
    DATE := MOINS (DATE-DU-JOUR, 14);
    for E := EMPRUNT (:DATE-DEBUT < DATE) do
        EX := EXEMPLAIRE (EXE: E);
        EMP := EMPRUNTEUR (EMPE: E);
        print NUMERO(: EX), NOM(:EMP);
    endfor;
endfor;
```

Alg 9.9. Algorithme effectif du module 3

Il reste à en évaluer le coût.

EMPRUNT (: DATE-DEBUT < X) : 1 séquence de 200 EMPRUNTS
EXEMPLAIRE (EXE: EMPRUNT) : 200 séquences de 1 EXEMPLAIRE
EMPRUNTEUR (EMPE: EMPRUNT) : 200 séquences de 1 EMPRUNTEUR

Soit un bilan global de 600 accès logiques.

9.3.4 DERIVATION DU SCHEMA DES ACCES NECESSAIRES

Nous établirons un relevé, par type d'articles, et pour chaque unité de traitement (essentiellement un module à ce niveau), des primitives utilisées. Pour chaque primitive nous reprendrons en outre le nombre d'activations par unité de temps (par jour en l'occurrence), noté NA/J, le nombre moyen d'éléments concernés par activation, noté NE/A, et nous en déduirons le nombre d'éléments traités par jour, noté NE/J. Le tableau 9.1 est relatif au module 1, le tableau 9.2 est une synthèse relative aux trois modules analysés (pour simplifier, les primitives d'accès aux items n'y sont plus reprises) et le tableau 9.3 serait un tableau complet correspondant à tous les modules de l'application. Le tableau 9.4 est un résumé obtenu par totalisation des accès par type d'articles et par classe de primitives; les valeurs expriment le nombre d'articles concernés par jour. Il permet accessoirement une première évaluation très grossière de la charge en entrées/sorties : en admettant un coût empirique pessimiste de 50 msec par accès logique, 120 msec par création et 150 msec par suppression, on obtient une charge journalière de 1200 secondes. Le schéma 9.7 constitue quant à lui la synthèse graphique des accès utilisés par les algorithmes effectifs, il représente donc le SCHEMA DES ACCES NECESSAIRES. On y observe que le complexe (EMPRUN-ARCH, EXEA, EMPEA) fait l'objet de créations mais n'est pas utilisé. On vérifie que sa définition est confirmée au niveau conceptuel et qu'il convient de le conserver. Le schéma 9.8 permet une visualisation comparative du trafic dont sont le siège les différents mécanismes d'accès.

Primitives	NA/J	NE/A	NE/J
AUTEUR(:NOM=X)	200	2,1	420
OUVRAGE(OUVAUT:AUTEUR)	420	4	1680
TITRE(:OUVRAGE)	1680	1	1680
ANNEE(:OUVRAGE)	1680	1	1680
EXEMPLAIRE(DE:OUVRAGE)	180	1	180
NUMERO(:EXEMPLAIRE)	180	1	180
EMPRUNT(EXE:EXEMPLAIRE)	180	1	180

Tableau 9.1 : Ventilation et quantification des primitives du module 1

Primitives	NA/J	NE/A	NE/J
AUTEUR(:NOM=X)	200	2,1	420
OUVRAGE(OUVAUT:AUTEUR)	420	4	1680
EXEMPLAIRE(:NUMERO=X)	400	1	400
EXEMPLAIRE(DE:OUVRAGE)	180	1	180
EXEMPLAIRE(EXE:EMPRUNT)	200	1	200
EMPRUNT(EXE:EXEMPLAIRE)	580	1	580
EMPRUNT(:DATE-DEBUT < X)	1	200	200
delete EMPRUNT	400	1	400
create EMPRUNT-ARCH	400	1	400
EMPRUNTEUR(EMPE:EMPRUNT)	600	1	600

Tableau 9.2 Ventilation et quantification des primitives des modules 1, 2 et 3

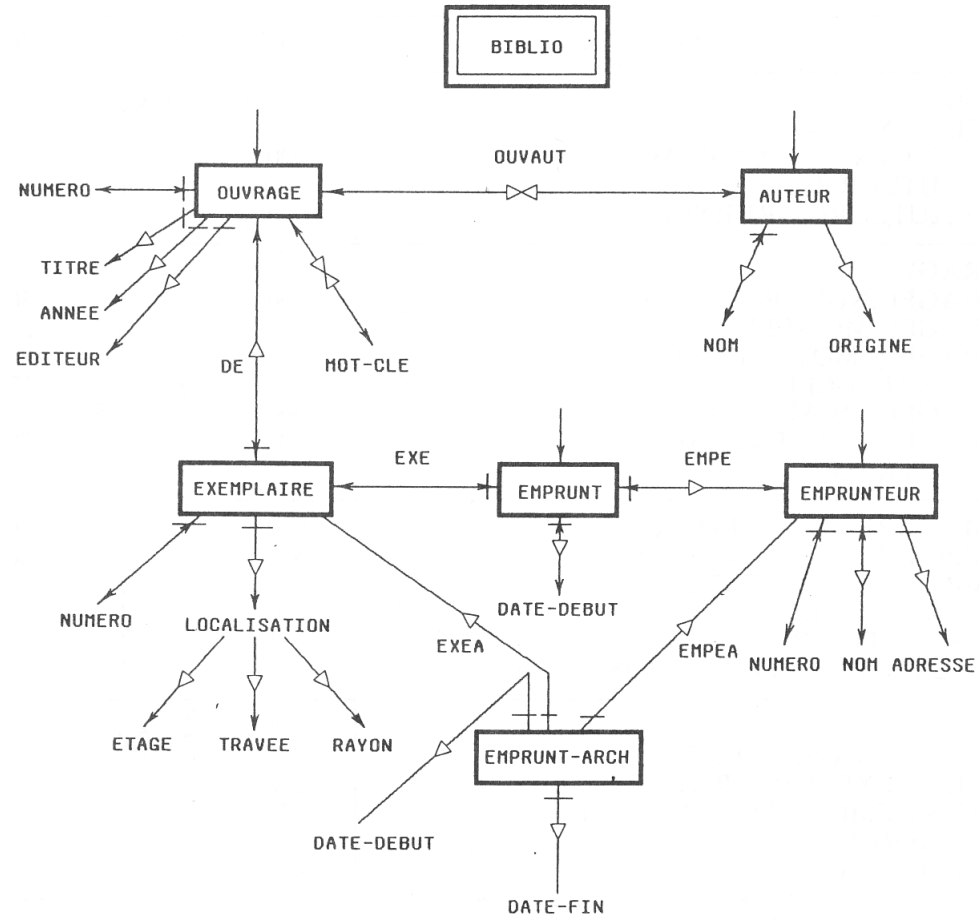
Primitives	NA/J	NE/A	NE/J
AUTEUR	0,05	60000	300
AUTEUR(:NOM=X)	600	2	1200
AUTEUR(OUVAUT:OUVRAGE)	2000	1,7	3400
create AUTEUR	15	1	15
modify AUTEUR(:ORIGINE)	0,5	1	0,5
OUVRAGE	0,01	160000	1600
OUVRAGE(:NUMERO=X)	300	1	300

OUVRAGE(:MOT-CLE=X)	17	16	270
OUVRAGE((:MOT-CLE=X1) and (:MOT-CLE=X2))	20	6	120
OUVRAGE(OUVAUT:AUTEUR)	600	4,5	2700
OUVRAGE(DE:EXEMPLAIRE)	700	1	700
create OUVRAGE	20	1	20
EXEMPLAIRE(:NUMERO=X)	1000	1	1000
EXEMPLAIRE(DE:OUVRAGE)	1800	2	3600
EXEMPLAIRE(EXE:EMPRUNT)	350	1	350
create EXEMPLAIRE	56	1	56
delete EXEMPLAIRE	1	1	1
modify EXEMPLAIRE(:LOCALISATION)	5	1	5
EMPRUNT	0,5	2000	1000
EMPRUNT(:DATE-DEBUT < X)	1	200	200
EMPRUNT(EXE:EXEMPLAIRE)	800	1	800
EMPRUNT(EMPE:EMPRUNTEUR)	25	1	25
create EMPRUNT	400	1	400
delete EMPRUNT	400	1	400
create EMPRUNT-ARCH	400	1	400
EMPRUNTEUR	1	2000	2000
EMPRUNTEUR(:NUMERO=X)	50	1	50
EMPRUNTEUR(:NOM=X)	10	1,05	10,5
EMPRUNTEUR(EMPE:EMPRUNT)	600	1	600
create EMPRUNTEUR	1	1	1
modify EMPRUNTEUR(:ADRESSE)	0,1	1	0,1

Tableau 9.3 : Ventilation et quantification des primitives pour l'application complète

Type d'articles	Accès	Création	Suppress.	Modif.
AUTEUR	4900	15	-	0,5
OUVRAGE	5690	20	-	-
EXEMPLAIRE	4950	56	1	5
EMPRUNT	2025	400	400	-
EMPRUNT-ARCH	-	400	-	-
EMPRUNTEUR	2660	1	-	0,1
TOTAL	20225	892	401	5,6

Tableau 9.4 : Synthèse de la quantification des primitives pour l'application



Contrainte supplémentaire : voir schéma 9.2

Figure 9.7 - Schéma des accès nécessaires de la BD BIBLIO

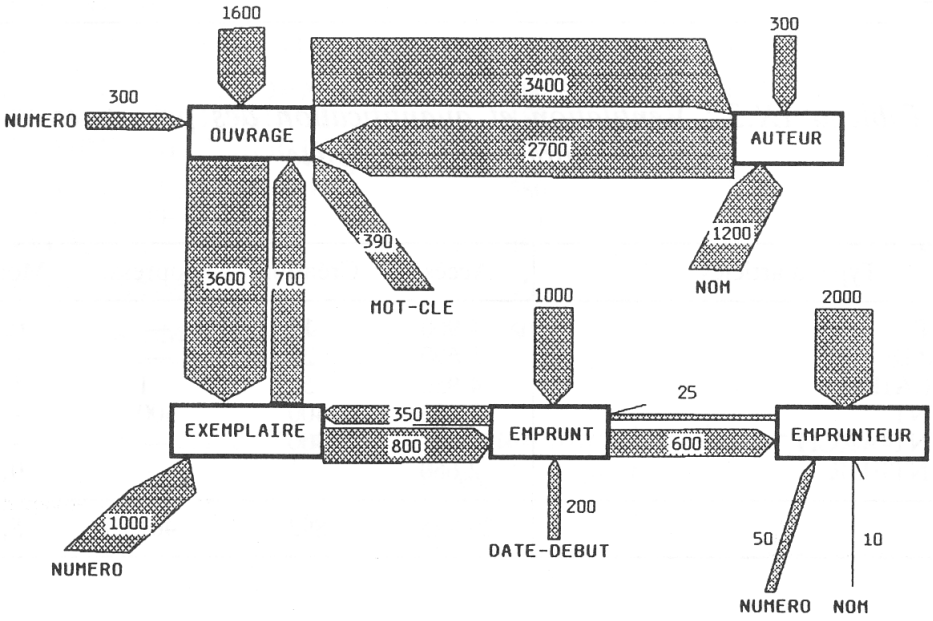


Figure 9.8 - Intensité du trafic d'accès pour toute l'application

Manque 9.2

9.4 CONCEPTION PHYSIQUE COBOL/CODASYL

9.4.1 PRODUCTION D'UN SCHEMA CONFORME A CODASYL

Si l'on s'en tient strictement à la conformité à CODASYL 71/73, on relève dans le schéma des accès nécessaire trois structures non conformes : les types de chemins inverses N-N OUVAUT, la clé d'accès répétitive non identifiante MOT-CLE ainsi qu'un couple de clés d'accès (NUMERO et NOM) associés au même type d'articles EMPRUNTEUR.

Type de chemins OUVAUT : l'élimination de OUVAUT selon la transformation 7.1 (Section 7.3.1) conduit à la définition du type d'articles OUVAUT et des types de chemins OO et AO, dotés chacun de leur inverse. La contrainte d'identifiant portant sur OUVAUT sera plus efficacement gérée par module d'accès que par le SGBD.

Clé d'accès MOT-CLE : on appliquera la transformation 7.1 à l'accès de MOT-CLE vers OUVRAGE mais pas à son inverse. En effet, un nombre variable, mais limité à 10, de valeurs de MOT-CLE est tout-à-fait tolérable. L'item MOT-CLE n'est plus clé d'accès et on introduit un nouveau type d'articles MOT-CLE-D-OUV dont l'item MOT-CLE est clé d'accès,

ainsi que le type de chemins OMC. Notons que par cette opération, nous délivrons OUVRAGE de sa deuxième clé d'accès. Le caractère variable du nombre de MOT-CLE d'un OUVRAGE est représenté par l'item compteur N-MOTS-CLES.

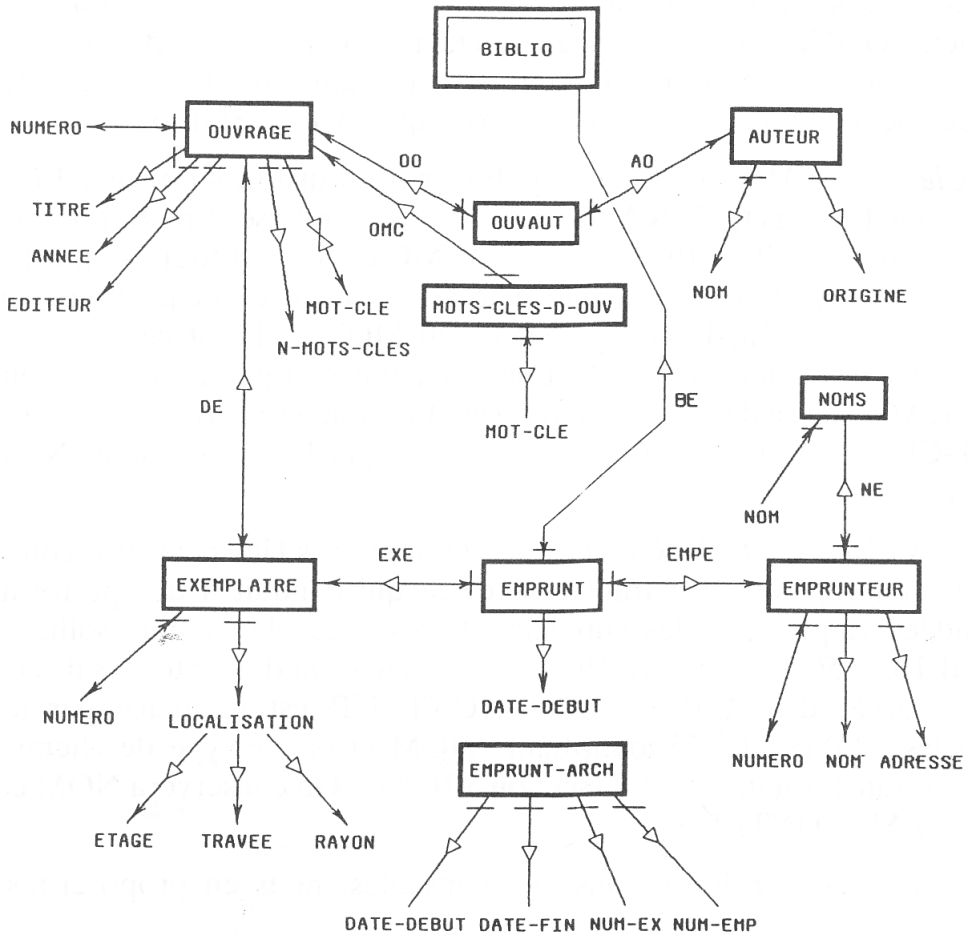
Clés d'accès d'EMPRUNTEUR : l'une de ces clés peut être confiée au SGBD; l'autre doit être transformée, ce qui conduit à des performances dégradées (type d'articles supplémentaires). La clé la plus sollicitée est NUMERO : 50 accès contre 10, outre la validation d'unicité lors des mises à jour. L'accès de NOM vers EMPRUNTEUR est remplacé par le type d'articles NOMS, de clé identifiante NOM et par le type de chemins NE selon la transformation 7.2 (section 7.3.1). On conservera NOM comme item d'EMPRUNTEUR.

Outre ces transformations indispensables, nous en proposerons deux autres.

1. Représentation d'EMPRUNT-ARCH : aucun accès ni modification n'ayant été prévus, on choisira une réalisation correcte mais minimale. Par rotation (transformation 7.3, section 7.3.2) on élimine les associations avec EXEMPLAIRE et EMPRUNTEUR pour les remplacer par NUM-EX (de même domaine que NUMERO d'EXEMPLAIRE) et NUM-EMP (de même domaine que NUMERO d'EMPRUNTEUR). On notera que les contraintes d'intégrité référentielles sur EMPRUNT-ARCH impliquent que l'on ne supprime pas d'EMPRUNTEUR ni d'EXEMPLAIRE sans suppression des EMPRUNT-ARCH correspondants.
2. Clé d'accès DATE-DEBUT de EMPRUNT : sans autre analyse, l'usage de cette clé suppose un accès trié via un index, dont le support en CODASYL est un SET singulier, d'OWNER SYSTEM et de nom BE, trié sur DATE-DEBUT et déclaré INDEXED. On observera cependant que les articles EMPRUNTS sont créés par ordre chronologique et donc par valeur croissante de DATE-EMPRUNT. Il suffit donc de déclarer pour ce SET un ordre d'insertion chronologique (LAST). Un simple parcours séquentiel permet d'obtenir sélectivement, et sans index, les EMPRUNTS en retard.

On pourrait encore suggérer des redondances d'efficacité telles que la reprise du NOM d'AUTEUR dans OUVAUT et celle de NUMERO D'EXEMPLAIRE et NUMERO D'EMPRUNTEUR dans EMPRUNT. Nous ne les retiendrons pas.

Ces décisions sont consignées dans le schéma 9.9.



Contraintes supplémentaires (autres que celle du schéma 9.7) :

1. (OUVRAGE,MOT-CLE) = OMC.(MOTS-CLES-D-OUV,MOT-CLE)
2. MOTS-CLE-D-OUV est identifié par MOT-CLE et OUVRAGE via OMC
3. (EMPRUNTEUR,NOM) = NE.(NOMS,NOM)
4. EMPRUNT-ARCH identifié par NUM-EX et DATE-DEBUT
5. NUM-EX(:EMPRUNT-ARCH) in NUMERO(:EXEMPLAIRE)
6. NUM-EMP(:EMPRUNT-ARCH) in NUMERO(:EMPRUNTEUR)
7. N-MOTS-CLES = compteur de MOT-CLE pour OUVRAGE
8. EXE(EXEMPLAIRE,EMPRUNT) est de classe fonctionnelle 1-1

9. BE en ordre chronologique

Figure 9.9 - Schéma de la base de données BIBLIO conforme à CODASYL

De nouveaux éléments de quantification doivent être définis :

- Nombre d'articles OUVAUT : 160.000 x 1.7 = 272.000
- Nombre d'articles MOT-CLE-D-OUV : 160.000 x 2 = 320.000
- Nombre d'articles NOMS : 1900 (taux de synonymie de 5%)
- Taille moyenne d'un chemin OO : 1,7 (15% de chemins vides)
- Taille moyenne d'un chemin AO : 4,5 (10% de chemins vides)
- Taille moyenne d'un chemin OMC : 2 (33% de chemins vides)
- Taille moyenne d'un chemin NE : 1,05
- Nombre moyen d'articles MOTS-CLE-D-OUV pour une valeur de MOT-CLE : 16

DECOUPE EN FICHIERS

Nous rangerons dans un seul fichier de nom ABIBLIO les types d'articles actifs (AUTEUR, OUVAUT, MOTS-CLE-D-OUV, OUVRAGE, EXEMPLAIRE, EMPRUNT, EMPRUNTEURS et NOMS) et dans le fichier ARCH-BIB les articles d'archive EMPRUNT-ARCH. Ces décisions pourraient être revues lors de la définition du schéma interne.

9.4.2 PRODUCTION D'ALGORITHMES CONFORMES A CODASYL

Par application des règles de transformation algorithmique énoncées dans la section 7.4.2 (et 7.4.1 pour le module 3) on obtient les algorithmes conformes 9.10 pour le module 1, 9.11 pour le module 2 et 9.12 pour le module 3. On y a également développé la rupture "exit AU" en ruptures élémentaires basées sur le booléen TROUVE.

```
for B := BIBLIO do
    input(XT, XAN, XAU);
    TROUVE := false;
    for AU := AUTEUR (: NOM = XAU) do
        for AO := OUVAUT (AO: AU) do
            for O := OUVRAGE (OO: AO) do
                if (TITRE(:O) =S= XT) and (ANNEE(:O) = XAN) then
                   for EX := EXEMPLAIRE (DE: O) do
                      EX-OK := true;
                      for EM := EMPRUNT (EXE: EX) do
                          EX-OK := false;
                      endfor;
                      if EX-OK then
                         TROUVE := true;
                         print NUMERO(:EX);
                      endif;
```

```
                    if TROUVE then exit EX; endif;
                  endfor EX;
               endif;
               if TROUVE then exit O; endif;
            endfor O;
            if TROUVE then exit AO; endif;

       endfor AO;
       if TROUVE then exit AU; endif;
    endfor AU;
endfor B;
```

Alg 9.10 Algorithme du module 1 conforme à CODASYL

```
for B := BIBLIO do
   input(XEX);
   begin-transaction;
      for EX := EXEMPLAIRE (: NUMERO = XEX) do
          for E := EMPRUNT (EXE: EX) do
              EMP := EMPRUNTEUR (EMPE: E);
             create EA := EMPRUNT-ARCH ( (:NUM-EX = NUMERO (:EX)) and
                                         (:NUM-EMP = NUMERO(:EMP)) and
                                  (: DATE-DEBUT = DATE-DEBUT(:E)) and
                                         (: DATE-FIN = DATE-DU-JOUR));
              delete E;
          endfor;
      endfor;
   close-transaction;
endfor;
```

Alg 9.11 Algorithme du module 2 conforme à CODASYL

```
for B := BIBLIO do
    DATE := MOINS (DATE-DU-JOUR, 14);
    for E := EMPRUNT do
        if DATE-DEBUT(:E) < DATE then
             EX := EXEMPLAIRE (EXE: E);
             EMP := EMPRUNTEUR (EMPE: E);
             print NUMERO(: EX), NUMERO(:EMP);
        else exit;
        endif;
    endfor;
endfor;
```


Alg 9.12 Algorithme du module 3 conforme à CODASYL

On associera également à ce schéma et à ces algorithmes le tableau des primitives et de leur quantification. Ce tableau est semblable au tableau 9.3. Les primitives d'accès à AUTEUR(OUVAUT:OUVRAGE), OUVRAGE(MOT-CLE), OUVRAGE(OUVAUT:AUTEUR), EMPRUNT(:DATE-DEBUT) et EMPRUNTEUR(:NOM) y disparaissent. Les autres modifications sont reprises dans le tableau 9.5.

Primitives	NA/J	NE/A	NE/J
OUVAUT(OO:OUVRAGE)	2000	1.7	3400
OUVAUT(AO:AUTEUR)	600	4.5	2700
create OUVAUT	34	1	34
AUTEUR(AO:OUVAUT)	3400	1	3400
OUVRAGE(OO:OUVAUT)	2700	1	2700
OUVRAGE(OMC:MOT-CLE-D-OUV)	590	1	590
MOT-CLE-D-OUV(:MOT-CLE=X)	37	16	590
create MOT-CLE-D-OUV	40	1	40
EMPRUNT(BE:BIBLIO)	1	200	200
NOMS(:NOM=X)	10	1	10
create NOMS	0.95	1	0.95
EMPRUNTEUR(NE:NOMS)	10	1.05	10.5

Tableau 9.5 : Ajouts au tableau 9.3 pour CODASYL

9.4.3 PRODUCTION DU SCHEMA CODASYL

On trouvera le texte du schéma en DDL CODASYL à l'annexe 1. Remarquons qu'en principe, nous n'introduisons encore aucun élément du schéma interne. En particulier, la clause LOCATION ne sera présente que lorsque l'on spécifiera une clé d'accès ou un identifiant. D'autre part, vu les structures algorithmiques proposées, il n'y a pas de raison de proposer d'autres clauses SET SELECTION que THRU CURRENT. Les clauses élaborées ici sont préfixées de la lettre L dans le texte de l'annexe.

9.4.4 DEFINITION DES SCHEMAS EXTERNES CODASYL

Un schéma externe est à définir pour une classe homogène de modules tels que ceux qui constituent une phase. Nous choisirons de matérialiser ce schéma externe par un module d'accès. Les données sont perçues par ce dernier au travers d'un subschema. Le schéma DDL de l'annexe 1 est suivi du subschema du module d'accès que nous allons développer pour le module 1. Il conviendrait d'y ajouter tous les autres subschemas nécessaires à l'application. Les lignes correspondantes seront préfixées de la lettre E dans le texte de l'annexe.

9.4.5 LES PROGRAMMES COBOL/CODASYL

De manière à limiter notre propos aux caractéristiques de la programmation sur base de données, nous réaliserons chaque module sous la forme d'un programme COBOL isolé. Ce programme, du moins s'il accède à la base de données, fait appel à un SGD. Ce dernier est soit un SGD réel, auquel cas le programme est dit de niveau 1, soit un module d'accès (voir section 8.7). L'objectif de ce chapitre étant d'illustrer la démarche, nous traiterons uniquement le cas du module 1, selon les modalités suivantes :

- l'architecture est de niveau 3, c'est-à-dire que le programme dérive de l'agorithme effectif, et qu'il sera indépendant de tout SGBD;
- le programme utilise les services d'un module d'accès restreint à ses besoins;
- le module d'accès prend la forme d'un sous-programme COBOL.

Afin de rendre l'algorithme effectif 9.7 conforme aux structures algorithmiques du COBOL, nous développerons les boucles d'accès selon les primitives d'accès ponctuel et nous traduirons la rupture "exit AU" d'une manière similaire à celle adoptée dans l'algorithme 9.10 (pilotage par le booléen TROUVE). On obtient de la sorte l'algorithme 9.13, dont la traduction en COBOL est reportée à l'annexe 2.

```
OUVRIR-BIBLIO (OK-B);
if (OK-B = 0) then
    input(XT, XAN, XAU);
    TROUVE := false;
    AUTEUR-NOM-PREM (XAU, OK-AU);
    while (OK-AU = 0) do
        OUVRAGE-AUTEUR-PREM (OK-O);
        while (OK-O = 0) do
            TITRE-OUVRAGE (TITRE, OK);
            ANNEE-OUVRAGE (ANNEE, OK);
            COMPARER (TITRE, XT, SEMBLABLE);
            if (SEMBLABLE and ANNEE = XAN) then
               EXEMPLAIRE-DE-PREM (OK-EX);
               while (OK-EX = 0) do
                  EX-OK := true;
```



```
                    EMPRUNT-EXE (OK-EMP);
                    if (OK-EMP = 0) then
                        EX-OK := false;
                    endif;
                    if EX-OK then
                       TROUVE := true;
                       NUMERO-EXEMPLAIRE (NUMERO, OK);
                       print NUMERO;
                    endif;
                    if TROUVE then exit;
                       else EXEMPLAIRE-DE-SUIV (OK-EX);
                    endif;
                 endwhile;
              endif;

              if TROUVE then exit;
                 else OUVRAGE-AUTEUR-SUIV (OK-O);
              endif;
           endwhile;
           if TROUVE then exit;
              else AUTEUR-NOM-SUIV (OK-AU);
           endif;
       endwhile;
       FERMER-BIBLIO (OK-B);
endif;
```

Alg 9.13 Développement conforme au COBOL de l'algorithme effectif du module 1

On trouvera dans le document (HAINAUT,84) le développement complet d'un cas résolu en CODASYL selon les quatre architectures de référence.

9.4.6 CONSTRUCTION DU MODULE D'ACCES COBOL/CODASYL

Afin de limiter le développement, nous définirons et construirons un module d'accès extrêmement simplifié, tant du point de vue de sa spécification que de son fonctionnement interne. Les limitations de spécification concernent essentiellement les contraintes d'enchaînement des primitives (par exemple, une seule séquence active simultanément par type d'articles). Les simplifications internes sont dues surtout à l'absence de vérification de cet enchaînement, à l'absence de récupération en cas d'erreur, et à l'ignorance des problèmes de concurrence tant dans la base de données que dans l'appel du module (appel par plusieurs autres modules). Une première extension à envisager consisterait à transmettre au module la référence de l'article dont on désire le suivant ou les valeurs d'item.

La spécification minimum s'établit aisément à partir du tableau 9.1, ou de manière équivalente par analyse de l'agorithme 9.13 :

- OUVRIR-BIBLIO (OK) : rendre accessible au programme les parties de la base de données BIBLIO qui contiennent toutes les données nécessaires au module 1.
- FERMER-BIBLIO (OK) : déconnecter du programme les parties de la base de données qui ont été rendues accessibles par OUVRIR-BIBLIO.
- AUTEUR-NOM-PREM (N, OK) : repérer le premier article AUTEUR dont le nom est égal à N.
- AUTEUR-NOM-SUIV (OK) : repérer l'article AUTEUR qui suit le dernier article AUTEUR repéré, et dont la valeur de NOM est la même que celle de ce dernier.
- OUVRAGE-AUTEUR-PREM (OK) : repérer le premier article OUVRAGE du chemin OUVAUT dont l'origine est le dernier article AUTEUR repéré.
- OUVRAGE-AUTEUR-SUIV (OK) : repérer le suivant, dans son chemin OUVAUT, du dernier article OUVRAGE repéré.
- EXEMPLAIRE-DE-PREM (OK) : repérer le premier article EXEMPLAIRE du chemin DE dont l'origine est le dernier article OUVRAGE repéré.
- EXEMPLAIRE-DE-SUIV (OK) : repérer le suivant, dans son chemin DE, du dernier article EXEMPLAIRE repéré.
- EMPRUNT-EXE (OK) : repérer l'article EMPRUNT cible du chemin DE dont l'origine est le dernier article EXEMPLAIRE repéré.
- TITRE-OUVRAGE (N, OK) : renvoyer dans N la valeur de l'item TITRE du dernier article OUVRAGE repéré.
- ANNEE-OUVRAGE (A, OK) : renvoyer dans A la valeur de l'item ANNEE du dernier article OUVRAGE repéré.
- NUMERO-EXEMPLAIRE (N, OK) : renvoyer dans N la valeur de l'item NUMERO du dernier article EXEMPLAIRE repéré.

Dans toutes ces primitives, le code de diagnostic OK indique la manière dont les opérations se sont déroulées (0 signifiant un succès, 1 l'absence de l'objet demandé, la spécification des autres valeurs est ignorée).

Nous donnerons au module d'accès la forme concrète d'un sous-programme de nom BD, doté de trois arguments (simplification par rapport à la section 8.7) :

- CODE-OP : code de la primitive de 1 (OUVRIR-BIBLIO) à 12 (NUMERO-EXEMPLAIRE).
- CHAMP : variable de longueur suffisante assurant la transmission d'une valeur d'item, cadrée à gauche;
- OK : code de diagnostic.

La construction du module ne pose guère de problème si ce n'est l'accès à OUVRAGE à partir d'AUTEUR qui se traduit par la composition de AO et OO. Sur base de la technique suggérée au paragraphe 8.7.5., on obtient une solution conforme à COBOL/CODASYL. Le dévelop-

pement correspondant est présenté en alg. 9.14. Le sous-programme COBOL est reporté à l'annexe 3. Sa forme peut varier en fonction des SGBD en raison de la forme même de sous-programme qui peut interférer avec les liaisons du DBCS.

```
OUVR-P:  for AO := OUVAUT (AO: AU) do
             for O := OUVRAGE (OO: AO) do
                 return (O);
OUVR-S:
             endfor O;
         endfor AO;
         return ();
```

```
OUVR-P:  OUVAUT-PREM (OK-AO);
         while (OK-AO = 0) do
             OUVRAGE (OK-O);
             if (OK-O = 0) then
                return;
             endif;
OUVR-S:
             OUVAUT-SUIV (OK-AO);
         endwhile;
         OK := 1;
         return;
```

```
OUVR-P:  OUVAUT-PREM (OK-AO);
TEST-AO: if not (OK-AO = 0) then goto FIN-AO; endif;
             OUVRAGE (OK-O);
             return;
OUVR-S:
             OUVAUT-SUIV (OK-AO);
         goto TEST-AO;
FIN-AO:  OK := 1;
         return;
```

Alg 9.14. Phases de développement de l'accès à OUVRAGE dans le module d'accès du module 1 (CODASYL)

9.4.7 DEFINITION DU SCHEMA INTERNE

Nous définirons les paramètres des structures de stockage ainsi que certains paramètres de fonctionnement du SGBD. Ces paramètres sont à spécifier comme complément du schéma DDL et sous la forme du schéma DMCL. Cette répartition est typique des SGBD CODASYL 71/73, et en particulier de DBMS-20 de DEC. Les spécifications correspondantes seront préfixées de la lettre P dans le texte de l'annexe.

Au niveau du schéma DDL : on utilisera la clause LOCATION MODE pour définir des agrégats physiques liés aux SETS (VIA) ou programmés (DIRECT); la clause SET MODE définit la technique d'implantation d'un SET (ce paramètre est figé en DBMS-20); dans les SETS toujours, on définira des pointeurs arrière et des pointeurs vers l'OWNER; on fixera enfin les clauses ORDER non encore spécifiées.

Dans le schéma DMCL (Device Media Control Language) : on définira essentiellement :

- la liaison AREA/fichier externe;
- la taille théorique des AREAS;
- la taille des pages de chaque AREA;
- une partition de chaque AREA en sous-espaces, et l'affectation des types d'articles à ces sous-espaces;
- le nombre de chaînes dans lesquelles sont répartis les synonymes d'une même page lors du rangement calculé;
- la taille du tampon de chaque AREA.

Les autres paramètres seront ignorés.

PARAMETRES PHYSIQUES DU SCHEMA DDL

A. Définition d'agrégats physiques pour EMPRUNT, EMPRUNT-ARCH et OUVAUT.

EMPRUNT : on provoque un rangement contigu par un LOCATION MODE VIA BE et par un SET ORDER LAST.

EMPRUNT-ARCH : on provoque un rangement contigu par un LOCATION MODE DIRECT et en faisant précéder toute création d'un FIND LAST OF ARCHBIB AREA.

OUVAUT : des agrégats de SET peuvent être créés autour de AUTEUR ou de OUVRAGE. Les performances des deux solutions sont identiques car OUVAUT n'est qu'une étape entre OUVRAGE et AUTEUR. Nous choisirons un LOCATION MODE VIA OO afin de créer des agréagats (OUVRAGE,OUVAUT).

B. Définition des pointeurs arrière dans les SETS.

BE : SET singulier de grande taille et soumis à des suppressions aléatoires fréquentes, d'où LINKED TO PRIOR.

Autres : taille faible et/ou suppression très rares. Pas de pointeur arrière.

C. Définition des pointeurs vers l'OWNER.

Ceux-ci sont utiles lorsque les SETS ne correspondent pas à des agrégats contenant l'OWNER et que le nombre moyen de MEMBERS est supérieur à 1. On spécifiera donc pour AO, OMC et DE, la clause LINKED TO OWNER.

D. Définition des clauses ORDER.

On admettra que les SETS EMPE soient rangés par ordre chronologique (ORDER LAST). Tous les autres ordres non encore déclarés étant indifférents, on les déclarera FIRST pour optimiser l'insertion et l'espace occupé.

PARAMETRES PHYSIQUES DU SCHEMA DMCL

L'un des principaux paramètres est la taille de l'espace accordé à chaque AREA, et la répartition des types d'articles dans les sous-espaces. Nous calculerons donc au préalable l'espace nécessaire à chaque type d'articles. Nous travaillerons à partir des hypothèses suivantes :

- taux de remplissage maximum de 85% pour les articles rangés en calculé. On a ainsi la garantie d'un temps d'accès par clé excellent : entre 1 (articles très courts) et 1.5 (articles longs) accès physiques par accès logique, pour une gestion des débordements en zone primaire.
- la base de données ne sera pas réorganisée avant 4 ans.
- on réservera un sous-espace à chaque type d'articles de l'AREA ABIBLIO, sauf pour les agrégats (OUVRAGE, OUVAUT).
- on adopte sans autre analyse des pages de 2048 caractères.
- DBMS-20 travaille par mots de 36 bits; on exprimera cependant toutes les mesures en caractères, un mot étant assimilé à 4 caractères.

A. Evaluation des volumes

Pour chaque type d'articles, on évaluera la longueur logique d'un article (LLA), sa longueur physique (LPA) (compte tenu des pointeurs et autres données de gestion), le nombre initial d'articles (NIA), le taux d'accroissement en 4 ans (%4), le nombre d'articles dans 4 ans (NA4), le nombre maximum d'articles par page (NAPP), ainsi que le nombre de pages nécessaires dans 4 ans (NP4).

Calculons par exemple ces grandeurs pour le type d'articles AUTEUR :

longueur logique d'un article : 85 car

longueur des données de gestion : 12 car. (header + pointeur CALC + pointeur FIRST)
longueur physique d'un article : 97 car
nombre initial d'articles : 60000
taux d'accroissement en 4 ans : 22%
nombre d'articles dans 4 ans : 73200
capacité d'une page : 20 articles
nombre de pages : 73200/20 = 3660 pages à 100%
3660/0,85 = 4306 pages à 85%

Le tableau 9.6 résume cette évaluation. Il appelle les remarques suivantes :

- la ligne OUVRAGE correspond en fait aux mesures relatives aux agrégats (OUVRAGE, OUVAUT) constitués en moyenne d'un article OUVRAGE (LPA=359 car) et de 1,7 articles OUVAUT (LPA=16);
- pour EMPRUNT, le paramètre NA4 est remplacé par un volume de pointe correspondant à 10 fois le volume en régime;
- pour EMPRUNT, l'analyse du nombre de pages est conduite comme suit : de par le mode de rangement les articles EMPRUNT en retard sont en tête de la séquence. Ils constituent 10% des articles et l'on admet que leurs pages sont remplies à 10%. Les 90% restant viennent ensuite avec un taux de remplissage des pages de 10 à 100% (moyenne 55%). Une page ayant une capacité à 100% de 93 articles, on calcule le nombre de pages en régime :

  200/(93x0,1) + 1800/(93 x 0,55) = 58 pages;

  En pointe, on ajoute 18000 articles dans des pages à 100%, pour atteindre un total de 252 pages;
- EMPRUNT-ARCH est chargé à 100%.

Type d'articles	LLA	LPA	NIA	%4	NA4	NAPP	NP4
AUTEUR	85	97	60.000	22%	73.800	20	4.306
OUVRAGE	339	387	160.000	11%	177.600	5	42.000
MOTS-CLES-D-OUV	20	36	320.000	11%	355.200	55	7.600
EXEMPLAIRE	20	40	420.000	11.6%	469.000	50	13.800
EMPRUNT	6	22	2.000	900%	20.000	93	252
EMPRUNTEUR	103	123	2.000	44%	2.880	16	212
NOMS	35	47	1.900	44%	2.736	42	78
EMPRUNT-ARCH	28	32	200.000	176%	552.000	63	8.762

Tableau 9.6 : Calcul des volumes de la base de données CODASYL

B. Taille des AREAS et sous-espaces

On déduit ainsi la taille des AREAS (FIRST PAGE et LASTPAGE) et les sous-espaces de répartition des articles de chaque type (RANGE) tels qu'ils sont exprimés dans le schéma CODASYL de l'annexe 1.

C. Nombre de chaînes de synonymes

Afin de réduire la recherche, on limitera à 10 le nombre moyen d'articles par chaîne. Le taux de remplissage le plus élevé est celui de MOT-CLE-D-OUV (55 articles par page maximum) qui détermine ainsi 6 chaînes par pages (CALC AT MOST).

D. Taille des tampons

Le principe adopté est de conserver en mémoire les pages qui risquent d'être réutilisées dans un proche avenir. Nous analyserons les besoins de chaque module puis nous déterminerons une taille globale.

- Besoins du module 1 : on conserve la page d'AUTEUR pendant l'accès à OUVAUT et OUVRAGE pour un accès éventuel au suivant; il faut donc offrir un espace suffisant aux complexes (OUVRAGE,OUVAUT) - 5 en moyenne s'il y en a au moins un - de chaque AUTEUR pour ne pas provoquer de défaut de page qui écraserait la page d'AUTEUR. On accorde 1 page à AUTEUR et 7 pages aux complexes, soit 8 pages au total. Quand l'OUVRAGE est trouvé, les 7 autres pages deviennent inutiles. Il faut ensuite une page dans laquelle défileront les EXEMPLAIRES et une page pour accueillir l'EMPRUNT éventuel. En résumé, 8 pages seront en général suffisantes pour le module 1.
- Besoins du module 2 : EMPRUNT, EXEMPLAIRE et EMPRUNTEUR doivent être conservés dans le tampon car ils font ensuite l'objet d'une mise à jour (3 pages au total). La suppression d'EMPRUNT entraîne un REMOVE dans EMPE, ce qui peut réclamer l'accès à 2 autres pages d'EMPRUNT. On comptera un total de 5 pages. La création de EMPRUNT-ARCH réclame 1 page.
- Besoins du module 3 : il faut garder en mémoire la page courante des EMPRUNTS à laquelle s'ajoutent 1 page d'EXEMPLAIRE et 1 page d'EMPRUNTEUR, d'où une taille nécessaire et suffisante de 3 pages.
- Besoins globaux : on fait l'hypothèse que le tampon est commun aux différents processus, et que la probabilité que ceux-ci réclament les mêmes données au même instant est

faible. On admet que le tampon doit satisfaire les besoins de 1 processus du module 1, de 3 processus du module 2 et de 1 processus du module 3. Il faut donc un tampon de 26 pages pour l'AREA ABIBLIO et un tampon d'une page pour l'AREA ARCHBIBLIO. Le calcul sera généralisé aux autres modules. Le raisonnement qui précède est pessimiste car il suppose que les processus consomment leur maxima au même instant.

9.4.8 CALCUL DES VOLUMES ET DES COUTS D'ACCES

Les volumes se calculent aisément à partir du tableau 9.6 : ils sont de l'ordre de 8800 pages pour ABCHBIB et 68.000 pages pour ABIBLIO. Ces valeurs sont des maxima. Il y correspond un volume global de 158 millions de caractères.

En ce qui concerne les coûts d'accès, ceux-ci se calculeront à partir des tableaux de quantification et ventilation des primitives élaborés en 9.3.4, et adaptés à CODASYL (tableau 9.5) en y associant le coût physique de chaque type d'accès logique. On admet que le nombre d'accès physiques nécessaires au parcours d'une séquence d'articles peut s'exprimer, si la séquence n'est pas vide, par :

$$c1 + cs \times (N-1)$$

où N est la taille de la séquence, c1 le nombre d'accès physiques nécessaires à l'obtention du premier article, et cs le nombre moyen d'accès physiques nécessaires à l'obtention d'un article suivant.

Bien qu'incorrecte du strict point de vue probabiliste, cette expression constitue ici une approximation largement suffisante en pratique.

Sur base des caractéristiques de stockage des données, de la répartition de celles-ci et de la taille des tampons, (Section 9.4.7), on calculera les paramètres (c1; cs) de chaque séquence utilisée par les algorithmes des modules. Pour calculer le temps d'accès, on considérera que le disque est fortement partagé, et que par conséquent tout accès physique est aléatoire, notamment en ce qui concerne le temps de positionnement du bras. Nous supposerons par exemple un temps moyen de 35 msec par accès physique. Nous négligerons le temps CPU des accès, ce qui, comme nous l'avons vu au chapitre 8, n'est pas toujours réaliste. Le tableau 9.7 est obtenu à partir du tableau des primitives des modules 1, 2 et 3 adapté au schéma conforme CODASYL qui serait obtenu à partir du tableau 9.2, corrigé selon la procédure adoptée pour le tableau 9.5. On y reprend, pour chaque primitive, le nombre d'activation par jour (NA/J), le nombre d'éléments (articles) par activation (NE/A), les coefficients d'accès physique (c1; cs), le nombre d'accès physiques par activation (NAP/A), le temps d'exécution d'une activation en msec (TE/A) et enfin le temps d'exécution par jour en msec (TE/J).

Pour être complet, on devrait aussi élaborer un tableau des temps pour toutes les primitives de l'application à partir des tableaux 9.3 et 9.5.

Primitive	NA/J	NE/A	(c1; cs)	NAP/A	TE/A	TE/J

AUTEUR(:NOM=X)	200	2,1	(1,3;0,2)	1.5	53	10.600
OUVAUT(AO:AUTEUR)	420	4	(1;1)	4	140	58.800
OUVRAGE(OO:OUVAUT)	1680	1	(0;0)	0	0	0
EXEMPLAIRE(:NUMERO=X)	400	1	(1;0)	1	35	14.000
EXEMPLAIRE(DE:OUVRAGE)	180	1	(1;1)	1	35	6.300
EXEMPLAIRE(EXE:EMPRUNT)	200	1	(1;0)	1	35	7.000
EMPRUNT(DE:BIBLIO)	1	200	(1;0,11)	23	800	800
EMPRUNT(EXE:EXEMPLAIRE)	580	1	(1;0)	1	35	20.300
delete EMPRUNT	400	1	4	4	140	56.000
create EMPRUNT-ARCH	400	1	2	2	70	28.000
EMPRUNTEUR(EMPE:EMPRUNT)	600	1	(1;0)	1	35	21.000
Total			226 sec			

Tableau 9.7 : Temps d'accès journaliers des modules 1, 2 et 3 (CODASYL)

Manque 9.4

9.5 CONCEPTION PHYSIQUE COBOL/COBOL

Le langage d'écriture des programmes est toujours COBOL tandis que le SGD est le système de gestion de fichiers du COBOL ANSI 74, qui offre les organisations séquentielle, séquentielle indexée et relative.

9.5.1 PRODUCTION D'UN SCHEMA CONFORME A COBOL

Le schéma des accès nécessaires (Schéma 9.7) contient des constructions qui ne sont pas compatibles avec les fichiers COBOL : l'absence d'identifiant d'AUTEUR, tous les types de chemins, ainsi que la clé d'accès répétitive MOT-CLE d'OUVRAGE.

Identifiant d'AUTEUR : Un tel item identifiant doit exister pour deux raisons. La première est le respect de la contrainte n° 6 relative au COBOL (Section 5.7.3). La seconde est que l'élimination des types de chemins par rotation exigera la présente d'un identifiant. On définit alors un item artificiel, obtenu par compostage par exemple, de nom NUM-AUT, identifiant, associé à AUTEUR.

Types de chemins OUVAUT : une solution traditionnelle consiste à éliminer d'abord le type N-N, puis, par rotation éliminant les types de chemins N-1, à définir un nouveau type d'arti-

cles intermédiaire. Cette solution étant illustrée en 9.6 pour la BD relationnelle, nous en choisirons une autre. On effectue une rotation de OUVAUT (OUVRAGE,AUTEUR) vers NUM-AUT, associant ainsi un item répétitif NUM-AUT à OUVRAGE (avec son compteur N-AUTEURS), puis une rotation de OUVAUT (AUTEUR,OUVRAGE) vers NUMERO d'OUVRAGE, ce qui définit symétriquement l'item NUM-OUV d'AUTEUR et son compteur N-OUV.

Types de chemins DE : par rotation, on définit l'item NUM-OUV d'EXEMPLAIRE. Cet item étant clé d'accès pour EXEMPLAIRE et NUMERO l'étant pour OUVRAGE, l'équivalence d'accès est également assurée.

Types d'associations EXEA et EMPEA : leur élimination par rotation est similaire à la solution CODASYL.

Types de chemins EXE : par rotation vers NUMERO d'EXEMPLAIRE, on définit l'item identifiant NUM-EX d'EMPRUNT. Ce dernier est une clé d'accès, de même que NUMERO l'est pour exemplaire, ce qui établit l'équivalence d'accès.

Types de chemins EMPE : une rotation vers NUMERO d'EMPRUNTEUR associe à EMPRUNT un item non identifiant NUM-EMP qui doit être une clé d'accès. On vérifie que NUMERO est une clé pour EMPRUNTEUR.

Clé répétitive MOT-CLE d'OUVRAGE : on conserve l'accès (OUVRAGE, MOT-CLE) mais on associe à cet item le compteur N-MOTS-CLES. Quant à l'accès (MOT-CLE, OUVRAGE), il est transformé selon la règle 7.1 (section 7.3.1), mettant ainsi en évidence le type d'articles MOT-CLES-D-OUV (cf solution CODASYL) doté d'un item clé d'accès non identifiant MOT-CLE. On élimine par rotation le type de chemins N-1 ainsi défini, ce qui associe à ce nouveau type d'articles l'item NUM-OUV, qui avec MOT-CLE constitue un identifiant. De manière à respecter les contraintes 5 et 6 du COBOL (voir section 5.7.3), on regroupe ces items pour constituer ID-MOT-CLE, clé d'accès identifiante et décomposable.

On exploite ici le fait que l'organisation sera séquentielle indexée, et donc que la clé d'accès est également clé de tri.

On propose, vu la petite taille de EMPRUNT, de ne pas réaliser de clé d'accès sur base de DATE-DEBUT, et d'effectuer tout accès sur ce critère par un parcours séquentiel filtré.

On pourrait encore suggérer d'autres transformations telles que la représentation d'EMPRUNT comme item décomposable d'EXEMPLAIRE, NUM-EMP devenant une clé d'accès non identifiante d'EXEMPLAIRE, ou encore l'établissement de redondances. Nous ne les retiendrons pas.

Le schéma conforme 9.10 représente graphiquement le résultat de ces transformations.

Les quantifications établies en 9.3.1 sont à modifier en considérant principalement les éléments suivants qui s'en déduisent :

- Nombre d'articles MOT-CLES-D-OUV : 320.000
- De 0 à 8 AUTEUR/NUM-OUV : moyenne 2 par NUM-OUV et 1,7 par NUMERO d'OUVRAGE

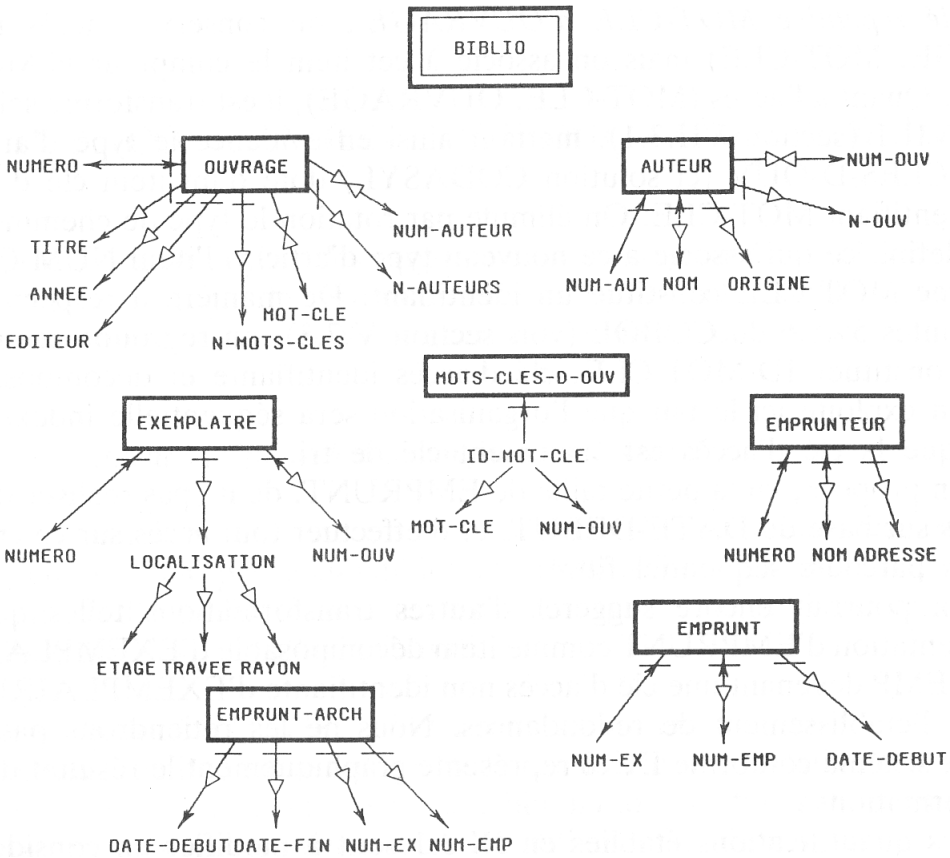
- De 0 à 15 OUVRAGE/NUM-AUT : moyenne 5 par NUM-AUTEUR et 4,5 par NUM-AUT

  d'AUTEUR
- De 0 à 20 EXEMPLAIRE/NUM-OUV : moyenne 3 par NUM-OUV et 2,7 par NUMERO d'OUVRAGE
- De 0 à 1 EMPRUNT/NUMERO d'EXEMPLAIRE, moyenne 0,005
- En moyenne 1 EMPRUNT par NUMERO d'EMPRUNTEUR, mais 2 EMPRUNT par NUM-EMP d'EMPRUNT
- En moyenne 2 MOT-CLE par OUVRAGE et 3 MOTS-CLES-D-OUV par NUM-OUV
- En moyenne 16 OUVRAGE par MOT-CLE et donc 16 MOTS-CLES-D-OUV par MOT-CLE

DECOUPE EN FICHIERS

Etant donné le petit nombre de types d'articles, on attribuera un fichier à chacun d'eux, appelés respectivement FAUT (AUTEUR), FOUVR (OUVRAGE), FMOT (MOT-CLE-D-OUV), FEX (EXEMPLAIRE), FEMPR (EMPRUNTEUR), FEMP (EMPRUNT) et FEMPA (EMPRUNT-ARCH).

9.5.2 PRODUCTION D'ALGORITHMES CONFORMES A COBOL



Contraintes supplémentaires :

1. (MOT-CLE,MOTS-CLES-D-OUV).(MOT-CLE-D-OUV,NUM-OUV) = (MOT-CLE,OUVRAGE).(OUVRAGE,NUMERO)
2. (NUMERO,OUVRAGE).(OUVRAGE,NUM-AUTEUR) = (NUM-OUV,AUTEUR).(AUTEUR,NUM-AUT)
3. N-MOTS-CLE = compteur de MOT-CLE pour OUVRAGE
4. N-AUTEURS = compteur de NUM-AUTEUR pour OUVRAGE
5. N-OUV = compteur de NUM-OUV pour AUTEUR
6. NUM-OUV(:EXEMPLAIRE) in NUMERO(:OUVRAGE)
7. NUM-EX(:EMPRUNT) in NUMERO(:EXEMPLAIRE)

8. NUM-EMP(:EMPRUNT) in NUMERO(:EMPRUNTEUR)

9. NUM-EX(:EMPRUNT-ARCH) in NUMERO(:EXEMPLAIRE)

10. NUM-EMP(:EMPRUNT-ARCH) in NUMERO(:EMPRUNTEUR)

11. Pour tout e d'EXEMPLAIRE et pour tout a d'EMPRUNT-ARCH(:NUM-EX=NUMERO(:e)):

DATE-DEBUT(EMPRUNT(:NUM-EX=NUMERO(:e))) DATE-FIN(:a)

12. EMPRUNT-ARCH est identifié par NUM-EX et DATE-DEBUT

Figure 9.10 - Schéma de la base de données BIBLIO conforme au COBOL

Par application des règles de transformation algorithmiques, on obtient l'algorithme 9.15 pour le module 1, 9.16 pour le module 2 et 9.17 pour le module 3. Outre le développement de la rupture "exit AU" comme dans l'algorithme 9.10 conforme à CODASYL, on y a réalisé le développement de :

```
for 0:=OUVRAGE (OUVAUT: AU) do
```

en :

```
for 0:=OUVRAGE (: NUMERO = NUM-OUV(:AU)) do
```

puis en structure conforme à l'algorithmique COBOL :

```
for IAO:=1 to N-OUV(:AU) do
     for 0:=OUVRAGE(:NUMERO = NUM-OUV[IAO](:AU)) do.
```

```
for B := BIBLIO do
    input(XT, XAN, XAU);
    TROUVE := false;
    for AU := AUTEUR (: NOM = XAU) do
        for IAO := 1 to N-OUV(:AU) do
            for O := OUVRAGE (:NUMERO=NUM-OUV[IAO](:AU)) do
                if (TITRE(:O) =S= XT) and (ANNEE(:O) = XAN) then
                   for EX := EXEMPLAIRE (:NUM-OUV=NUMERO(:O)) do
                      EX-OK := true;
                      for EM := EMPRUNT (:NUM-EX=NUMERO(:EX)) do
                          EX-OK := false;
                      endfor;
                      if EX-OK then
                         TROUVE := true;
                         print NUMERO(:EX);
                      endif;
                      if TROUVE then exit EX; endif;
                   endfor EX;
```

```
                endif;
                if TROUVE then exit O; endif;
            endfor O;
            if TROUVE then exit IAO; endif;
        endfor IAO;
        if TROUVE then exit AU; endif;
    endfor AU;
endfor B;
```

Alg 9.15 Algorithme du module 1 conforme au COBOL

```
for B := BIBLIO do
   input(XEX);
   begin-transaction;
      for EX := EXEMPLAIRE (: NUMERO = XEX) do
          for E := EMPRUNT (:NUM-EX=NUMERO(:EX)) do
              EMP := EMPRUNTEUR (:NUMERO=NUM-EMP(:E));
             create EA := EMPRUNT-ARCH ( (:NUM-EX = NUMERO (:EX)) and
                                         (:NUM-EMP = NUMERO(:EMP)) and
                                   (:DATE-DEBUT = DATE-DEBUT(:E)) and
                                         (:DATE-FIN = DATE-DU-JOUR));
              delete E;
          endfor;
      endfor;
   close-transaction;
endfor;
```

Alg 9.16 Algorithme du module 2 conforme au COBOL

```
for B := BIBLIO do
    DATE := MOINS (DATE-DU-JOUR, 14);
    for E := EMPRUNT do
        if DATE-DEBUT(:E) < DATE then
             EX  := EXEMPLAIRE (:NUMERO=NUM-EX(:E));
             EMP := EMPRUNTEUR (:NUMERO=NUM-EMP(:E));
             print NUMERO(: EX), NUMERO(:EMP);
        endif;
    endfor;
endfor;
```

Alg 9.17 Algorithme du module 3 conforme au COBOL

En fonction de ce schéma et de ces algorithmes, nous adapterons les tableaux 9.2 et 9.3 relatifs aux primitives et à leur quantification comme nous l'avons fait pour CODASYL (tableau correctif 9.5), le procédé étant analogue à ce qui a déjà été fait, nous ne le reprendrons pas ici.

9.5.3 PRODUCTION DU SCHEMA DES DONNEES COBOL

On trouvera à l'annexe 1 deux textes, l'un correspondant à l'ENVIRONNEMENT DIVISION et l'autre à la DATA DIVISION. Ces textes, encore incomplets du point de vue du COBOL, décrivent les caractéristiques logiques des fichiers de la base de données (préfixées par la lettre L). Ils seront stockés sous forme de deux fichiers qui seront insérés par des ordres COPY dans tout programme accédant aux données. Les organisations choisies (séquentielle indexée et séquentielle) découlent naturellement de la nature et du nombre des clés d'accès pour chaque type d'articles.

9.5.4 DEFINITION DES SCHEMAS EXTERNES

Ce concept est traditionnellement réalisé sous la forme de modules d'accès. Nous devons cependant compléter la description COBOL par les propriétés d'usage ACCESS MODE et FILE STATUS, qui dans l'annexe 1 sont préfixées de la lettre E.

En fait, si l'on désirait définir un schéma externe exclusivement réservé au module 1, il ne serait nécessaire de spécifier que les fichiers effectivement utilisés où l'on masquerait par des FILLERS les items non utilisés.

9.5.5 LES PROGRAMMES COBOL/COBOL

Pour les raisons qui ont été évoquées à l'occasion de la construction des programmes CODASYL, nous choisirons également de développer une architecture de niveau 3 pour le module 1. Il en résulte que le programme COBOL du module 1 est le même que pour la machine COBOL/CODASYL puisqu'il est indépendant du SGD. On le trouvera donc à l'annexe 2.

9.5.6 CONSTRUCTION DU MODULE D'ACCES COBOL/COBOL

La spécification du module d'accès est évidemment celle qui a été élaborée pour les programmes COBOL/CODASYL (section 9.4.6). Ici encore, la réalisation des primitives par des accès COBOL s'effectue aisément, sauf en ce qui concerne l'accès à OUVRAGE à partir d'AUTEUR. Les schémas algorithmiques 9.18 détaillent le procédé de construction des primitives correspondant selon des principes déjà appliqués à CODASYL. Le texte complet du module d'accès se trouve à l'annexe 3.

```
OUVR-P:  for IAO := 1 to N-OUV(:AU) do
             for O := OUVRAGE (: NUMERO=NUM-OUV[IAO](:AU)) do
                 return (O);
OUVR-S:
```

```
            endfor O;
         endfor AO;
         return ();

OUVR-P:  IAO := 1;
         while IAO <= N-OUV(:AU) do
            OUVRAGE-NUM (NUM-OUV[IAO](:AU),OK-O);
            if (OK-O = 0) then
               return;
            endif;
OUVR-S:
            IAO := IAO + 1;
         endwhile;
         OK := 1;
         return;

OUVR-P:   IAO := 1;
TEST-IAO: if not (IAO <= N-OUV(:AU)) then goto FIN-IAO; endif;
            OUVRAGE-NUM (NUM-OUV[IAO](:AU),OK-O);
            return;
OUVR-S:
            IAO := IAO + 1;
          goto TEST-IAO;
FIN-IAO:  OK := 1;
          return;
```

Alg 9.18 Phases du développement de l'accès à OUVRAGE dans le module d'accès 1 (fichiers COBOL).

9.5.7 DEFINITION DU SCHEMA INTERNE

Les paramètres du schéma internes sont à spécifier dans la description COBOL des fichiers (voir annexe 1, lignes préfixées par la lettre P), dans la description interne (Système d'exploitation) de ces fichiers ainsi que sous forme de caractéristiques d'exploitation.

Dans la description COBOL, nous spécifierons le nom externe des fichiers (clause ASSIGN), la taille des blocs physiques ou pages, que nous fixerons à 2048 caractères (clause BLOCK CONTAINS) ainsi que la taille des tampons pour chaque fichier (clause RESERVE AREA). Ce dernier paramètre est aussi fonction du programme car les tampons ne sont généralement pas centralisés en un pool.

Les autres caractéristiques que nous retiendrons sont la taille des fichiers (souvent inutile à spécifier en COBOL) et le taux de remplissage initial.

Nous ferons un certain nombre d'hypothèses sur le fonctionnement du séquentiel indexé :

- l'extension aléatoire s'effectue par division cellulaire avec répartition d'une page pleine sur deux pages;
- l'index primaire est du type B-TREE, chaque entrée d'index étant constituée d'une valeur de clé et d'un pointeur de page (4 car.); l'extension de cet index est gérée elle aussi par division cellulaire;
- un index secondaire est dense et utilise une technique de fichier inverse : à chaque valeur de clé est associée une liste de pointeurs (4 car.) accompagnée de sa longueur (2 car.); l'extension de ces index est gérée comme ci-dessus;
- à chaque article en longueur variable est associée une longueur (4 car.);
- un article est entièrement contenu dans une page;
- le taux de remplissage d'un index est celui du fichier de base.

A. Calcul des taux de remplissage

Ce taux conditionne le fonctionnement du fichier durant la première période d'utilisation. Dans un fichier en croissance pure (ou à très faible taux de suppression) on admet qu'un taux initial de 80% permet de conserver une moyenne de 80% de remplissage durant un accroissement de 15 à 20% au volume: le taux augmente, puis diminue pour se stabiliser autour de 70%. Ce taux sera adopté pour AUTEUR, OUVRAGE, MOTS-CLES-D-OUV, EXEMPLAIRE et EMPRUNTEUR. EMPRUNT est initialement vide puis est exploité dans un régime à 50% d'additions et 50% de suppressions. Ces conditions sont très défavorable pour cette organisation de fichiers car on démontre (et on observe) que le taux de remplissage tend alors vers zéro. Il sera donc nécessaire de prévoir une croissance importante en espace physique, de surveiller le volume et de réorganiser de temps à autre. Le facteur d'expansion de 10 prévu en CODASYL est ici pleinement justifié. On admettra un taux de remplissage moyen de 50%. Le fichier contenant EMPRUNT-ARCH est séquentiel, et donc rempli à 100%.

B. Evaluation des volumes

On trouvera dans le tableau 9.8 une synthèse de cette évaluation. Les grandeurs reprises sont les suivantes :

LPA : longueur physique moyenne d'un article (y compris zone longueur)

NIA : le nombre initial d'articles

NRP : nombre réel de pages du fichier de base (compte tenu du taux de remplissage moyen)

LI1 : longueur d'une entrée de l'index primaire

NI1 : nombre de niveaux de l'index primaire

VI1 : volume (en pages) de l'index primaire

LI2 : longueur d'une liste inversée de l'index secondaire

NI2 : nombre de niveaux de l'index secondaire
VI2 : volume (en pages) de l'index secondaire
VTOT : volume total (en pages) du fichier
VTOT4 : volume total (en pages) dans 4 ans.

Type d'articles	LPA	NIA	NRP	LI1	NI1	VI1	LI2	NI2	VI2	VTOT	TVOT4
AUTEUR	121	60000	4690	10	2	30	49	3	623	5350	6530
OUVRAGE	405	160000	40000	10	3	249	-	-	-	40250	44680
MOTS-CLES-D-OUV	26	320000	5130	30	3	98	-	-	-	5230	5805
EXEMPLAIRE	26	420000	6735	12	2	51	20	3	1742	8540	9540
EMPRUNT	22	2000	44	12	1	1	18	2	19	64	640
EMPRUNTEUR	103	2000	135	12	2	3	41	3	52	190	275
EMPRUNT-ARCH	28	200000	2740	-	-	-	-	-	-	2740	7570

Tableau 9.8 : Calcul des volumes des fichiers COBOL

C. Taille des tampons

On admet que le SGD alloue à chaque fichier séquentiel indexé un tampon spécifique à l'index (typiquement une page par niveau). Dans le cas du module 1, qui n'effectue que des accès aléatoires et pas de mise-à-jour, il ne faut qu'une page par fichier de base.

9.5.8 CALCUL DES VOLUMES ET DES COUTS D'ACCES

Les volumes ont été calculés dans le tableau 9.8. Le volume total dans 4 ans est de 75060 pages ou 153.750.000 caractères.

Le calcul des coûts d'accès suit le même raisonnement qu'en CODASYL. Il est présenté dans le tableau 9.9. On fera à ce propos deux remarques :

- comme en CODASYL, le coût d'ouverture et fermeture des fichiers n'est pas comptabilisé;
- le mode de fonctionnement, à savoir un accès aléatoire pour chaque activation du programme s'avère ici très défavorable. Le coût initial c1, très élevé, est à consentir intégralement pour chaque activation. En particulier, des accès aléatoires répétés pourraient profiter de la présence dans le tampon du premier niveau d'index (1 page) réduisant ainsi

d'une unité les coûts c1 des accès par clé. Une solution correspondrait à réaliser le module 1 sous la forme d'un processus permanent.

Primitive	NA/J	NE/A	(c1; cs)	NAP/A	TE/A	TE/J
AUTEUR(:NOM=X)	200	2.1	(4;1)	5,1	180	36000
OUVRAGE(:NUMERO=X)	1680	1	(4;0,025)	4	140	235200
EXEMPLAIRE(:NUMERO=X)	600	1	(4;0,016)	3	105	63000
EXEMPLAIRE(:NUM-OUV=X)	180	1	(4;1)	4	140	25200
EMPRUNT	1	2000	(1;0,022)	44	1540	1540
EMPRUNT(:NUM-EX=X)	580	1	(2;0,013)	2	70	40600
delete EMPRUNT	400	1	2,1	2,1	75	30000
create EMPRUNT-ARCH	400	1	2	2	70	28000
EMPRUNTEUR(:NUMERO=X)	600	1	(3;0,07)	3	105	63000
Total			550 sec			

Tableau 9.9 : Temps d'accès journaliers des modules 1, 2 et 3 (fichiers COBOL)

9.6 CONCEPTION PHYSIQUE COBOL/SQL

Le système SQL/DS disposant d'un processeur interactif de requête (ISQL), nous chercherons à traduire les algorithmes en ISQL, puis, si la chose n'est pas possible, en COBOL et SAL. Le système de gestion de données est bien entendu SQL/DS

Première partie : CONCEPTION BASEE SUR UNE OPTIMISATION IMPLICITE

Cette approche a été décrite en 8.6.3.2.

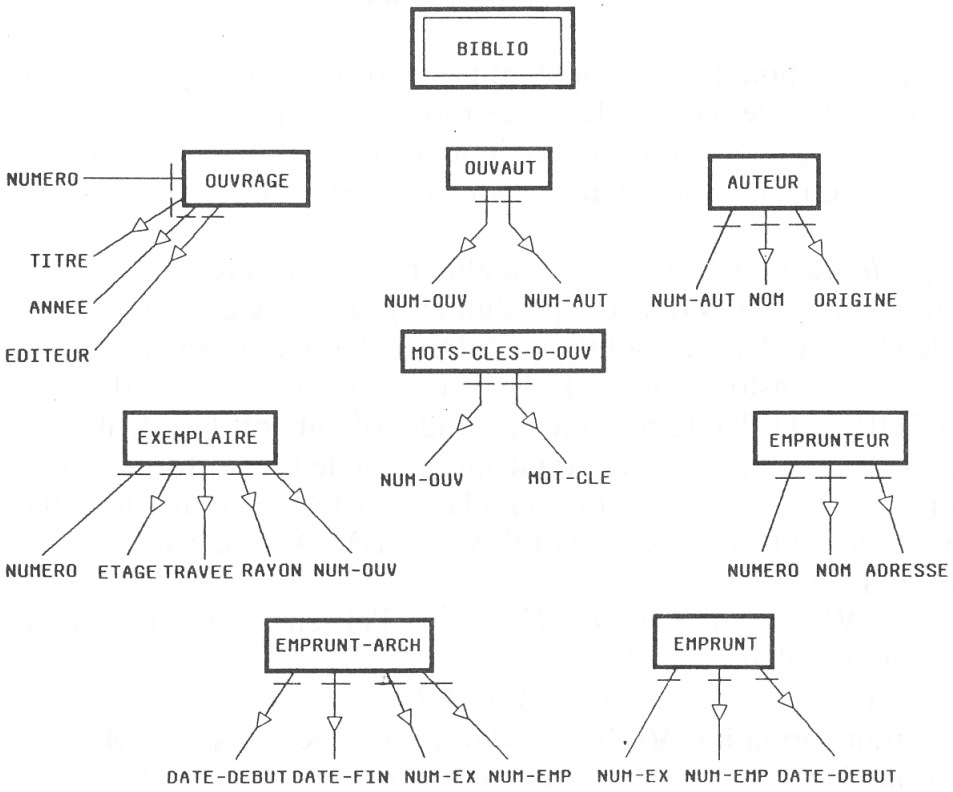
9.6.1 PRODUCTION D'UN SCHEMA CONFORME AU RELATIONNEL

La spécification des accès n'étant pas pertinente, on ne spécifiera pas de clés d'accès. Dans le schéma des accès possibles (schéma 9.2), on relève les constructions non compatibles avec

le modèle relationnel (plus précisément SQL/DS) et on en propose une transformation conforme.

- Type de chemins OUVAUT : on élimine le caractère N-N de OUVAUT par la transformation 7.1, qui produit le type d'articles OUVAUT et deux types de chemins 1-N (cfr. solution CODASYL). Ces derniers sont éliminés par rotation (transformation 7.3), vers les identifiants d'OUVRAGE et d'AUTEUR. AUTEUR n'ayant pas d'identifiant, on lui attribue un item NUM-AUT qui joue ce rôle, et rend ainsi possible la transformation. Le type de chemins OUVAUT est alors remplacé par le type d'articles OUVAUT doté des items obligatoires NUM-OUV et NUM-AUT constituant ensemble un identifiant.
- ORIGINE facultatif pour AUTEUR : l'absence de valeur sera représentée par la valeur NULL.
- MOT-CLE répétitif pour OUVRAGE : l'association N-N est éliminée par une transformation 7.1, qui définit le type d'articles MOT-CLE-D-OUV, doté de l'item MOT-CLE et en association N-1 avec OUVRAGE (cfr CODASYL). On transforme alors par rotation (7.3) ce type de chemins en un item obligatoire NUM-OUV de ce type d'articles. Ces deux items constituent un identifiant.
- type de chemins DE : éliminé par rotation vers NUMERO d'OUVRAGE, ce qui définit l'item obligatoire NUM-OUV d'EXEMPLAIRE.
- item LOCALISATION décomposable : remplacé par ses composants.
- type de chemins EXE : éliminé par rotation vers NUMERO d'EXEMPLAIRE, ce qui définit l'item identifiant obligatoire NUM-EX d'EMPRUNT.
- type de chemins EMPE : éliminé par rotation vers NUMERO d'EMPRUNTEUR, ce qui définit l'item obligatoire NUM-EMP d'EMPRUNT.
- types de chemins EXEA et EMPEA : éliminés par rotation comme dans les schémas CODASYL et COBOL. On propose de ne pas confier à SQL le soin de vérifier l'unicité des valeurs (NUM-EX, DATE-DEBUT) mais d'en assurer le respect par programme et par les règles de gestion du système.

Les transformations ci-dessus sont de pure conformité. On pourrait en proposer d'autres qui viseraient par exemple une plus grande efficacité. Ainsi l'ajout à OUVAUT du NOM d'AUTEUR pourrait accélérer certaines requêtes au détriment du volume occupé et des performances de mise-à-jour. Nous n'envisagerons pas de telles extensions.



Contraintes supplémentaires :

1. NUM-AUT(:OUVAUT) in NUM-AUT(:AUTEUR)
2. NUM-OUV(:OUVAUT) in NUMERO(:OUVRAGE)
3. NUM-OUV(:MOTS-CLES-D-OUV) in NUMERO(:OUVRAGE)
4. NUM-OUV(:EXEMPLAIRE) in NUMERO(:OUVRAGE)
5. NUM-EX(:EMPRUNT) in NUMERO(:EXEMPLAIRE)
6. NUM-EMP(:EMPRUNT) in NUMERO(:EMPRUNTEUR)
7. NUM-EX(:EMPRUNT-ARCH) in NUMERO(:EXEMPLAIRE)
8. NUM-EMP(:EMPRUNT-ARCH) in NUMERO(:EMPRUNTEUR)
9. Pour tout e d'EXEMPLAIRE et pour tout a d'EMPRUNT-ARCH(:NUM-EX=NUMERO(:e)) : DATE-DEBUT(EMPRUNT(:NUM-EX=NUMERO(:e))) >= DATE-FIN(:a)
10. EMPRUNT-ARCH est identifié par NUM-EX et DATE-DEBUT

Figure 9.11 - Schéma de la base de données BIBLIO conforme à SQL

Ces transformations proposées engendrent des contraintes d'inclusion (appelées ici contraintes référentielles). Le schéma 9.11 est l'expression graphique, accompagnée des contraintes, du schéma conforme au relationnel.

Les quantifications établies en 9.3.1. sont modifiées d'une manière analogue à ce qui a été réalisé pour l'analyse COBOL (9.5.1.), sauf principalement en ce qui concerne OUVAUT :

- De 0 à 8 OUVAUT par NUM-OUV, moyenne 2 par NUM-OUV et 1,7 par NUMERO d'OUVRAGE
- De 0 à 15 OUVAUT par NUM-AUT, moyenne 5 par NUM-AUT d'OUVAUT et 4,5 par NUM-AUT D'AUTEUR.

9.6.2 PRODUCTION D'UN SCHEMA RELATIONNEL

On trouvera à l'annexe 1 la séquence des ordres SQL (ou ISQL) qui permet de créer la base de données. Les clauses préfixées de la lettre L correspondent à l'expression SQL des constructions et des contraintes du schéma 9.11. On y remarque que certains index doivent déjà être déclarés comme supports d'identifiant. De manière à rendre homogènes les trois solutions, nous n'avons pas utilisé les types INTEGER, VARCHAR et DECIMAL qui auraient été plus indiqués dans certains cas.

9.6.3 PRODUCTION D'ALGORITHMES CONFORMES SQL/DS

L'algorithme prédicatif du module 1 (9.1) serait directement transformable en algorithme conforme s'il ne contenait la condition "TITRE = XT" qui, comme nous l'avons vu plus haut doit être transformée en "TITRE =S= XT". Cette dernière condition n'est pas évaluable par SQL (certains SGBD l'admettraient cependant) et doit donc être transformée en filtre associé à OUVRAGE. On développe donc la condition sur EXEMPLAIRE de manière à rendre explicite l'accès à OUVRAGE qui peut ainsi être suivi du filtre spécifié ci-dessus. La technique employée est celle des boucles développables à condition complexe (Section 7.4.1). On obtient ainsi l'algorithme 9.19. Il reste alors à rendre cet algorithme conforme, selon les principes de transformation 7.4.2, et en développant l'ordre "exit 0", ce qui conduit à l'algorithme 9.20, conforme à SQL. On a donné à cet algorithme une forme qui obéit aux conventions habituelles d'écriture en SQL. La structure de cet algorithme exclut la traduction en ISQL. Il sera donc traduit en un programme COBOL/SQL.

```
for B := BIBLIO do
    input(XT, XAN, XAUT);
    for O := OUVRAGE ((: ANNEE = XAN) and (OUVAUT: AUTEUR (:NOM =
XAUT))) do
        if TITRE(:O) =S= XT then
           for EX := EXEMPLAIRE ((EXE: 0 EMPRUNT) and (DE:O)) do
               print NUMERO(:EX);
               exit O;
           endfor;
```



```
        endif;
    endfor;
endfor;
```

Alg 9.19 Algorithme prédicatif du module 1 développé pour rendre explicite l'accès à OUVRAGE

```
for B := BIBLIO do
    input(XT, XAN, XAUT);
    TROUVE := false;
    for O := OUVRAGE
             (   (: ANNEE = XAN)
             and (:NUMERO in NUM-OUV
                             (: OUVAUT
                             (: NUM-AUT in NUM-AUT
                                          (: AUTEUR
                                          (: NOM = XAUT))))) ) do
             if TITRE(:O) =S= XT then
                for EX := EXEMPLAIRE
                          (   (:NUMERO not-in NUM-EX
                                           (: EMPRUNT))
                          and (:NUM-OUV = NUMERO(: O)  )      ) do
                    TROUVE := true;
                    print NUMERO(:EX);
                    exit EX;
                endfor EX;
             endif;
             if TROUVE then exit O;
    endfor O;
endfor B;
```

Alg 9.20 Algorithme du module 1 conforme à SQL/DS (optimisation implicite)

L'algorithme prédicatif du module 2 contient deux expressions non conformes : l'accès à EMPRUNT et l'ordre de création.

- L'accès à EMPRUNT se transforme en :
  `EMPRUNT(:NUM-EX = NUMERO(:EXEMPLAIRE(:NUMERO = XEX)))`
  De par la contrainte d'intégrité n° 5 du schéma 9.11, cette expression peut se simplifier selon la règle 1 donnée en 5.3.10 :
  `EMPRUNT(:NUM-EX = XEX)`

- Transformons l'ordre de création selon les règles de la section 7.4.2. Appliquées à la première condition du create, elles donnent :

```
EMPRUNT-ARCH(:NUM-EX = NUMERO(:EXEMPLAIRE(:NUMERO = NUM-EX(:E))))
```

Pour la même raison que ci-dessus, cette expression se réduit à :

```
EMPRUNT-ARCH(:NUM-EX = NUM-EX(:E))
```

La deuxième condition se traite d'une manière similaire. On obtient de la sorte l'algorithme 9.21, qui est conforme à SQL/DS.

```
for B := BIBLIO do
    input(XEX);
    begin-transaction;
    for E := EMPRUNT (: NUM-EX = XEX) do
        create EA := EMPRUNT-ARCH ( (NUM-EX  = NUM-EX  (: E)) and
                                    (NUM-EMP = NUM-EMP (: E)) and
                                  (: DATE-DEBUT = DATE-DEBUT(:E)) and
                                    (: DATE-FIN = DATE-DU-JOUR));
        delete E;
    endfor;
    close-transaction;
endfor;
```

Alg 9.21 Algorithme du module 2 conforme à SQL/DS (optimisation implicite)

Dans l'algorithme prédicatif du module 3, après avoir extrait le calcul de date, nous transformons l'ordre PRINT, dont les arguments deviennent respectivement :

1. NUMERO(:EXEMPLAIRE(:NUMERO = NUM-EX(:E)))
   qui se réduit, suivant le même principe que ci-dessus, à :
   NUM-EX(:E)

2. NOM(:EMPRUNTEUR(:NUMERO=NUM-EMP(:E)))
   En sortant de l'ordre d'impression l'accès à EMPRUNTEUR, on obtient l'algorithme 9.22.

```
for B := BIBLIO do
```



```
    DATE := MOINS (DATE-DU-JOUR, 14);
    for E := EMPRUNT (:DATE-DEBUT < DATE) do
        EMP := EMPRUNTEUR(: NUMERO = NUM-EMP(: E))
        print NUM-EX(: E)), NOM(: EMP);
    endfor;
endfor;
```

Alg 9.22 Algorithme du module 3 conforme à SQL/DS (optimisation implicite)

La traduction en SQL pourra s'effectuer à partir de cet algorithme, soit directement, soit en observant que les accès à E et EMP sont couplés par l'égalité "NUMERO=NUM-EMP" qui est l'expression typique de la jointure naturelle; la structure de l'algorithme autorise alors à grouper ces deux accès en un seul, qui se traduirait comme suit :

```
select NUM-EX, NOM
from   EMPRUNT, EMPRUNTEUR
where  DATE-DEBUT=:DATE
and    NUM-EMP=NUMERO
```

9.6.4 DEFINITION DES SCHEMAS EXTERNES RELATIONNELS

Le programme construit en 9.6.5 travaille sur les tables de base, et ne nécessite pas de vues qui lui seraient propres. Aucune contrainte de confidentialité n'ayant été définie, le schéma étant sous troisième forme normale et de faible complexité, il ne semble guère utile de définir de vues qui seraient spécifiquement destinées aux programmes des modules 1, 2 et 3. Nous construirons donc le programme du module 1 directement sur les tables de base. Nous pourrions cependant, si son utilité s'avérait réelle, définir la vue suivante :

```
create  view   OUVRAUTEUR (NUMERO,TITRE,ANNEE,EDITEUR,AUTEUR) as
select NUMERO,TITRE,ANNEE,EDITEUR,NOM
from   OUVRAGE,OUVAUT,AUTEUR
where  NUMERO=NUM-OUV
and    OUVAUT.NUM-AUT=AUTEUR.NUM-AUT
```

La séquence des OUVRAGES dans le programme du module 1 (voir ci-dessous) se réduirait comme suit :

```
select TITRE,NUMERO
from   OUVRAUTEUR
where  ANNEE = :XAN
and    AUTEUR = :XAUT
```

On notera cependant que cette vue donne une apparence non normalisée des données et qu'en outre elle ne permet pas la mise à jour.

9.6.5 CONSTRUCTION DES PROGRAMMES

Dans la section suivante, traitant de l'optimisation explicite, nous adopterons une architecture basée sur les modules d'accès. Nous proposerons ici la solution d'un programme contenant les ordres SQL eux-mêmes. SQL n'offrant pas la structure de boucle, et étant donné les caractéristiques de COBOL, nous développons d'abord l'algorithme 9.20 en rendant explicites les accès ponctuels, ce qui conduit à l'algorithme 9.23. On y observe que la primitive EXEMPLAIRE-SUIVANT n'y sera jamais exécutée et que l'ordre "exit" terminant le corps de la boucle intérieure réduit celle-ci à une alternative. En outre SQL ne dispose pas d'ordres explicites de connexion et de déconnexion relatifs à une base de données. En réduisant les structures "while" et "if" à leur forme la plus primitive, on obtient l'algorithme 9.24, que l'on traduit immédiatement sous la forme du programme de l'annexe 2.

```
input(XT, XAN, XAU);
TROUVE := false;
PREMIER-OUVRAGE ("OUVRAGE((:ANNEE=XAN) and (:NUMERO not-in ...))", O,
OK-O);
while (OK-O = 0) do
      COMPARER (TITRE(:O), XT, SEMBLABLE);
      if SEMBLABLE then
        PREMIER-EXEMPLAIRE ("EXEMPLAIRE((NUMERO not-in ...)", EX, OK-
EX);
         while (OK-O = 0) do
               TROUVE := true;
               print NUMERO(:EX);
               exit;
               EXEMPLAIRE-SUIVANT("EXEMPLAIRE((NUMERO not-in ...)",
EX, OK-EX);
         endwhile;
      endif;
      if TROUVE then exit;
            else OUVRAGE-SUIVANT("OUVRAGE((:ANNEE=XAN) and (:NUMERO
not-in
                                                    ...))", O, OK-O);
      endif;
endwhile;
```

Alg. 9.23 Algorithme du module 1 conforme à COBOL/SQL (Optimisation implicite) - Première étape.

```
     input(XT, XAN, XAU);
```



```
     TROUVE := false;
     PREMIER-OUVRAGE ("OUVRAGE((:ANNEE=XAN) and
                              (:NUMERO not-in ...))", O, OK-O);
TEST-OUVRAGE:
     if not (OK-O = 0) then goto FIN-OUVRAGE; endif;
        COMPARER (TITRE(:O), XT, SEMBLABLE);
        if not SEMBLABLE then goto FIN-SEMBLABLE; endif;
           PREMIER-EXEMPLAIRE ("EXEMPLAIRE((NUMERO not-in ...)", EX,
OK-EX);
            if not (OK-O = 0) then goto FIN-EXEMPLAIRE; endif;
                  TROUVE := true;
                  print NUMERO(:EX);
FIN-EXEMPLAIRE:
FIN-SEMBLABLE:
        if TROUVE then goto FIN-OUVRAGE; endif;
        OUVRAGE-SUIVANT("OUVRAGE((:ANNEE=XAN) and
                                (:NUMERO not-in ...))", O, OK-O);
        goto TEST-OUVRAGE;
FIN-OUVRAGE:
```

Alg. 9.24 Algorithme du module 1 conforme à COBOL/SQL (Optimisation implicite) - Version finale.

9.6.6 DEFINITION DU SCHEMA INTERNE DES DONNEES

La spécification du schéma interne consiste à définir les index non identifiants ainsi que différents paramètres de stockage et de fonctionnement. La détermination des index peut être délicate dans la mesure où c'est le SGBD lui-même qui choisit les stratégies d'accès. Afin de définir un jeu initial d'index, nous supposerons que, s'il disposait de tous les index qui pourraient lui être utiles, le SGBD choisirait l'algorithme d'accès que choisirait un bon programmeur. Le principe de détermination des index initiaux consiste donc à retenir les index qui seraient utilisés par les algorithmes effectifs rendus conformes au relationnel. Cette solution est strictement analogue à celle qui découle d'une optimisation explicite. On se reportera donc à son étude, qui est développée ci-dessous. La surveillance (monitoring) de l'exploitation confirmera ou infirmera l'utilité de chaque index et pourra suggérer la création de nouveaux index. Avant même l'exploitation, il est possible d'obtenir des informations sur les stratégies qui seront utilisées par SQL grâce à des fonctions comme EXPLAIN, relatives à un prototype réaliste de la future base de données.

Deuxième partie : CONCEPTION BASEE SUR UNE OPTIMISATION EXPLICITE

Selon cette approche (voir 8.6.3.3), on considère comme acquis les résultats de la conception logique standard, constituée du schéma des accès nécessaires (Schéma 9.7) ainsi que les algorithmes effectifs LDA/MAG (algorithmes 9.7, 9.8 et 9.9).

9.6.7 PRODUCTION D'UN SCHEMA CONFORME AU RELATIONNEL

Cette phase conduit à transformer le schéma des accès nécessaires sous une forme relationnelle. Le résultat (schéma 9.12) est obtenu de la même manière que le schéma 9.11; on y a en outre défini les clés d'accès issues, soit directement de celles du schéma des accès nécessaires, soit de la transformation des types de chemins. Les contraintes d'intégrité supplémentaires sont celles du schéma 9.11. Nous ferons remarquer, une fois encore, qu'idéalement seul le schéma 9.11, sans mécanismes d'accès, est connu du programmeur.

Certaines clés d'accès pourraient être remises en question :

- NOM d'EMPRUNTEUR, étant donné le faible volume de la future table; nous la conserverons cependant car le taux de mise-à-jour est très faible.
- NUM-EMP d'EMPRUNT, étant donné le faible volume d'accès via cette clé (25 par jour) et le taux important de mise-à-jour;
- DATE-DEBUT d'EMPRUNT pour les mêmes raisons. Nous déciderons cependant de les conserver sans autre analyse.

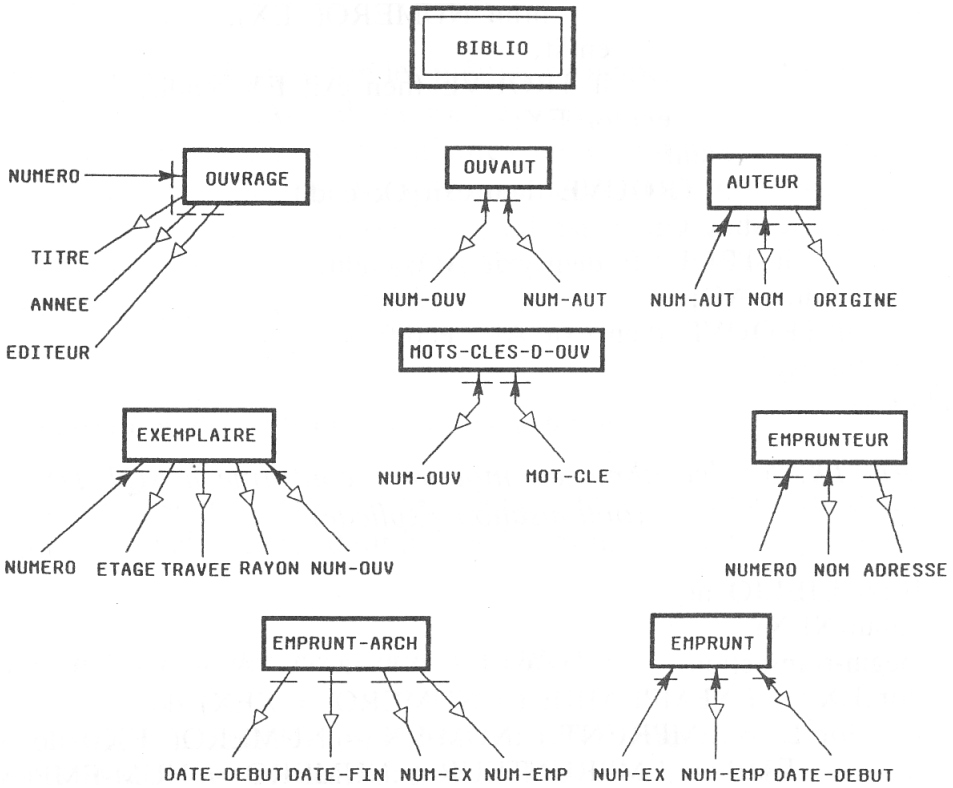


Figure 9.12 - Schéma des accès conforme au relationnel

9.6.8 PRODUCTION D'ALGORITHMES EFFECTIFS CONFORMES AU RELATIONNEL

Les algorithmes conformes (9.25, 9.26 et 9.27) s'obtiennent aisément à partir des algorithmes effectifs LDA/MAG. On y applique la transformation algorithmique 7.4.2 induite par une rotation d'un type d'associations. Il est alors possible d'établir le relevé des primitives utilisées et leur quantification selon la même démarche que pour CODASYL (tableaux 9.2, 9.3 et 9.5).

```
for B := BIBLIO do
    input(XT, XAN, XAU);
    TROUVE := false;
    for AU := AUTEUR (: NOM = XAU) do
        for AO := OUVAUT (:NUM-AUT = NUM-AUT(:AU)) do
            for O := OUVRAGE (:NUMERO = NUM-OUV(:AO)) do
                if (TITRE(:O) =S= XT) and (ANNEE(:O) = XAN) then
```

```
                    for EX := EXEMPLAIRE (:NUM-OUV = NUMERO(:O)) do
                        EX-OK := true;
                        for EM := EMPRUNT (:NUM-EX = NUMERO(:EX)) do
                            EX-OK := false;
                        endfor;
                        if EX-OK then
                           TROUVE := true;
                           print NUMERO(:EX);
                        endif;
                        if TROUVE then exit EX; endif;
                    endfor EX;
                endif;
                if TROUVE then exit O; endif;
            endfor O;
            if TROUVE then exit AO; endif;
        endfor AO;
        if TROUVE then exit AU; endif;
    endfor AU;
endfor B;
```

Alg 9.25 Algorithme du module 1 conforme à SQL/DS (optimisation explicite)

```
for B := BIBLIO do
   input(XEX);
   begin-transaction;
      for EX := EXEMPLAIRE (: NUMERO = XEX) do
          for E := EMPRUNT (:NUM-EX = NUMERO(:EX)) do
              EMP := EMPRUNTEUR (:NUMERO = NUM-EMP(:E));
             create EA := EMPRUNT-ARCH ( (:NUM-EX = NUMERO (:EX)) and
                                         (:NUM-EMP = NUMERO(:EMP)) and
                                    (:DATE-DEBUT = DATE-DEBUT(:E)) and
                                          (:DATE-FIN = DATE-DU-JOUR));
              delete E;
          endfor;
      endfor;
   close-transaction;
endfor;
```

Alg 9.26 Algorithme du module 2 conforme à SQL/DS (optimisation explicite)

```
for B := BIBLIO do
    DATE := MOINS (DATE-DU-JOUR, 14);
    for E := EMPRUNT (: DATE-DEBUT(:E) < DATE) do
             EX  := EXEMPLAIRE (:NUMERO = NUM-EX(:E));
```



```
            EMP := EMPRUNTEUR (:NUMERO = NUM-EMP(:E));
            print NUMERO(: EX), NUMERO(:EMP);
    endfor;
endfor;
```

Alg 9.27 Algorithme du module 3 conforme à SQL/DS (optimisation explicite)

9.6.9 PRODUCTION DU SCHEMA RELATIONNEL

Ce schéma est élaboré selon les mêmes principes qu'en 9.6.2 relatifs à l'optimisation implicite. Afin de conserver une démarche systématique, les clés d'accès qui ne sont pas identifiantes ne seront traduites que dans la définition du schéma interne. Rappelons encore que cette différence avec CODASYL et COBOL dérive du fait qu'il est possible de programmer en ignorant ces clés d'accès. Le texte de déclaration se trouve à l'annexe 1 (lignes préfixées par la lettre L).

9.6.10 DEFINITION DES SCHEMAS EXTERNES RELATIONNELS

L'absence de spécification de confidentialité et l'architecture de modules d'accès qui va être choisie rendent inutile la définition de vues relationnelles.

9.6.11 CONSTRUCTION DES PROGRAMMES COBOL/SQL

De manière à rendre homogène les différentes démarches selon les SGD, nous choisirons ici encore une architecture de niveau 3, que nous développerons, à titre d'exemple, pour le module 1. L'objectif désiré est d'obtenir un programme COBOL pour le module 1 qui soit indépendant du SGD. Ce programme est donc celui qui a été élaboré à l'annexe 2.

9.6.12 CONSTRUCTION DU MODULE D'ACCES COBOL/SQL

La spécification du module d'accès est indépendante du SGD (voir section 9.4.6). Les schémas algorithmiques 9.28 donnent les étapes du développement de l'accès à OUVRAGE, selon les mêmes principes que pour les autres SGD. Le texte du module d'accès est reporté à l'annexe 3.

```
OUVR-P:  for AO := OUVAUT (: NUM-AUT = NUM-AUT(:AU)) do
             for O := OUVRAGE (: NUMERO = NUM-OUV(:AO)) do
                 return (O);
OUVR-S:
             endfor O;
         endfor AO;
         return ();
```

```
OUVR-P:  OUVAUT-PREM (NUM-AUT(:AU),OK-AO);
         while (OK-AO = 0) do
             OUVRAGE (NUM-OUV(:AO),OK-O);
             if (OK-O = 0) then
                return;
             endif;
OUVR-S:
             OUVAUT-SUIV (OK-AO);
         endwhile;
         OK := 1;
         return;
```

```
OUVR-P:  OUVAUT-PREM (NUM-AUT(:AU),OK-AO);
TEST-AO: if not (OK-AO = 0) then goto FIN-AO; endif;
             OUVRAGE (NUM-OUV(:AO),OK-O);
             return;
OUVR-S:
             OUVAUT-SUIV (OK-AO);
         goto TEST-AO;
FIN-AO:  OK := 1;
         return;
```

Alg 9.28 Phases du développement de l'accès à OUVRAGE dans le module d'accès 1 (SQL/DS)

9.6.13 DEFINITION DU SCHEMA INTERNE

Nous considérerons comme appartenant au schéma interne la spécification des DBSPACES, l'affectation des tables à ces DBSPACES, le choix des agrégats physiques réalisés via les CLUSTERED INDEX et la fixation du taux de remplissage des index. Nous ignorerons les autres paramètres. Ces concepts, ainsi que certaines règles de conception sont décrits dans le manuel "SQL/DS Planning and Administration" (SQL3,84).

Le concept de base est celui de DBSPACE qui est un ensemble virtuel de pages (seules les pages contenant des données ont une existence réelle) dans lesquelles sont rangés les lignes et les index d'une ou plusieurs tables. On peut lui faire correspondre la notion de fichier. Un DBSPACE (de taille PAGES) est décomposé en un entête, une zone contenant les lignes des tables, un espace libre de réserve, puis une zone contenant les index, et dont la taille doit être prédéterminé (PCTINDEX). Lors du chargement, on peut fixer le taux de remplissage des pages de données (PCTFREE), puis le modifier par la suite. De même, on peut préciser et

modifier le taux de remplissage des index (PCTFREE). Nous déterminerons principalement les paramètres PAGES, PCTINDEX, PCTFREE des DBSPACES, ainsi que PCTFREE des index.

Nous exploiterons également la notion de CLUSTERED INDEX qui est un index associé à une table dont les lignes sont rangées par valeurs (dé-) croissantes des colonnes de cet index. On accélère ainsi notamment l'accès par un index non identifiant. On constitue un tel index après rangement des lignes dans l'ordre voulu. Enfin, nous proposerons, par chargement dit "entrelacé", la constitution d'agrégats physiques de lignes de plusieurs tables de manière à regrouper les lignes qui sont corrélées selon une "jointure" fréquemment demandée. Par exemple, on fera suivre physiquement chaque OUVRAGE de ses EXEMPLAIRES. Remarquons que ces structures ont tendance à se dégrader et qu'une réorganisation sera prévue de temps à autre.

Nous préciserons brièvement quelques caractéristiques techniques d'SQL/DS :

- toutes les pages sont de 4096 car.
- les lignes d'une table sont rangées en vrac, sauf s'il existe un CLUSTERED INDEX, auquel cas SQL tente de respecter l'ordre trié;
- un index est organisé comme un "B-tree", géré comme on l'a supposé en COBOL; tout index est dense, et contient donc une entrée par valeur présente dans la table. L'accès se fait classiquement par liste inversée;
- la longueur d'une ligne est calculée comme suit : 8 + L + N, où L est la somme des longueurs moyennes des colonnes (types LONG exclus), N est le nombre de colonnes acceptant la valeur NULL;
- la longueur d'une entrée d'index correspondant à moins de 256 lignes se calcule comme suit : 1 + L + Nx4, où L est la longueur de la clé, et N le nombre moyen de lignes correspondant à une même valeur de clé;

A. Détermination des modes de rangement et choix des DBSPACES

Nous choisirons de ranger dans un DBSPACE une table ayant un comportement différent des autres, ou les tables pour lesquelles on désire optimiser la jointure par rangement entrelacé associé à des CLUSTERED INDEX.

Les choix ci-dessous se feront en consultant le tableau 9.3 ou le schéma des trafics 9.8, qui devraient être traduits pour la structure relationnelle.

Nous grouperons OUVRAGE et EXEMPLAIRE dans un DBSPACE de nom DBSOUV. Le rangement s'y fera par valeurs croissantes de NUMERO pour OUVRAGE et de NUM-OUV pour EXEMPLAIRE, de telle sorte qu'en outre un OUVRAGE soit suivi de ses EXEMPLAIRES. Après chargement, on construira des index (dits CLUSTERED) sur chacune de ces deux colonnes. On privilégie ainsi l'accès à l'OUVRAGE d'un EXEMPLAIRE et aux EXEMPLAIRES d'un OUVRAGE, qui totalisent 4300 accès par jour.

Nous grouperons AUTEUR et OUVAUT dans un DBSPACE de nom DBSAUT, et nous les rangerons par valeurs croissantes de NUM-AUT d'AUTEUR et de OUVAUT, les OUVAUT suivant chacun de leurs AUTEURS. Un (CLUSTERED) index sera ensuite défini sur chacune de ces deux colonnes. On privilégie ainsi l'accès aux OUVRAGES d'un AUTEUR, et aux AUTEURS d'un OUVRAGE auxquels correspondent 6100 accès par jour. Ceci se fera au détriment de l'accès aux AUTEURS par NOM, qui n'est sollicité que 1200 fois par jour.

Nous rangerons MOT-CLE-D-OUV dans un DBSPACE autonome de nom DBSMOT. On définira un (CLUSTERED) index sur MOT-CLE de manière à privilégier l'accès aux OUVRAGES de MOT-CLE donné, au détriment de l'accès aux MOTS-CLE d'un OUVRAGE.

Nous rangerons EMPRUNT dans un DBSPACE autonome de nom DBSEMP, car cette table est très dynamique et de taille très variable, bien que de moyenne stable. Parmi les accès non identifiants (DATE-DEBUT et NUM-EMP), nous choisirons de privilégier par un CLUSTERED index le plus utilisé : DATE-DEBUT. Par la nature même des insertions cet index ne se dégradera pas (restera clustered), à condition que le DBSPACE soit très grand.

Nous rangerons EMPRUNTEUR dans un DBSPACE de nom DBSEMPR. Nous construirons un CLUSTERED index sur NUMERO. De par l'attribution des valeurs de NUMERO, il s'agit d'un fichier en extension séquentielle.

La table EMPRUNT-ARCH sera rangée dans un DBSPACE de nom DBSEMPA.

Ces décisions nous permettent de préciser la clause "in" dans les ordres de création de table à l'annexe 1. Comme d'habitude, de telles clauses y seront préfixées de la lettre P.

B. Calcul du volume des tables

On effectue le calcul par DBSPACE, en considérant selon son contenu soit des articles individuels, soit des agrégats. Ces derniers seront constitués dans DBSOUV d'1 article OUVRAGE et 2,7 articles EXEMPLAIRE, et dans DBSAUT d'1 article AUTEUR et de 4,5 articles OUVAUT.

On déterminera la longueur physique moyenne d'un article ou agrégat (LPA), le nombre initial d'articles ou agrégats (NIA), le pourcentage d'augmentation en 4 ans (avec une interprétation particulière pour EMPRUNT, cfr 9.4.7) (%4), le pourcentage d'espace libre dans chaque page au chargement (%FREE), la taille effective des pages, telle que spécifiée dans la table 179 du manuel (SQL3,84) (EPS), ainsi que le nombre réel de pages (NRP).

Les résultats de cette évaluation sont consignés dans le tableau 9.10. Détaillons par exemple le calcul du DBSPACE DBSOUV, contenant OUVRAGE et EXEMPLAIRE.

Un article OUVRAGE a une longueur logique de 6+80+6+45 = 137 et une longueur physique de 137+8 = 145. Pour EXEMPLAIRE ces longueurs sont de 26 et 34. Un agrégat OUVRAGE/EXEMPLAIRE a une longueur physique moyenne de 145+2,7x34 = 237. Dans 4 ans, on compte sur une augmentation en volume de l'ordre de 11%. En admettant 15% d'espace libre au chargement, la table des tailles effectives indique le remplissage de 3096

car. par page. Il y a donc 13 agrégats par page au chargement, ce qui donne un nombre réel de 12308 pages de 4096 car.

DBSPACE	LPA	NIA	%4	%FREE	EPS	NRP
DBSAUT	190	60.000	16	16	3096	3750
DBSOUV	237	160.000	11	15	3096	12308
DBSMOT	34	320.000	11	15	3096	3517
DBSEMP	30	2.000	900	0	4066	350
DBSEMPR	111	2.000	44	0	3846	110
DBSEMPA	36	200.000	176	0	4046	4930

Tableau 9.10 Calcul du volume des tables SQL/DS

C. Calcul du volume des index

Pour chacun des 14 index retenus, on déterminera le nombre de valeurs (NVAL), le pourcentage d'espace libre au chargement (%FREE), qui sera, pour les mêmes raisons qu'en COBOL, de 20% lorsqu'on observe des insertions aléatoires, et de 0% pour des insertions par valeurs strictement croissantes. On déterminera également la taille effective des pages donnée par la formule : 4096-41x%FREE (EIPS), la longueur moyenne d'une entrée dans l'index (LI), et le nombre de niveaux d'index (NI). On donnera enfin le volume de l'index (VI). Les résultats sont consignés dans le tableau 9.11. On développera par exemple le calcul de l'index NOM d'AUTEUR.

Il y a en moyenne 3 AUTEURS par NOM présent dans l'index, et donc 20.000 valeurs dans cet index. Les insertions ultérieures étant aléatoires quant au NOM, on admettra le taux de chargement traditionnel de 80%., ce qui donne une taille effective de 3236 car. par page. Chaque valeur a une longueur de 35 caractères, et est accompagnée de trois pointeurs en moyenne, ce qui constitue une entrée d'une longueur moyenne de 35+1+3x4 = 47. Une page peut contenir 68 entrées d'index au chargement. Il faut donc prévoir 295 pages au premier niveau de l'index. Le deuxième niveau contient 295 entrées constituées d'une valeur (35 car) et d'un pointeur (4 car). Ces entrées occupent 4 pages, et nécessitent donc un troisième niveau d'index d'une seule page. Cet index dispose donc de 3 niveaux et occupe 300 pages.

Il est à présent possible d'établir les paramètres des DBSPACES, qui sont synthétisés dans le tableau 9.12. On y reprend, exprimées en pages, la taille des tables (NRP), la taille totale des index (VI), la taille utile totale, (TTOT) égale à NRP+VI, la taille des DBSPACES (TDBS),

égale à NRP+VI+réserve arrondi à un multiple de 128, ainsi que la proportion en % de VI dans NRP+VI (%IDX).

Index	NVAL	%FREE	EIPS	LI	NI	VI
AUTEUR (NUM-AUT)	60.000	20	3.236	10	2	187
AUTEUR (NOM)	20.000	20	3.236	47	3	300
OUVAUT (NUM-OUV,NUM-AUT)	272.000	20	3.236	16	3	1.355
OUVAUT (NUM-AUT)	54.400	20	3.236	26	3	442
OUVRAGE (NUMERO)	160.000	20	3.236	11	3	547
EXEMPLAIRE (NUMERO)	420.000	20	3.236	13	3	1696
EXEMPLAIRE (NUM-OUV)	140.000	20	3.236	19	3	826
MOT-CLE-D-OUV (NUM-OUV,MOT-CLE)	320.000	20	3.236	31	3	4.007
MOT-CLE-D-OUV (MOT-CLE)	20.000	20	3.236	85	3	531
EMPRUNT (NUM-EX)	20.000	20	3.236	13	2	81
EMPRUNT (NUM-EMP)	1.000	20	3.236	89	2	29
EMPRUNT (DATE-DEBUT)	20	0	4.056	*	2	21
EMPRUNTEUR (NUMERO)	2.880	0	4.056	13	2	12
EMPRUNTEUR (NOM)	2.830	20	3.236	40	2	31

Tableau 9.11 Calcul des index SQL/DS

DBSPACE	NRP	VI	TTOT	TDBS	%IDX
DBSAUT	3.750	2.284	6.034	8.064	37
DBSOUV	12.308	3.069	15.377	20.096	20
DBSMOT	3.517	4.538	8.055	10.240	56
DBSEMP	350	131	481	768	28
DBSEMPR	110	43	153	256	28

DBSEMPA	4.930	0	4.930	7.680	0
Total	24.965	10.065	35.030	47.103	

Tableau 9.12 Calcul des paramètres des DBSPACES

Le texte SQL de l'annexe 1 peut ainsi être complété par les clauses PAGES (= TTOT dans 9.12), PCTINDEX (= %IDX dans 9.12), PCTFREE (= %FREE dans 9.10) dans la spécification des DBSPACES, et par les clauses PCTFREE (= %FREE dans 9.11) dans la spécification des index. On veillera également à opérer un chargement des tables et la création des index selon les directives définies plus haut.

9.6.14 CALCUL DES VOLUMES ET DES COUTS D'ACCES

Les volumes totalisent 35030 pages de 4096 car., soit 143.500.000 car. Les résultats du calcul des temps d'accès sont repris au tableau 9.13, selon les mêmes conventions qu'en CODASYL (9.4.8). On fera à leur sujet les remarques suivantes :

- les temps CPU, d'accès au dictionnaire, de gestion de la concurrence et ceux liés à la sécurité n'ont pas été comptabilisés;
- plus que pour les autres SGD, les chiffres obtenus risquent d'être imprécis, en raison des fonctions d'optimisation que le SGD peut mettre en oeuvre; on pourra cependant considérer ces valeurs comme pessimistes dans la mesure où SQL/DS est susceptible de trouver une meilleure stratégie d'accès;
- on admet que les buffers sont suffisamment grands pour que la page du premier niveau de chaque index reste en permanence en mémoire centrale.

Primitives	NA/J	NE/J	(c1; cs)	NAP/A	TE/A	TE/J
AUTEUR(:NOM=X)	200	2,1	(3;1)	4,1	144	29.000
OUVAUT(:NUM-AUT=X)	420	4	(2;0,015)	2,05	72	30.000
OUVRAGE(:NUMERO=X)	1680	1	(3;0)	3	105	176.000
EXEMPLAIRE(:NUMERO=X)	600	1	(3;0)	3	105	63.000
EXEMPLAIRE(:NUM-OUV=X)	180	1	(3;0,03)	3	105	19.000
EMPRUNT(:DATE-DEBUT<X)	1	200	(2;0,25)	52	1820	1.800
EMPRUNT(:NUM-EX=X)	580	1	(2;0)	2	70	40.000
delete EMPRUNT	400	1	4,1	4,1	144	58.000

create EMPRUNT-ARCH	400	1	2	2	70	28.000
EMPRUNTEUR(:NUMERO=X)	600	1	(2;0)	2	70	42.000
Total	487 sec					

Tableau 9.13 Temps d'accès journaliers des modules 1, 2, 3 (SQL/DS)

Chapitre 10 : BIBLIOGRAPHIE

ABRIAL, "Data Semantics", in Data Base Management, North Holland Publish. 1974, pp1-59.

ADIBA, "Les Systèmes de Base de Données relationnelles", in Informatique et Gestion, 125, Juin 1981.

ANSI, "Network Database Language", Draft Report X3H2-84-1, ANSI, Jan. 1984.

ANSI, "Relational Database Language", Draft Report X3H2-84-2, ANSI, Feb. 1984.

ANSI, "ANSI/X3/SPARC Study Group on Data Base Management System Interim Report", FDT (ACM-SIGMOD Bulletin), 7, No 2 (1975)

BACHMAN, "Data Structure Diagrams", in Data Base, 1, No 2, 1969, Public. of ACM SIG on Business Data Processing

BENCI, BODART, BOGAERT, CABANES, "Concepts for the Design of a Conceptual Schema", in "Modelling in Data Base Management Systems", 1976, North Holland Publish.

BENCI, ROLLAND, "Bases de Données", Editions SCM, Paris, 1979.

BILLER, "On the equivalence of Data Base Schemas - A semantic Approach to Data Translation", in Information Systems, Vol 4, pp 35-47.

BLASGEN, ASTRAHAN, and al, "SYSTEM R : An architecture overview", in IBM Syst. J., 20, 1, 1981, p41-62.

BODART, PIGNEUR, "Conception assistée des applications informatiques, 1. Etude d'opportunité et Analyse conceptuelle", MASSON, 1983.

BOCK, DELVAL, "Génération automatique d'interfaces d'accès à des bases de données", mémoire de fin d'études, 1982, Institut d'Informatique de Namur.

BRACCHI, PAOLINI, PELAGATI, "Binary Logical Association in Data Modelling", in "Modelling in DBMS", North Holland Publish.

BUBENKO, BERILD, LINDENCRONA, NACHMENS, "From Information Requirements to DBTG-Data Structures", in ACM Sigmod-Sigplan proceedings, March 1976, pp 73-85.

CERI (Ed), "Methodology and tools for data base design", North-Holland Publish. 1983

CHAMBERLIN, and al, "SEQUEL 2 : A unified Approach to Data Definition, Manipulation and Control", in IBM J. Res. Develop., Nov 1976, p560-575.

CHEN, "An Entity/Relationship Model - Toward a unified view of data", ACM TODS, 1976, Vol 1,1.

CLARINVAL, "Comprendre, connaître et maitriser le COBOL, normes ANSI COBOL 1974", Presses Universitaires de Namur, 1981.

CODASYL, Development Committee"An Information Algebra", Com. ACM, 5, no 4 (April 1962), pp.190-204.

CODASYL, "Data Base Task Group, April 1971 Report", ACM/BSC/IFIP.

CODASYL, "Data Definition Language Committee Report", DDL Journal of Development, June 1973, IFIP Public.

CODASYL, "Data Definition Language Committee Report", Special issue of INFORMATION SYSTEMS, vol 3, 4, 1978.

E. F. CODD, "A Relational Model of Data for large, shared, Data Banks", in Com. ACM, 13,6, June 1970, p377-387.

COLUSSI, DEHENEFFE, GUILLEBAUD, HAINAUT, HENNEBERT, LECHARLIER, PAULUS,"Système de conception et l'exploitation de bases de données" vol. 1 à 5, Rapport final de la deuxième partie du projet CIPS I.2/15, Janv. 1978, Institut d'Informatique de Namur.

DATE, "An Introduction to Data Base Systems", Vol 1, ADDISON WESLEY, 3rd Edition, 1981.

DATE, "An Introduction to Data Base Systems", Vol 2, ADDISON WESLEY, 1983.

DATE, "An introduction to th Unified Database Language (UDL)", in 1980 V.L.D.B. Conf. Proceedings, ACM/IEEE, pp15-29.

DEHENEFFE, HENNEBERT, PAULUS, "Relational Model for A Data Base", 1974 IFIP Congress Proceedings, North Holland Publish.

DEHENEFFE, HAINAUT, TARDIEU, "The Individual Model", in the Proceedings of the International Workshop of Namur, 1974, Presses Universitaires de Namur.

DEHENEFFE, HAINAUT, HENNEBERT, LECHARLIER, PAULUS, "Système de conception et l'exploitation d'une base de données" vol. 1 et 2, Rapport final de la première partie du projet CIPS I.2/15, Déc. 1974, Institut d'Informatique de Namur.

DELOBEL, ADIBA, "Bases de Données et Systèmes Relationnels", DUNOD, 1983.

DELVAUX, HAINAUT, "Système portable de manipulation de bases de données hétérogénes", Actes du Congrès AFCET 1981, Ed. Hommes et Techniques.

E. M. DIECKMAN, "Three Relational DBMS", in DATAMATION, Sept 1981, p137-148.

FLORY, "Bases de données, Conception et réalisation", ECONOMICA, 1982

GARDARIN, " ", 1984

HAINAUT, "Evaluation analytique des performances de programmes ADL", Rapport de recherche, 1976, Institut d'Informatique.

HAINAUT, "Evaluation des performances d'une base de données par modèle probabiliste", in Cahier INFORSID Nr. 2, IRIA (act. INRIA, France), 1977, 45p.

HAINAUT, "Some tools for data independence in multilevel DBMS", in"Architecture and Models in DBMS", North Holland Publish., 1977, pp 187-211.

HAINAUT, "IDML, Système d'interaction avec des bases de données Codasyl", Actes de l'Ecole d'Eté de l'AFCET, Namur, 1978.

HAINAUT, "Analyse du cas PETITPAS - Dossier d'analyse organique- La base de données", notes de cours, 1979, Institut d'Informatique.

HAINAUT, "Analyse des accès proposés par le DBTG CODASYL 71", notes de cours, Institut d'Informatique, Juin 80.

HAINAUT, "Un modèle relationnel généralisé", notes de cours, Institut d'Informatique, 1980,

HAINAUT, "Dérivation d'une première structure d'accès à partir d'un schéma conceptuel Entité/Association", notes de cours, Institut d'Informatique, Déc 80.

HAINAUT, "Construction de schémas conceptuels d'une base de données. Etude de cas n[1", Rapport de recherche, 1980, Institut d'Informatique.

HAINAUT, "Traduction de programmes ADL en COBOL/DML CODASYL 71", notes de cours, Institut d'Informatique, Mars 81.

HAINAUT, "Conception de la base de données d'un Système d'Information", Chapitre du cours de Conception de Systèmes d'Information donné par F. Bodart à l'Ecole d'Eté de l'AFCET, Juillet 1981 à Thiès (Sénégal). Public. Institut d'Informatique.

HAINAUT, "Un modèle descriptif de Bases de Données au niveau organique : Le Modèle d'Accès", notes de cours, Institut d'Informatique, 1981.

HAINAUT, "General Model and Languages for Logical Data Description and Manipulation in DBMSs", Technical report, ISO/97/5/5/N22, Feb 82.

HAINAUT, " ISO MUNICH

HAINAUT, "Conception de Bases de Données, un essai d'analyse des concepts et des méthodes", in Actes du Colloque "Current Trends in Data Bases", Namur, Juin 1982, Public. Institut d'Informatique.

HAINAUT, "Theoretical and practical tools for data base design", 1982 V.L.D.B conf. Proceedings, Sept 82, ACM/IEEE, pp 216-224.

HAINAUT, "ADL: un langage de description d'algorithmes", notes de cours, Institut d'Informatique, 1983.

HAINAUT, "Cadre de Référence pour la Conception de Bases de Données", 121p, Public. Institut d'Informatique, Namur, Déc. 1983.

HAINAUT, "Le Modèle d'Accès Généralisé", 60p, Public. Institut d'Informatique, Namur, Janv. 1984.

HAINAUT, "Introduction à un système de gestion de bases de données relationnel", 38p, Public. Institut d'Informatique, Namur, Janv. 1984.

HAINAUT, LECHARLIER, "Modèles, langages et système pour la conception et l'exploitation de bases de données", in "Théorie et techniques de l'Informatique", Actes du Congrès AFCET 1878, pp 179-189, Edit. Hommes et Techniques.

HAINAUT, LECHARLIER, "An extensible semantic model of data base and its data manipulation language", 1974 IFIP Congress Proceedings, North Holland Publish., pp 1026-1030.

HERSHEY, DISSEN,MESSINK, "A description of ADBMS V2.0", ISDOS Working paper Nr. 122, Aug. 1975, The University of Michigan.

HOWE, "Data Analysis for Data Base Design", 1983, Edward Arnold Publish.

IRANI, PURKAYASTHA, TEOREY, "A designer for DBMS-Processable Logical Database Structures", 1975 V.L.D.B. Conf. Proceedings, ACM.

INFOTECH, "INFOTECH State of the Art Report : System Tuning", Infotech Intern. Ltd, 1977

ISO, " ", WG3, 1982

JACKSON, " ", 1983

JARDINE, DAVIS, "A Data-Base Application Design Language", in Information and Management, 4 (1981), pp 81-93.

JOUFFROY, LETANG, "Les Fichiers", Dunod Informatique, 1977.

KATZAN, "Computer Data Management and Data Base Technology", Van Nostrand Reinhold (New York, London), 1975

KING, " ", 1974

LUM, SHU, HOUSEL, "A general Methodology for Data Conversion and Restructuring", in IBM J. Res. Develop., Sept 76, pp 483-497.

MARTIN, "Computer Data-Base Organization", Prentice-hall, 1977.

MARTIN, "Managing the Data-Base Environment", Prentice-Hall, 1983.

MIRANDAS, BUSTA, "L'art des bases de données. 1 Introduction aux bases de données", Eyrolles, 1984

MITOMA, IRANI, "Automatic Data Base Schema Design", in 1975 V.L.D.B. Conf. Proceedings, ACM.

NBS, "Guide on Logical Database Design", National Bureau of Standards Special Publication 500-122, 1985

NIJSSEN, "An overall Model for Information Systems Design and associated Practical Tools", in "Formal and Practical Tools for Information System Design", Oxford, April 1979, North Holland Publish.

OLLE, "The Codasyl Approach to Data Base Management", Wiley, 1978.

PEETERS, "Conception et gestion de banques de données", Editions d'Organisation, Paris, 1984

PELAGATTI, PAOLINI, BRACCHI, "Mappings in database systems", in 1977 IFIP Congress Proceedings, North Holland Publish.

PELLAUMAIL, "AXIAL, une méthode de conception de Système d'Information proposée par IBM France", Informatique et Gestion - Octobre 1980.

PERRON, "Design guide for Codasyl DBMS", Q.E.D. Information Sciences, 1981.

PRABUDDHA, HASEMAN, KRIEBEL, "Towards an Optimal Design of a Network Database from Relational Descriptions", in Operation Research, vol 26, nr 5, 1978, pp 805-823.

RISCH, "Production Program Generation in a Flexible Data Dictionary System", in 1980 V.L.D.B Conf. Proceedings, IEEE.

ROBERT, "Génération automatique d'interfaces d'accès à des bases de données COBOL", mémoire de fin d'études, 1981, Institut d'Informatique de Namur.

ROBINSON, "Database Analysis and Design", Chartwell-Bratt Publish., England, 1981.

RUSTIN, "Data Base Management System Performance Estimation", in (INFOTECH,77), pp373-399

SCHKOLNICK, "A survey of Physical Database Design Methodology and Techniques" in 1978 V.L.D.B. Conf. Proceedings, IEEE, pp474-487.

SENKO, " ", 1969

SENKO, "The DDL in the context of a Multilevel Structured Description : DIAM II with FORAL", in "Data Base Description", 1975, North Holland Publish., pp 239-257.

SEVERANCE, "A practioner's guide to addressing algorithms", Com. of ACM, vol 19, 6, 1976

SHNEIDERMAN, THOMAS, "Path expressions for complex queries and automatic database program conversion", 1980 V.L.D.B. Conf. Proceedings, ACM/IEEE, pp 15-29.

SIBLEY, TAYLOR, "A Data Definition and Mapping Language", in Com. ACM, vol 16, 12, pp 750-759.

SMITH, SMITH, "Database Abstraction : aggregation and generalization", ACM TODS, 2,2 (JUNE 1977)

TAMIR, MISSINAI, ARDITI, RABAN, SOLE, TIMOR, ZUKOVSKY, "DB1, A DBMS-Based Application Generator", in 1980 V.L.D.B. Conf. Proceedings, IEEE.

TARDIEU, NANCI, PASCOT, "Conception d'un Système d'Information", Les Editions d'Organisation, Paris, 1979.

TARDIEU, ROCHFELD, COLETTI, "La Méthode Merise", Les Editions d'Organisation, 1983.

TEOREY, FRY, "Design of Database Structures", Prentice-Hall, 1982.

TEOREY, FRY, "The Logical Record Access Approach to Database Design", in Computing Surveys, vol 12, june 1980, pp 179-211.

TSICHRITZIS, "LSL: A Link and Selector Language", 1976 ACM-SIGMOD Conf. Proceedings.

TSICHRITZIS, KLUG (Ed), "The ANSI/X3/SPARC DBMS framework : Report of the Study Group on Data Base Management System", Information Systems 3 (1978)

ULLMAN, "Principles of Database Systems", Computer Sciences Press, Washington, 1979

WANG, WEDEKIND, "Segment Synthesis in Logical Data Base Design", in IBM J. Res. Develop., Jan. 1975, pp 71-77.

WATERS, "Estimating magnetic disk seeks", The Computer Journal, Vol 18, 1, 1975

WEDEKING, "System-independent program design with databases", Report nr ID78-1, Feb 1978, Technical University Darmstadt, WG

WIEDERHOLD, "Data Base Design", Mac-Grawhill, 1977.

WON KIM, "Relational Database Systems", in ACM Computing Survey, 11, 3, Sept 1979.

- , "Relational Database System, Analysis and Comparison", Springer-Verlag, Berlin, 1983, 618p.

- , "Journal of Development", Summer 1982 Report of the Data Dictionary Systems Working Party, British Computer Society.

- , "The ANSI/SPARC DBMS Model", Proceedings of the 2d SHARE WC on DBMS, Canada, 1976, North Holland Publish.

- , "Data Structure Models for Information Systems", Proceedings of the International Workshop of Namur, May 1974. Presses Universitaires de Namur.

- , "Data Base Design Aid: Designer's Guide", IBM Documentation, GH10-1627.

- , "ORACLE, Terminal User Guide", Oracle Corporation, May, 1983

- , "SQL/DATA SYSTEM Terminal User's Reference", IBM documentation, SH24-5017-1, 1982

- , "SQL/DATA SYSTEM Application Programming", IBM documentation, SH24-5018-1, 1982

- , DBMS-20 : Data Base Management System, - Programmer's Procedure Manual, DEC Documentation, May 1977.

- , DBMS-20 : Data Base Management System, - Administrator's Procedure Manual, DEC Documentation, May 1977.

- , ADABAS, Software A.G., W.G.

- , "SESAM, Wiedergewinnung und Direkt~anderung", Juni 1975,
Documentation Siemens.

- , "MDBS, Data Base Design Reference Manual", July 1981, Micro Data Base Systems Inc.
- , "SIBAS, Users Manual" Oct. 1976, Norsk Data Public. Nr. ND-60.057.02.
- , "DMS II, Data and Structure Definition Language Reference Manual", ref. 5001084, BURROUGHS, Detroit USA, 1976.
- , "SOCRATE, Manuel d'utilisation et manuel d'opération", ERIA-ECA AUTOMATION, 1977.
- , TOTAL, Cincom Inc.
- , "IMS Primer", IBM Documentation.
- , "IDS1, Integrated Data Store", Honeywell Documentation.

Chapitre 11 : ANNEXE 1 : TEXTE DES SCHEMAS EXECUTABLES DE LA BASE DE DONNEES "BIBLIO"

11.1 SCHEMA CODASYL DE LA BASE DE DONNEE "BIBLIO"

```
P    images not by command.
P    journal is SYSJ1 size is 100 transactions.

P    assign ARCHBIB to SYSAB
P    buffer count is 1
P    first page is 1
P    last page  is 8762
P    page size is 512 words.

P    assign ABIBLIO to SYSBI.
P    buffer count is 26
P    calc 6 record-per-page
P    first page is 9000
P    last page  is 83000
P    page size is 512 words
P    range of AUTEUR        is page 9000  to page 13302
P    range of OUVRAGE       is page 14000 to page 55999
P    range of OUVAUT        is page 14000 to page 55999
P    range of MOT-CLE-D-OUV is page 57000 to page 64599
P    range of EXEMPLAIRE    is page 66000 to page 79799
P    range of EMPRUNT       is page 80000 to page 80251
P    range of EMPRUNTEUR    is page 81000 to page 81211
P    range of NOMS          is page 82000 to page 82078.


L    schema name is BIBLIO.

L    area name is ABIBLIO.
L    area name is ARCHBIB.
```



```
L    record name is AUTEUR,
L           location mode calc using NOM,
L           within ABIBLIO.
L           2 NOM pic X(35).
L           2 ORIGINE pic X(50).

L    record name is OUVRAGE,
L           location mode calc using NUMERO, duplicates not allowed,
L           within ABIBLIO.
L           2 NUMERO pic 9(6).
L           2 TITRE pic X(80).
L           2 ANNEE pic 9(6).
L           2 EDITEUR pic X(45).
L           2 N-MOTS-CLES pic 9(2).
L           2 MOT-CLE pic X(20) occurs 10 depending on N-MOTS-CLES.

L    record name is EXEMPLAIRE,
L           location mode calc using NUMERO, duplicates not allowed,
L           within ABIBLIO.
L           2 NUMERO pic 9(8).
L           2 LOCALISATION pic X(12).

L    record name is EMPRUNTEUR,
L           location mode calc using NUMERO, duplicates not allowed,
L           within ABIBLIO.
L           2 NUMERO pic 9(8).
L           2 NOM pic X(35).
L           2 ADRESSE pic X(60).

L    record name is EMPRUNT,
P           location mode via BE,
L           within ABIBLIO.
L           2 DATE-DEBUT pic 9(6).

L    record name is EMPRUNT-ARCH,
P           location mode direct EA-DBKEY,
L           within ARCHBIB.
L           2 NUM-EX pic 9(8).
L           2 NUM-EMP pic 9(8).
L           2 DATE-DEBUT pic 9(6).
L           2 DATE-FIN pic 9(6).

L    record name is OUVAUT,
P           location mode via OO,
L           within ABIBLIO.
```

```
L    record name is MOT-CLE-D-OUV,
L           location mode calc using MOT-CLE,
L           within ABIBLIO.
L           2 MOT-CLE pic X(20).

L    record name is NOMS,
L           location mode calc using NOM, duplicates not allowed,
L           within ABIBLIO.
L           2 NOM pic X(35).

L    set name DE,
P           mode is chain,
P           order is always first,
L           owner OUVRAGE,
L           member EXEMPLAIRE, mandatory automatic,
P           linked to owner,
L           set selection thru current.

L    set name EXE,
P           mode is chain,
P           order is always first,
L           owner EXEMPLAIRE,
L           member EMPRUNT, mandatory automatic,
L           set selection thru current.

L    set name EMPE,
P           mode is chain,
P           order is aways last,
L           owner EMPRUNTEUR,
L           member EMPRUNT, mandatory automatic,
L           set selection thru current.

L    set name AO,
P           mode is chain,
P           order is always first,
L           owner AUTEUR,
L           member OUVAUT, mandatory automatic,
P           linked to owner,
L           set selection thru current.

L    set name OO,
P           mode is chain,
P           order is always first,
L           owner OUVRAGE,
L           member OUVAUT, mandatory automatic,
L           set selection thru current.
```



```
L    set name OMC,
P           mode is chain,
P           order is always first,
L           owner OUVRAGE,
L           member MOT-CLE-D-OUV, mandatory automatic,
P           linked to owner,
L           set selection thru current.

L    set name NE,
P           mode is chain,
P           order is always first,
L           owner NOMS,
L           member EMPRUNTEUR, mandatory automatic,
L           set selection thru current.

L    set name BE,
P           mode is chain linked to prior,
P           order always last,
L           owner system,
L           member EMPRUNT, mandatory automatic,
L           set selection thru current.


E  subschema SS1.
E  area section.
E         copy ABIBLIO.
E  record section.
E         01 AUTEUR.
E            02 NOM.
E         01 OUVAUT.
E         01 OUVRAGE.
E            02 TITRE.
E            02 ANNEE.
E         01 EXEMPLAIRE.
E            02 NUMERO.
E         01 EMPRUNT.
E            O2.
E  set section.
           copy AO,OO,DE,EXE.
```

11.2 SCHEMA DES FICHIERS COBOL DE LA BASE DE DONNEES "BIBLIO"

11.2.1 ENVIRONMENT DIVISION (Fichier BDMOD11)

```
L+P     select FAUT assign to DA-SYSA,
L       organization is indexed,
E       access mode is dynamic,
L       record key is NUM-AUT of AUTEUR,
L       alternate key is NOM of AUTEUR with duplicates,
E       file status is FSTAT
P       reserve 1 area.

L+P     select FOUVR assign to DA-SYSO,
L       organization is indexed,
E       access mode is dynamic,
L       record key is NUMERO of OUVRAGE,
E       file status is FSTAT,
P       reserve 1 area.

L+P     select FMOT assign to DA-SYSM,
L       organization is indexed,
E       access mode is dynamic,
L       record key is ID-MOT-CLE of MOTS-CLE-D-OUV,
E       file status is FSTAT,
P       reserve 1 area.

L+P     select FEX assign to DA-SYSX,
L       organization is indexed,
E       access mode is dynamic,
L       record key is NUMERO of EXEMPLAIRE,
L       alternate key is NUM-OUVR of EXEMPLAIRE with duplicates,
E       file status is FSTAT,
P       reserve 1 area.

L+P     select FEMPR assign to DA-SYSER,
L       organization is indexed,
E       access mode is dynamic,
L       record key is NUMERO of EMPRUNTEUR,
L       alternate key is NOM of EMPRUNTEUR with duplicates,
E       file status is FSTAT
```



```
P       reserve 1 area.

L+P     select FEMP assign to DA-SYSEP,
L       organization is indexed,
E       access mode is dynamic,
L       record key is NUM-EX of EMPRUNT,
L       alternate key is NUM-EMP of EMPRUNT with duplicates,
E       file status is FSTAT,
P       reserve 1 area.

L+P     select FEMPA assign to DA-SYSEA,
L       organization is sequential,
E       access mode is sequential,
E       file status is FSTAT,
P       reserve 1 area.
```

11.2.2 DATA DIVISION (Fichier BDMOD12)

```
L       fd FAUT label is omitted;
P       block contains 2048 characters;
L       data record is AUTEUR.
L          01 AUTEUR.
L             02 NUM-AUT
L             02 NOM
L             02 ORIGINE
L             02 N-OUV
L             02 NUM-OUV ... occurs 15 depending on N-OUV.

L       fd FOUVR label is omitted;
P       block contains 2048 characters;
L       data record is OUVRAGE.
L          01 OUVRAGE.
L             02 NUMERO
L             02 TITRE
L             02 ANNEE
L             02 EDITEUR
L             02 N-MOTS-CLES
L             02 MOT-CLE ... occurs 10 depending on N-MOTS-CLES.
L             02 N-AUTEURS
L             02 NUM-AUTEUR ... occurs 8 depending on N-AUTEURS.

L       fd FMOT  label is omitted;
P       block contains 2048 characters;
L       data record is MOTS-CLES-D-OUVR.
```

```
L          01 MOTS-CLES-D-OUV.
L             02 ID-MOT-CLE.
L                03 MOT-CLE
L                03 NUM-OUV

L       fd FEX   label is omitted;
P       block contains 2048 characters;
L       data record is EXEMPLAIRE.
L          01 EXEMPLAIRE.
L             02 NUMERO
L             02 LOCALISATION.
L                03 ETAGE
L                03 TRAVEE
L                03 RAYON
L             02 NUM-OUV

L       fd FEMPR label is omitted;
P       block contains 2048 characters;
L       data record is EMPRUNTEUR.
L          01 EMPRUNTEUR.
L             02 NUMERO
L             02 NOM
L             02 ADRESSE

L       fd FEMP  label is omitted;
P       block contains 2048 characters;
L       data record is EMPRUNT.
L          01 EMPRUNT.
L             02 NUM-EX
L             02 NUM-EMP
L             02 DATE-DEBUT

L       fd FEMPA label is omitted;
P       block contains 2048 characters;
L       data record is EMPRUNT-ARCH.
L          01 EMPRUNT-ARCH.
L             02 DATE-DEBUT
L             02 DATE-FIN
L             02 NUM-EX
L             02 NUM-EMP
```


11.3 SCHEMA SQL/DS DE LA BASE DE DONNEES "BIBLIO"

```
P    acquire public dbspace named DBSAUT
P                         ( pages = 8064,
P                           pctindex = 37,
P                           pctfree = 16                  )

P    acquire public dbspace named DBSOUV
P                         ( pages = 20096,
P                           pctindex = 20,
P                           pctfree = 15                  )
        .
        .
        .

P    acquire public dbspace named DBSEMPA
P                         ( pages = 7680,
P                           pctindex = 0,
P                           pctfree = 0                   )


L    create table AUTEUR  ( NUM_AUT    char(6) not null,
L                           NOM        char(35) not null,
L                           ORIGINE    char(50)           )
P                                                          in DBSAUT

L    create table OUVRAGE ( NUMERO     char(6) not null,
L                           TITRE      char(80) not null,
L                           ANNEE      char(6) not null,
L                           EDITEUR    char(45) not null  )
P                                                          in DBSOUV

L    create table EXEMPLAIRE ( NUMERO  char(8) not null,
L                              ETAGE   char(4) not null,
L                              TRAVEE  char(4) not null,
L                              RAYON   char(4) not null,
L                              NUM_OUV char(6) not null   )
P                                                          in DBSOUV

L    create table EMPRUNTEUR ( NUMERO  char(8) not null,
L                              NOM     char(35) not null,
L                              ADRESSE char(60) not null  )
P                                                         in DBSEMPR
```

```
L    create table EMPRUNT    ( NUM_EX  char(8) not null,
L                              NUM_EMP char(8) not null,
L                              DATE_DEBUT char(6) not null )
P                                                      in DBSEMP

L    create table EMPRUNT_ARCH ( DATE_DEBUT char(6) not null,
L                                DATE_FIN   char(6) not null,
L                                NUM_EX     char(8) not null,
L                                NUM_EMP    char(8) not null )
P                                                     in DBSEMPA

L    create table OUVAUT     ( NUM_OUV char(6) not null,
L                              NUM_AUT char(6) not null      )
P                                                      in DBSAUT

L    create table MOTS_CLES_D_OUV ( NUM_OUV char(6) not null,
L                                   MOT_CLE char(20) not null)
P                                                      in DBSMOT


L    create unique index IDXNUMAU
L                   on AUTEUR (NUM_AUT)
P                                     pctfree = 20

L    create unique index IDXNUMOUV
L                   on OUVRAGE (NUMERO)
P                                     pctfree = 20

L    create unique index IDXNUMEX
L                   on EXEMPLAIRE (NUMERO)
P                                     pctfree = 20

L    create unique index IDXNUMEMP
L                   on EMPRUNTEUR (NUMERO)
P                                     pctfree = 0

L    create unique index IDXEMP
L                   on EMPRUNT (NUM_EX)
P                                     pctfree = 20

L    create unique index IDXOUVAUT
L                   on OUVAUT (NUM_OUV, NUM_AUT)
P                                     pctfree = 20
L    create unique index IDXMOTCLE
L                   on MOTS_CLES_D_OUV (NUM_OUV, MOT_CLE)
```



```
P                                       pctfree = 20

P    create index  IDXNOM
P                  on AUTEUR (NOM)
P                                       pctfree = 20
```

Chapitre 12 : ANNEXE 2 : TEXTE DES PROGRAMMES COBOL DU MODULE 1

12.1 PROGRAMME COBOL DU MODULE 1 (Programme de niveau 3)

```
identification division.
program-id.  MODULE1.

data division.
working-storage section.

    01 OUVRIR-BIBLIO       pic 99 usage comp value 1.
    01 FERMER-BIBLIO       pic 99 usage comp value 2.
    01 AUTEUR-NOM-PREM     pic 99 usage comp value 3.
    01 AUTEUR-NOM-SUIV     pic 99 usage comp value 4.
    01 OUVRAGE-AUTEUR-PREM pic 99 usage comp value 5.
    01 OUVRAGE-AUTEUR-SUIV pic 99 usage comp value 6.
    01 EXEMPLAIRE-DE-PREM  pic 99 usage comp value 7.
    01 EXEMPLAIRE-DE-SUIV  pic 99 usage comp value 8.
    01 EMPRUNT-EXE         pic 99 usage comp value 9.
    01 TITRE-OUVRAGE       pic 99 usage comp value 10.
    01 ANNEE-OUVRAGE       pic 99 usage comp value 11.
    01 NUMERO-EXEMPLAIRE   pic 99 usage comp value 12.

    01 OK-B   pic 9(2) usage comp.
    01 OK-AU  pic 9(2) usage comp.
    01 OK-O   pic 9(2) usage comp.
    01 OK-EX  pic 9(2) usage comp.
    01 OK-EMP pic 9(2) usage comp.
    01 OK     pic 9(2) usage comp.

    01 TROUVE    pic X.
    01 SEMBLABLE pic X.
    01 EX-OK     pic X.
    01 TRUE      pic X value "T".
    01 FALSE     pic X value "F".
```



```
    01 XT  pic X(80).
    01 XAN pic 9(6).
    01 XAU pic X(35).

    01 CODE-ECRAN   pic X(7) value "001/001".
    01 ERREUR-ECRAN pic 9(2) usage comp.

    01 ANNEE  pic 9(6).
    01 TITRE  pic X(80).
    01 NUMERO pic 9(8).
    01 NUL    pic X.

procedure division.

*   OUVRIR-BIBLIO (OK-B);
    call "BD" using OUVRIR-BIBLIO, NUL, OK-B.
*   if (OK-B = 0) then
    if (OK-B = 0) perform CORPS-B.
*   endif;
    stop run.

 CORPS-B.
*       input(XT, XAN, XAU);
       call "LIRE-ECRAN" using CODE-ECRAN, XT, XAN, XAU, ERREUR-ECRAN.
*       TROUVE := false;
        move FALSE to TROUVE.
*       AUTEUR-NOM-PREM (XAU, OK-AU);
        call "BD" using AUTEUR-NOM-PREM, XAU, OK-AU.
*       while (OK-AU = 0) do
        perform CORPS-AU until not (OK-AU = 0).
*       endwhile;
*       FERMER-BIBLIO (OK-B);
        call "BD" using FERMER-BIBLIO, NUL, OK-B.

 CORPS-AU.
*           OUVRAGE-AUTEUR-PREM (OK-O);
            call "BD" using OUVRAGE-AUTEUR-PREM, NUL, OK-O.
*           while (OK-O = 0) do
            perform CORPS-O until not (OK-O = 0).
*           endwhile;
*           if TROUVE then exit;
            if TROUVE = TRUE move 1 to OK-AU
*              else AUTEUR-NOM-SUIV (OK-AU);
               else call "BD" using AUTEUR-NOM-SUIV, NUL, OK-AU.
*           endif;
```

```
 CORPS-O.
*                TITRE-OUVRAGE (TITRE, OK);
                 call "BD" using TITRE-OUVRAGE, TITRE, OK.
*                ANNEE-OUVRAGE (ANNEE, OK);
                 call "BD" using ANNEE-OUVRAGE, ANNEE, OK.
*                COMPARER (TITRE, XT, SEMBLABLE);
                 call "COMPARER" using TITRE, XT, SEMBLABLE.
*                if (SEMBLABLE and ANNEE = XAN) then
              if (SEMBLABLE = TRUE and ANNEE = XAN) perform OUVRAGE-OK.
*                endif;
*                if TROUVE then exit;
                 if TROUVE = TRUE move 1 to OK-O
*                   else OUVRAGE-AUTEUR-SUIV (OK-O);
                  else call "BD" using OUVRAGE-AUTEUR-SUIV, NUL, OK-O.
*                endif;

 OUVRAGE-OK.
*                   EXEMPLAIRE-DE-PREM (OK-EX);
                    call "BD" using EXEMPLAIRE-DE-PREM, NUL, OK-EX.
*                   while (OK-EX = 0) do
                    perform CORPS-EX until not (OK-EX = 0).
*                   endwhile;
 CORPS-EX.
*                      EX-OK := true;
                       move TRUE to EX-OK.
*                      EMPRUNT-EXE (OK-EMP);
                       call "BD" using EMPRUNT-EXE, NUL, OK-EMP.
*                      if (OK-EMP = 0) then EX-OK := false; endif;
                       if (OK-EMP = 0) move FALSE to EX-OK.
*                      if EX-OK then
                       if (EX-OK = TRUE) perform EXEMPLAIRE-OK.
*                      endif;
*                      if TROUVE then exit EX;
                       if (TROUVE = TRUE) move 1 to OK-EX
*                          else EXEMPLAIRE-DE-SUIV (OK-EX);
                           else call "BD" using EXEMPLAIRE-DE-SUIV,
NUL, OK-EX.
*                      endif;

 EXEMPLAIRE-OK.
*                         TROUVE := true;
                          move TRUE to TROUVE.
*                         NUMERO-EXEMPLAIRE (NUMERO, OK);
                        call "BD" using NUMERO-EXEMPLAIRE, NUMERO, OK.
*                         print NUMERO;
                        call "ECRIRE-ECRAN" using CODE-ECRAN, NUMERO,
```



```
                                                    ERREUR-ECRAN.
```

12.2 PROGRAMME COBOL/SQL BASE SUR UNE OPTIMISATION IMPLICITE

```
 identification division.
 program-id. MODULE1.

 data division.
 working-storage section.

    exec SQL begin declare section end-exec.
    01 XT  pic X(80).
    01 XAN pic 9(6).
    01 XAU pic X(35).
    01 ZTITRE  pic X(80).
    01 ZNUMERO-DE-O pic 9(6).
    01 ZNUMERO-DE-EX pic 9(8).
    exec SQL end declare section end-exec.

    exec SQL include SQLCA  end-exec.

    01 TROUVE    pic X.
    01 SEMBLABLE pic X.
    01 TRUE      pic X value "T".
    01 FALSE     pic X value "F".

    01 CODE-ECRAN   pic X(7) value "001/001".
    01 ERREUR-ECRAN pic 9(2) usage comp.

 procedure division.

*     input(XT, XAN, XAUT);
      call "LIRE-ECRAN" using CODE-ECRAN, XT, XAN, XAU, ERREUR-ECRAN.
*     TROUVE := false;
      move FALSE to TROUVE.
*     PREMIER-OUVRAGE ("OUVRAGE((:ANNEE=XAN) and
*                               (:NUMERO not-in ...))", O, OK-O);
```

```
      perform PREMIER-OUVRAGE.

 TEST-OUVRAGE.
*     if not (OK-O = 0) then goto FIN-OUVRAGE endif;
      if not (OK-O = 0) go to FIN-OUVRAGE.
*             COMPARER (TITRE(:O), XT, SEMBLABLE);
              call "COMPARER" using ZTITRE, XT, SEMBLABLE.
*             if not SEMBLABLE then goto FIN-SEMBLABLE endif;
              if not (SEMBLABLE = TRUE) go to FIN-SEMBLABLE.
*           PREMIER-EXEMPLAIRE ("EXEMPLAIRE((NUMERO not-in ...)", EX,
OK-EX);
              perform PREMIER-EXEMPLAIRE.
*             if not (OK-EX = 0) then goto FIN-EXEMPLAIRE endif;
              if not (OK-EX = 0) go to FIN-EXEMPLAIRE.
*                   TROUVE := true;
                    move TRUE to TROUVE.
*                   print NUMERO(:EX);
                    call "ECRIRE-ECRAN" using CODE-ECRAN, NUMERO,
                                                          ERREUR-ECRAN.
 FIN-EXEMPLAIRE.

 FIN-SEMBLABLE.
*             if TROUVE then goto FIN-OUVRAGE endif;
              if (TROUVE = TRUE) go to FIN-OUVRAGE.
*             OUVRAGE-SUIVANT("OUVRAGE((:ANNEE=XAN) and
*                                 (:NUMERO not-in ...))", O, OK-O);
              perform OUVRAGE-SUIVANT.
*             goto TEST-OUVRAGE;
              go to TEST-OUVRAGE.

 FIN-OUVRAGE.

      stop run.


 PREMIER-OUVRAGE.

*     O := OUVRAGE ( (: ANNEE = XAN)
*                 and (: NUMERO in NUM-OUV(: OUVAUT (: NUM-AUT
*                             in NUM-AUT (: AUTEUR (: NOM = XAUT))))) )

      exec SQL declare O cursor for
           select TITRE, NUMERO
           from   OUVRAGE
           where  ANNEE = :XAN
           and    NUMERO in select NUM_OUV
```



```
                          from   OUVAUT
                          where  NUM_AUT in select NUM_AUT
                                            from   AUTEUR
                                            where  NOM = :XAUT
      end-exec.

      exec SQL open O end-exec.

      exec SQL fetch O into :ZTITRE, :ZNUMERO-DE-O end-exec.

      if SQLCODE = 0 then move 0 to OK-O
                     else if SQLCODE = 100 then move 1 to OK-O
                                           else move ... .
      if not (SQLCODE = 0)  then
                            exec SQL close O end-SQL.


 OUVRAGE-SUIVANT.

      exec SQL fetch O into :ZTITRE, :ZNUMERO-DE-O end-exec.

      if SQLCODE = 0 then move 0 to OK-O
                     else if SQLCODE = 100 then move 1 to OK-O
                                           else move ... .
      if not (SQLCODE = 0)  then
                            exec SQL close O end-SQL.

 PREMIER-EXEMPLAIRE.

*     EX := EXEMPLAIRE ((:NUMERO not-in NUM-EX (: EMPRUNT))
*                    and (:NUM-OUV = NUMERO(: O)))

      exec SQL declare EX cursor for
               select NUMERO
               from   EXEMPLAIRE
               where  NUMERO not in (select NUM_EX
                                     from EMPRUNT  )
               and    NUM_OUV  = :ZNUMERO-DE-O
      end-exec.

      exec SQL open EX end-exec.

      exec SQL fetch EX into :ZNUMERO-DE-EX end-exec.

      if SQLCODE = 0 then move 0 to OK-O
                     else if SQLCODE = 100 then move 1 to OK-O
```

```
                                                    else move ... .
exec SQL close EX end-exec.
```

Chapitre 13 : ANNEXE 3 : TEXTE DES MODULES D'ACCES DE NIVEAU 3 POUR LE MODULE 1

13.1 PROGRAMME COBOL/CODASYL DU MODULE D'ACCES (niveau 3) DU MODULE 1

```
identification division.
    program-id.  BD.

data division.
schema section.
    invoke SS1 of schema BIBLIO.
linkage section.
    01 CODE-OP   pic 9(2) usage comp.
    01 CHAMPS    pic X(80).
    01 CH-NOM    redefines CHAMPS.
       02 NOM    pic X(35).
       02 filler pic X(45).
    01 CH-TITRE  redefines CHAMPS.
       02 TITRE  pic X(80).
    01 CH-ANNEE  redefines CHAMPS.
       02 ANNEE  pic 9(6).
       02 filler pic X(74).
    01 CH-NUMERO redefines CHAMPS.
       02 NUMERO pic 9(8).
       02 filler pic X(72).
    01 OK pic 9(2) usage comp.

procedure division using CODE-OP, CHAMPS, OK.

    if CODE-OP < 1 or CODE-OP > 12 move 5 to OK, go to SORTIE.
    move 0 to OK.
    go to OUVRIR, FERMER, AUTEUR-P, AUTEUR-S, OUVR-P, OUVR-S, EX-P,
EX-S,
          EMPRUNT, TITRE, ANNEE, NUMERO  depending on CODE-OP.
```

```
OUVRIR.
    open ABIBLIO usage-mode is protected retrieval.
    go to CLOTURE.

FERMER.
    close ABIBLIO.
    go to CLOTURE.

AUTEUR-P.
    move NOM of CH-NOM to NOM of AUTEUR.
    find AUTEUR record.
    go to CLOTURE.

AUTEUR-S.
    find current of AUTEUR record.
    find next duplicate within AUTEUR record.
    go to CLOTURE.

OUVR-P.
    find first OUVAUT record of AO set.
TEST-AO.
    if not (error-count = 0) go to FIN-AO.
         find owner of OO set.
         get OUVRAGE; TITRE, ANNEE.
         go to CLOTURE.

OUVR-S.
         find next OUVAUT record of AO set.
         go to TEST-AO.
FIN-AO.
    move 1 to 0K.
    go to SORTIE.

EX-P.
    find first EXEMPLAIRE record of DE set.
    get EXEMPLAIRE; NUMERO.
    go to CLOTURE.

EX-S.
    find next EXEMPLAIRE record of DE set.
    get EXEMPLAIRE; NUMERO.
    go to CLOTURE.

EMPRUNT.
    find first EMPRUNT record of EXE set.
    go to CLOTURE.
```



```
TITRE.
    move TITRE of OUVRAGE to TITRE of CH-TITRE.
    go to SORTIE.

ANNEE.
    move ANNEE of OUVRAGE to ANNEE of CH-ANNEE.
    go to SORTIE.

NUMERO.
    move NUMERO of EXEMPLAIRE to NUMERO of CH-NUMERO.
    go to SORTIE.

CLOTURE.
    if error-count > 0 move 1 to OK.
SORTIE.
    exit program.
```

13.2 PROGRAMME COBOL/COBOL DU MODULE D'ACCES (niveau 3) DU MODULE 1

```
identification division.
    program-id.  BD.

environment division.
input-output section.
file-control.
    copy BDMOD11.

data division.
file section.
    copy BDMOD12.

working-storage section.
    01 FSTAT pic X(2).

linkage section.
    01 CODE-OP   pic 9(2) usage comp.
    01 CHAMPS    pic X(80).
```

```
    01 CH-NOM    redefines CHAMPS.
       02 NOM    pic X(35).
       02 filler pic X(45).
    01 CH-TITRE  redefines CHAMPS.
       02 TITRE  pic X(80).
    01 CH-ANNEE  redefines CHAMPS.
       02 ANNEE  pic 9(6).
       02 filler pic X(74).
    01 CH-NUMERO redefines CHAMPS.
       02 NUMERO pic 9(8).
       02 filler pic X(72).
    01 OK pic 9(2) usage comp.

procedure division using CODE-OP, CHAMPS, OK.

declaratives.
DUMMY section. use after exception procedure on FAUT FOUVR FEX FEMP.
end declaratives.

MAIN section.

    if CODE-OP < 1 or CODE-OP > 12 move 5 to OK, go to SORTIE.
    move 0 to OK.
    go to OUVRIR, FERMER, AUTEUR-P, AUTEUR-S, OUVR-P, OUVR-S, EX-P,
EX-S,
          EMPRUNT, TITRE, ANNEE, NUMERO  depending on CODE-OP.

OUVRIR.
    open input FAUT.
    open input FOUVR.
    open input FEX.
    open input FEMP.
    go to CLOTURE.

FERMER.
    close FAUT.
    close FOUVR.
    close FEX.
    close FEMP.
    go to CLOTURE.

AUTEUR-P.
    move NOM of CH-NOM to NOM of AUTEUR.
    read FAUT key is NOM of AUTEUR.
    go to CLOTURE.
```



```
AUTEUR-S.
    read FAUT next.
    go to CLOTURE.

OUVR-P.
    move 1 to IAO.
TEST-IAO.
    if not (IAO <= N-OUV(:AU)) go to FIN-IAO.
         move NUM-OUV of AUTEUR (IAO) to NUMERO of OUVRAGE.
         read FOUVR key is NUMERO of OUVRAGE.
         go to CLOTURE.

OUVR-S.
         add 1 to IAO.
         goto TEST-IAO.
FIN-IAO.
    move 1 to OK.
    go to SORTIE.

EX-P.
    move NUMERO of OUVRAGE to NUM-EX of EXEMPLAIRE.
    read FEX key is NUM-EX of EXEMPLAIRE.
    go to CLOTURE.

EX-S.
    read FEX next.
    go to CLOTURE.

EMPRUNT.
    move NUMERO of EXEMPLAIRE to NUM-EX of EMPRUNT.
    read FAUT key is NUM-EX of EXEMPLAIRE.
    go to CLOTURE.

TITRE.
    move TITRE of OUVRAGE to TITRE of CH-TITRE.
    go to SORTIE.

ANNEE.
    move ANNEE of OUVRAGE to ANNEE of CH-ANNEE.
    go to SORTIE.

NUMERO.
    move NUMERO of EXEMPLAIRE to NUMERO of CH-NUMERO.
    go to SORTIE.

CLOTURE.
```



```
    if FSTAT = "00" or "02" move 0 to OK go to SORTIE.
    if FSTAT = "10" or "23" move 1 to OK else move 10 to OK.
SORTIE.
    exit program.
```

13.3 PROGRAMME COBOL/SQL DU MODULE D'ACCES (niveau 3) DU MODULE 1

```
identification division.
    program-id.  BD.

data division.
working-storage section.

    exec SQL begin declare section end-exec.
    01 IDENT pic X(8) value "USER1   ".
    01 PASS  pic X(8) value "PW1     ".
    01 XNOM pic X(35).
    01 XNUM-AUT pic 9(6).
    01 XNUM-OUV pic 9(6).
    01 XNUMERO  pic 9(6).
    01 XTITRE   pic X(80).
    01 XANNEE   pic 9(6).
    01 ZNUM-EX  pic 9(6).
    exec SQL end declare section end-exec.

    exec SQL include SQLCA  end-exec.


linkage section.
    01 CODE-OP   pic 9(2) usage comp.
    01 CHAMPS    pic X(80).
    01 CH-NOM    redefines CHAMPS.
       02 NOM    pic X(35).
       02 filler pic X(45).
    01 CH-TITRE  redefines CHAMPS.
       02 TITRE  pic X(80).
    01 CH-ANNEE  redefines CHAMPS.
       02 ANNEE  pic 9(6).
       02 filler pic X(74).
    01 CH-NUMERO redefines CHAMPS.
```



```
       02 NUMERO pic 9(8).
       02 filler pic X(72).
    01 OK pic 9(2) usage comp.

procedure division using CODE-OP, CHAMPS, OK.

declaratives.
DUMMY section. use after exception procedure on FAUT FOUVR FEX FEMP.
end declaratives.

MAIN section.

    if CODE-OP < 1 or CODE-OP > 12 move 5 to OK, go to SORTIE.
    move 0 to OK.
    go to OUVRIR, FERMER, AUTEUR-P, AUTEUR-S, OUVR-P, OUVR-S, EX-P,
EX-S,
          EMPRUNT, TITRE, ANNEE, NUMERO  depending on CODE-OP.

OUVRIR.
    exec SQL  connect :IDENT identified by :PASS  end-exec.
    go to CLOTURE.

FERMER.
    go to CLOTURE.

AUTEUR-P.
    move NOM of CH-NOM to XNOM.
    exec SQL declare AU cursor for
         select NUM_AUT
         from   AUTEUR
         where  NOM = :XNOM
    end-exec.
    exec SQL  open AU  end-exec.
    exec SQL  fetch AU into :XNUM-AUT  end-exec.
    if SQLCODE = 0 go to CLOTURE.
    exec SQL  close AU  end-exec.
    go to CLOTURE.

AUTEUR-S.
    read FAUT next.
    exec SQL  fetch AU into :XNUM-AUT  end-exec.
    if SQLCODE = 0 go to CLOTURE.
    exec SQL  close AU  end-exec.
    go to CLOTURE.

OUVR-P.
```

```
    exec SQL declare AO cursor for
         select NUM_OUV
         from   OUVAUT
         where  NUM_AUT = :XNUM-AUT
    end-exec.
    exec SQL  open AO  end-exec.
    exec SQL  fetch AO into :XNUM-OUV  end-exec.
    if SQLCODE = 0 go to TEST-AO.
    exec SQL  close AO  end-exec.
TEST-AO.
    if not (SQLCODE = 0) go to FIN-AO.
        exec SQL select NUMERO, TITRE, ANNEE
                 into  :XNUMERO, :XTITRE, :XANNEE
                 from   OUVRAGE
                 where  NUMERO = :XNUM-OUV
        end-exec.
        go to CLOTURE.
OUVR-S.
        exec SQL  fetch AO into :XNUM-OUV  end-exec.
        if SQLCODE = 0 go to TES-AO.
        exec SQL  close AU  end-exec.
        go to TEST-AO.
FIN-AO.
    move 1 to OK.
    go to SORTIE.

EX-P.
    exec SQL declare cursor EX for
         select NUMERO
         from   EXEMPLAIRE
         where  NUM_OUV = :XNUMERO
    end-exec.
    exec SQL  open EX  end-exec.
    exec SQL  fetch EX into :XNUM-EX  end-exec.
    if SQLCODE = 0 go to CLOTURE.
    exec SQL  close EX  end-exec.
    go to CLOTURE.

EX-S.
    exec SQL  fetch EX into :XNUM-EX  end-exec.
    if SQLCODE = 0 go to CLOTURE.
    exec SQL  close EX  end-exec.
    go to CLOTURE.

EMPRUNT.
    exec SQL select NUM_EX
```



```
            into  :XNUM-EX
            from   EMPRUNT
            where  NUM_EX = :XNUM-EX
    end-exec.
    go to CLOTURE.


TITRE.
    move XTITRE to TITRE of CH-TITRE.
    go to SORTIE.

ANNEE.
    move XANNEE to ANNEE of CH-ANNEE.
    go to SORTIE.

NUMERO.
    move XNUM-EX to NUMERO of CH-NUMERO.
    go to SORTIE.

CLOTURE.
    if SQLCODE = 0   move 0 to OK go to SORTIE.
    if SQLCODE = 100 move 1 to OK else move 10 to OK.
SORTIE.
    exit program.
```