MASTER EN SCIENCES INFORMATIQUES

Contribution à l'atelier logiciel de conception de bases de données
Étude de transformations de schémas

Charlot, C.; Müller, I.

Award date:
1986

Awarding institution:
Universite de Namur

FACULTÉS
UNIVERSITAIRES
N. D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

ANNEE ACADEMIQUE 1985-1986

CONTRIBUTION A L'ATELIER LOGICIEL DE CONCEPTION DE BASES DE DONNEES : ETUDE DE TRANSFORMATIONS DE SCHEMAS

PROMOTEUR : J-L. HAINAUT

Mémoire présenté par
Charlot C. et Müller I.
en vue de l'obtention du titre
de Licencié et Maître en
Informatique

RUE GRANDGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Nous tenons tout d'abord à remercier Monsieur J-L. Hainaut, promoteur de ce mémoire, pour l'intérêt qu'il a porté au sujet traité et pour l'aide qu'il nous a constamment prodiguée, aussi bien lors de l'étude que lors de la rédaction.

Nous exprimons également notre profonde gratitude à Monsieur F. Bodart pour ses conseils et critiques qui ont permis d'améliorer la présentation de ce texte.

Notre reconnaissance s'adresse aussi à Monsieur M. Leonard et son équipe pour l'acceuil qu'ils nous ont réservé au sein du Centre Universitaire d'Informatique à Genève, ainsi que pour nous avoir éclairci sur quelques notions dans le domaine de la conception de bases de données.

Enfin nous remercions Messieurs A. Delcourt et B. Van Houtte et tous les membres de l'Institut d'Informatique qui, d'une manière ou d'une autre, nous ont aidées au cours de cette année.

TABLE DES MATIERES.

INTRODUCTION.

Le mémoire se situe dans le cadre du développement d'un atelier logiciel de conception de Bases de Données (BD). Il s'y intègre comme outil d'aide à la réalisation de transformations de structures de données afin de traduire un schéma conceptuel des données en un schéma exécutable par un système de gestion de fichiers ou de bases de données (que nous appellerons en toute généralité Systèmes de Gestion de données ou SGD). L'atelier doit en particulier permettre la mise en oeuvre de la démarche de conception de bases de données qui a été développée par J-L HAINAUT, Professeur à l'Institut d'Informatique de Namur, ainsi que des modèles qui la supportent (HAINAUT,86a).

Ces modèles et les différents processus de la démarche présentant une certaine complexité, il nous a paru utile de les aborder et de les justifier au travers d'une étude de cas présentée de manière informelle.

UNE ETUDE DE CAS INFORMELLE.

Dans cette étude on présentera de façon intuitive la conception d'une base de données représentant les informations nécessaires à une université pour la gestion des travaux effectués par ses étudiants. On demande de réaliser cette base de données selon un Système de Gestion de Bases de Données CODASYL et selon un SGBD relationnel (en l'occurrence SQL).

Ces informations incluront notamment le type du travail (laboratoire, mémoire), son titre (supposé identifiant),l'année de réalisation ainsi que le nom du professeur promoteur (également supposé identifiant). Si le travail est de type mémoire, des renseignements seront donnés sur le stage lui-même tels que le lieu de stage, le nom et la profession du maitre de stage. De l'étudiant on connait le numéro identifiant, les nom et prénoms, la date de naissance et l'adresse du domicile légal.

Le schéma conceptuel de ce Système d'Informations exprimé selon les concepts du modèle Entité/Association (E/A) pourrait être le suivant :

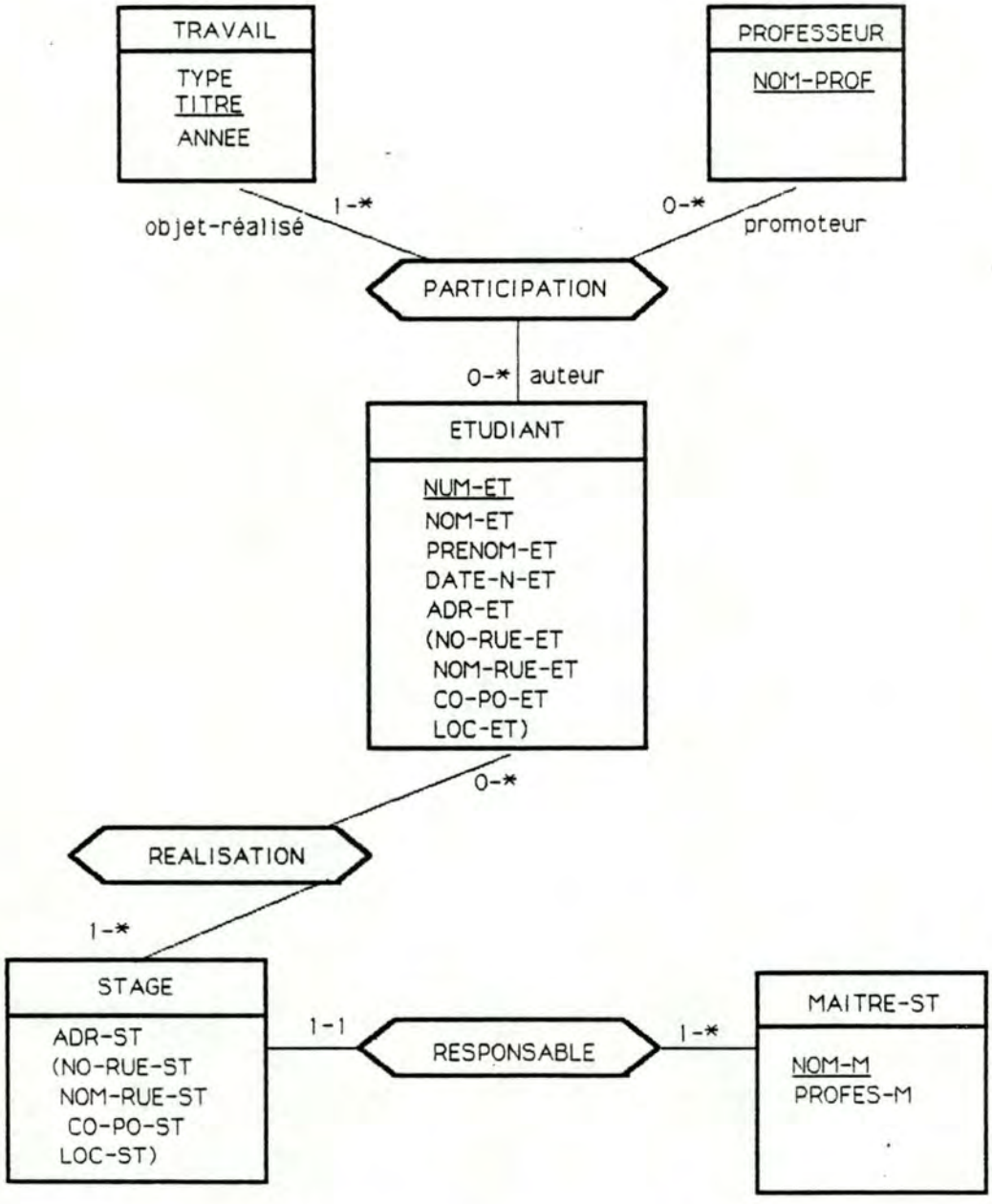


Figure 0.1: Schéma conceptuel des données.

Dans le cadre de l'application de gestion des travaux des étudiants, on envisage de réaliser les fonctions suivantes (il y en a certainement d'autres, que nous ignorons ici) :

fonction 1 : étant donnés le NOM-PROF d'un PROFESSEUR et une ANNEE scolaire, trouver tous les TITREs des TRAVAUX, classés par TYPE, pour lesquels ce PROFESSEUR est promoteur. On désirera également connaitre les informations de base concernant les ETUDIANTs réalisateurs de ces TRAVAUX.

fonction 2 : étant donné le TITRE d'un TRAVAIL de TYPE 'mémoire', trouver les noms des promoteurs et du MAITRE-ST, le lieu de stage, ainsi que les NOM-ET, PRENOM-ET et DATE-N-ET de chaque ETUDIANT auteur.

ELABORATION D'UNE BASE DE DONNEES CODASYL.

Une base de données CODASYL est constituée d'un ensemble de RECORDs appartenant chacun à un type (RECORD-TYPE). Ces RECORDs peuvent souvent être assimilés aux entités du modèle E/A.

A un RECORD d'un type sont généralement associées des valeurs de DATA-ITEMs qui peuvent être élémentaires ou décomposables, simples ou répétitifs.

Un DATA-ITEM (ou groupe de DATA-ITEMs) peut constituer une clé d'accès (CALC-KEY) aux RECORD-TYPEs. Il n'est cependant pas permis de définir plus d'une clé d'accès par RECORD-TYPE. Une clé d'accès peut être identifiante ou non.

Entre les RECORD-TYPEs peuvent être définis des SET-TYPEs. Chaque SET est un groupe constitué d'un RECORD appelé OWNER et de 0, 1 ou plusieurs RECORDs appelés MEMBERs. Les SET-TYPES correspondent plus ou moins à des types d'associations binaires sans attributs, de classe fonctionnelle 1-N, permettant d'accéder aux RECORDs d'un membre à partir des articles de l'autre.

Ces concepts étant rappelés, il nous faut maintenant adapter le schéma conceptuel aux particularités de CODASYL et ajouter les accès aux structures de données en fonction des besoins des futures applications.

Analysons d'abord les besoins en accès.

Ces derniers seront dérivés intuitivement des structures générales des futurs programmes qui peuvent s'exprimer comme suit :

pour la fonction 1 :

```
pour chaque PROFESSEUR p dont le NOM-PROF est spécifié
     accéder à tous les TRAVAUX t dont p est promoteur
     dans une année donnée
          imprimer le TITRE de t
          pour chaque ETUDIANT e auteur de t
               imprimer NUM-ET, NOM-ET et PRENOM-ET de e.
```

pour la fonction 2 :

```
pour chaque TRAVAIL t de TYPE 'mémoire' dont le TITRE est spécifié
     accéder à tous les PROFESSEURs p promoteurs de t
          imprimer le NOM-PROF de p
     accéder à tous les ETUDIANTs e auteurs de t
          imprimer NUM-ET, NOM-ET et PRENOM-ET de e
          accéder à chaque STAGE st réalisé par e
               imprimer ADR-ST de st
               accéder au MAITRE-STAGE m-st responsable de st
                    imprimer NOM-MAITRE de m-st.
```

Ces algorithmes suggèrent l'utilité des accès suivants :

pour la fonction 1 :

- un accès à PROFESSEUR par une clé d'accès sur NOM-PROF;
- un accès de PROFESSEUR vers TRAVAIL via PARTICIPATION;
- un accès à TRAVAIL par une clé d'accès ANNEE;
- un accès de TRAVAIL vers ETUDIANT via PARTICIPATION.

pour la fonction 2 :

- un accès à TRAVAIL par une clé d'accès TITRE;
- un accès à TRAVAIL par une clé d'accès TYPE;
- un accès de TRAVAIL vers PROFESSEUR via PARTICIPATION;
- un accès de TRAVAIL vers ETUDIANT via PARTICIPATION;
- un accès de ETUDIANT vers STAGE via REALISATION;
- un accès de STAGE vers MAITRE-ST via RESPONSABLE.

Signalons qu'une méthode est définie dans (HAINAUT,86a) permettant une détermination intuitive et rigoureuse des structures d'accès strictement nécessaires à une exécution efficace des programmes.

Une fois connus ces besoins en termes d'accès, il reste à adapter le schéma conceptuel aux particularités de CODASYL de manière à conserver les informations initiales tout en satisfaisant aux besoins en accès évoqués ci-dessus.

Adaptation du schéma conceptuel aux particularités CODASYL.

Le type d'associations PARTICIPATION défini sur trois types d'entités devra être transformé en un RECORD-TYPE et trois SET-TYPEs. Dans chacun de ces SET-TYPEs le nouveau RECORD-TYPE jouera le rôle de MEMBER.

Le type d'associations REALISATION de classe fonctionnelle "Plusieurs-à-Plusieurs" sera transformé de la même façon que PARTICIPATION.

En ce qui concerne les accès, tout SET-TYPE sera bidirectionnel. CODASYL n'admettant pas plus d'une clé d'accès par RECORD-TYPE, deux des clés d'accès définies sur TRAVAIL devront être transformées. La clé d'accès TITRE étant identifiante, nous choisissons de la conserver. L'accès de ANNEE vers TRAVAIL se fera par l'intermédiaire d'un nouveau RECORD-TYPE (soit E-ANNEE) dans lequel ANNEE sera clé d'accès identifiante. La clé d'accès TYPE sera transformée de façon similaire.

En adaptant le schéma conceptuel aux particularités de CODASYL et aux transformations choisies, nous obtenons un schéma CODASYL qui peut s'exprimer de la manière suivante :

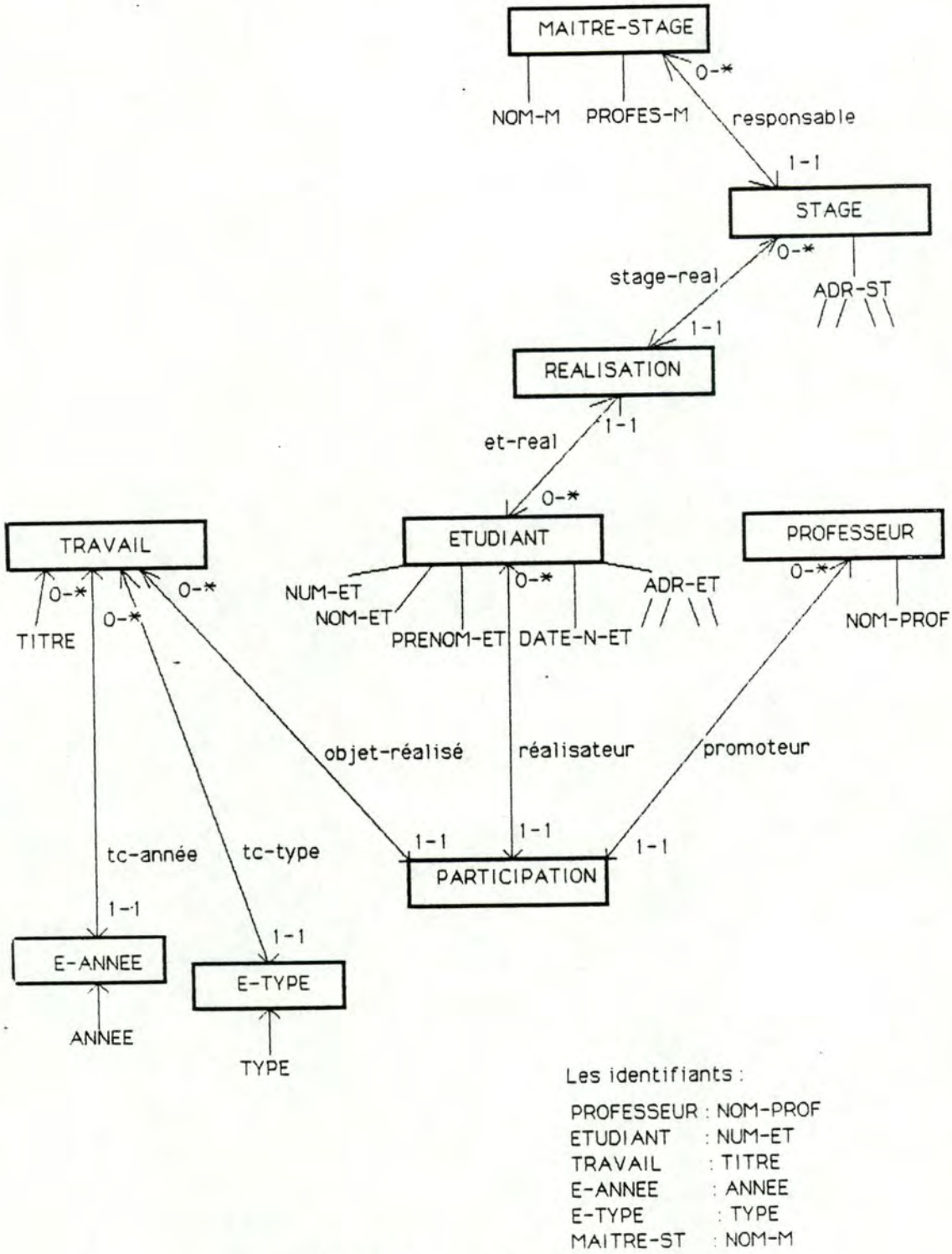


Figure 0.2: Schéma CODASYL.

Notons que ce schéma n'est certes pas complet. En effet, il devrait être accompagné des contraintes d''intégrité telles que, en l'occurrence, la contrainte spécifiant que le RECORD-TYPE PARTICIPATION est identifié par le groupe de SET-TYPEs (OBJET-REALISE,REALISATEUR,PROMOTEUR).

Il est évident que les algorithmes définis auparavant devront être adaptés aux transformations effectuées sur le schéma. Ne nous intéressant pas dans ce mémoire au développement des programmes, nous n'entrons pas dans ce sujet.

Le schéma traduit dans le langage de description de données propre à CODASYL serait alors le suivant :

```
schema name is GES-MEM.

record name is ETUDIANT
       location calc using NUM-ET, duplicates not allowed.
       2  NUM-ET pic 9(6).
       2  NOM-ET pic X(30).
       2  ...
 ...
set name is REALISATEUR
       owner ETUDIANT
       member PARTICIPATION mandatory automatic
       set selection thru current.
 ...
```

ELABORATION D'UNE BASE DE DONNEES SQL.

En SQL, les données sont présentées sous forme de tables qui peuvent souvent être assimilées aux types d'entités. Une table est constituée d'un nombre fixe de colonnes et de lignes en nombre variable. L'intersection d'une ligne et d'une colonne constitue une donnée élémentaire qui peut être assimilée à un attribut non décomposable et non répétitif. Sur chaque table peut être déclaré un nombre quelconque d'index (clé d'accès, identifiante ou non), constitué chacun d'un nombre quelconque de colonnes.

En adaptant le schéma conceptuel de notre cas à ces particularités tout en gardant les accès nécessaires à l'exécution des deux fonctions définies, nous obtenons un schéma SQL.

La spécification des accès étant la même que celle qui a été donnée pour CODASYL, nous pouvons passer directement à l'élaboration du schéma SQL.

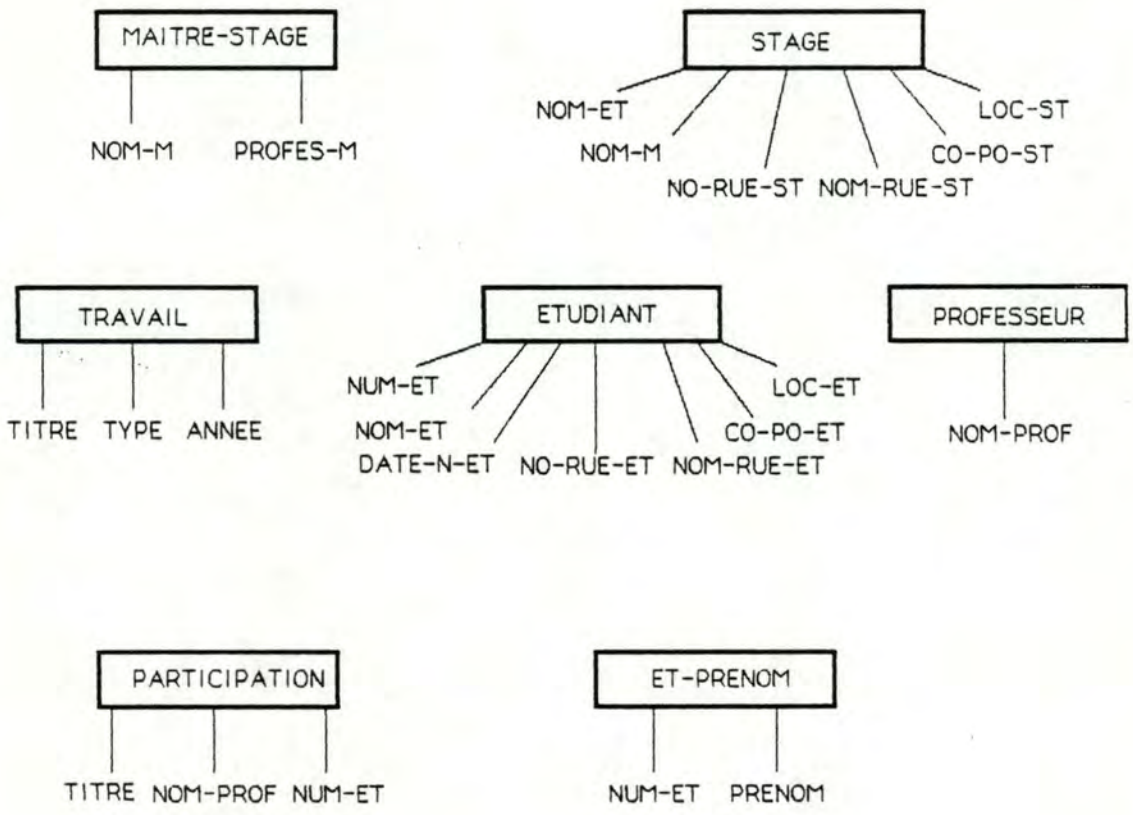
Ce SGD ignorant la notion de type d'associations, tous les types d'associations du schéma conceptuel devront être éliminés.

Le type d'associations PARTICIPATION sera d'abord transformé comme dans le cas CODASYL, c'est-à-dire en un type d'entités et trois types d'associations binaires. Ces derniers seront ensuite éliminés en copiant les identifiants de TRAVAIL, PROFESSEUR et ETUDIANT dans le nouveau type d'entités qui sera de plus identifié par l'ensemble de ces nouveaux attributs.

Le type d'associations REALISATION sera éliminé par le même procédé en copiant dans STAGE l'attribut NOM-ET de ETUDIANT. L'attribut NOM-M de MAITRE-ST sera également copié dans STAGE pour supprimer le type d'associations RESPONSABLE.

Puisque SQL n'admet pas que les attributs soient décomposables, ADR-ST et ADR-ET devront être aplatis. Les attributs répétitifs n'étant pas non plus admis, PRENOM-ET devra être remplacé par un nouveau type d'entités (soit ET-PRENOM), d'attribut non répétitif PRENOM, et un nouveau type d'associations TC-PRENOM qui sera ensuite éliminé en copiant l'attribut NUM-ET dans ET-PRENOM.

En adaptant le schéma conceptuel aux particularités de SQL et aux transformations choisies, nous obtenons un schéma SQL qui peut s'exprimer de la manière suivante :



Les identifiants :

PARTICIPATION	: (TITRE,NOM-PROF,NUM-ET)
MAITRE-ST	: NOM-M
TRAVAIL	: TITRE
PROFESSEUR	: NOM-PROF
ETUDIANT	: NUM-ET

Figure 0.3: Schéma SQL.

Le schéma traduit dans le langage de description de données propre à SQL est le suivant :

```
create table ETUDIANT   ( NUM-ET integer not null,
                          NOM-ET char(30),
                          PRENOM-ET char(30),
                          ...                    )
create table PROFESSEUR ( ...                    )
 ...
```

Les index sont à déclarer ici tout comme les CALC-KEY CODASYL dérivées de l'analyse des besoins des applications.

```
create unique index IDX-ET on ETUDIANT (NUM-ET).
 ...
```

CONCLUSION.

L'exemple analysé ci-dessus montre que certaines opérations telles que la spécification des mécanismes d'accès sont communes à CODASYL et SQL, et que les autres opérations sont similaires. Cette constatation peut de plus être généralisée à un grand nombre de SGDs. Il apparait dès lors intéressant d'introduire une phase intermédiaire (qui sera appelée phase de conception logique), entre l'analyse conceptuelle et la conception de la base de données réelle, qui utilisera un modèle de représentation des structures de données indépendant d'un SGD particulier. Les concepts de ce modèle devront cependant être proches de ceux intrinsèques aux SGDs et permettre notamment de spécifier les mécanismes d'accès aux structures de données.

L'exemple montre aussi la nécessité d'une démarche méthodologique de conception de BDs dont les étapes clés successives pourraient être les suivantes :

- construction du schéma conceptuel;
- analyse des besoins en termes d'accès;
- synthèse de ces besoins dans un schéma indépendant du SGD;
- particularisation du schéma en fonction des caractéristiques propres au SGD choisi;
- expression du schéma particularisé au SGD dans son langage de description de données (DDL).

L'exemple montre de plus la nécessité de trouver des règles de transformation de schémas qui, selon la démarche proposée, devront permettre d'une part, de produire un premier schéma MAG à partir des spécifications conceptuelles, d'autre part, de rendre un schéma conforme aux particularités d'un SGD et finalement, de traduire ce dernier schéma en un texte exécutable par le SGD. De plus, pour la conception de BDs en vraie grandeur, apparait rapidement la nécessité de s'appuyer sur des outils qui permettront d'enregistrer les spécifications successives et qui aideront le concepteur à opérer des transformations.

DEFINITION DU MEMOIRE.

La première partie de ce travail a pour but d'introduire le cadre général dans lequel s'inscrit le mémoire. Une description relativement sommaire du modèle conceptuel E/A et plus détaillée d'un modèle de spécification de structures de données et d'accès (MAG) fera l'objet des deux premiers chapitres. Le troisième chapitre sera consacré à l'exposé de la démarche de conception d'une BD. Une brève description de l'atelier logiciel support de la démarche sera donnée au chapitre 4.

La seconde partie de ce travail étudiera les principes de transformation de structures de données. Le chapitre 5 s'attachera à la production d'un premier schéma MAG à partir d'un schéma conceptuel exprimé dans le modèle E/A. La notion de conformité sera définie dans le sixième chapitre et sera particularisée pour trois SGDs généralement considérés comme concurrents : un SGBD CODASYL, un SGBD relationnel (SQL), et un simple Système de Gestion de Fichiers (COBOL). Les transformations sur le MAG constitueront le coeur de ce mémoire et feront l'objet du chapitre 7. Dans une étude théorique nous présenterons un ensemble assez vaste de transformations élémentaires assurant l'équivalence sémantique et d'accès. Les transformations particulièrement utiles au concepteur feront l'objet d'une étude détaillée. Dans le chapitre 8 nous aborderons brièvement le problème de la traduction d'un schéma conforme à un SGD en un texte exécutable par ce SGD.

Dans une troisième partie seront données quelques propositions de mise en oeuvre d'un outil d'aide à la réalisation des transformations. Le chapitre 9 s'intéressera à rechercher une structuration adéquate des transformations en vue d'ajouter à la fonction d'assistance de l'outil une fonction de guidance. Une liste structurée de toutes les restrictions possibles d'un SGD par rapport au MAG sera donnée dans le dixième chapitre. Le chapitre 11 donnera quelques éléments concernant l'architecture physique de l'outil qui restera à développer et l'étude détaillée d'un vérificateur de conformité paramétrable fera l'objet du chapitre 12.

En guise de conclusion nous donnerons une évaluation de l'analyse effectuée par rapport aux objectifs fixés et quelques perspectives intéressantes pour ce projet.

Pour éviter de lasser le lecteur par des détails d'analyse des transformations, nous n'en donnerons ici qu'un résumé et nous nous exprimerons non pas en fonction des concepts de l'atelier logiciel mais selon les termes plus généraux des modèles E/A et MAG. Le lecteur intéressé par une étude plus précise et exhaustive est reporté aux annexes ainsi qu'aux documents de travail préliminaires (CHAR-MUL,85a-d). L'approche théorique et rigoureuse des transformations ayant été effectuée dans les documents de travail, les annexes se limiteront aux cas existant en pratique sans apporter une justification complète des restrictions apportées à l'application des transformations. Ces justifications pourront être trouvées dans les documents de travail. De plus, les annexes restent indépendantes du schéma de l'atelier. En ce qui concerne les documents de travail, ceux-ci résultant d'une première approche des transformations, ils ne prennent pas en compte les transformations découvertes lors d'étapes d'analyse ultérieures et s'attardent sur des transformations qui, n'assurant pas l'équivalence sémantique, n'étaient pas à considérer.

PARTIE I

LE CADRE GENERAL

INTRODUCTION.

Cette première partie a pour but de définir l'environnement dans lequel seront étudiées les transformations permettant de passer d'une spécification conceptuelle à une solution opérationnelle, correcte et efficace.

La démarche de conception d'une base de données développée à l'Institut d'Informatique de Namur s'inspire directement d'une hiérarchie traditionnelle dont les trois niveaux successifs sont :

- l'analyse conceptuelle,
- la conception logique et
- la conception physique.

Chaque phase, jusqu'à présent supportée à l'Institut par ses propres modèles, est décomposée en plusieurs étapes, chacune caractérisée par des sous-produits spécifiques.

La figure suivante montre le schéma général de la démarche(a):

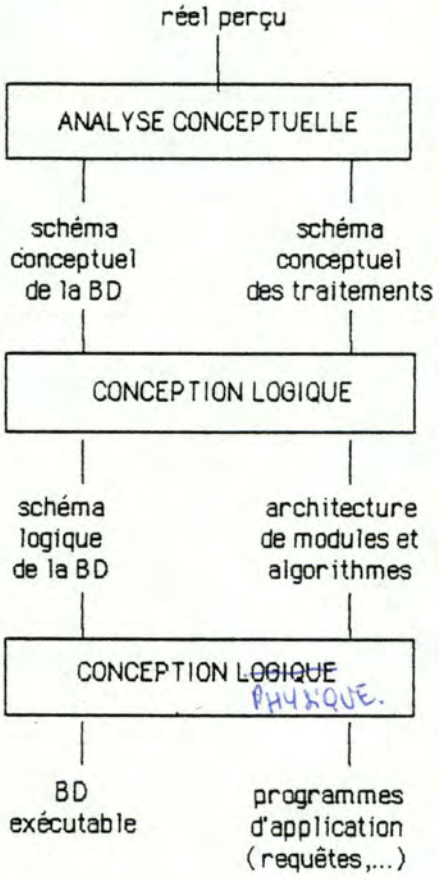


Figure 0.3: Schéma général de la démarche.

(a) Dans un premier temps nous ignorerons le fait que la linéarité de la démarche n'est qu'apparente.

Dans le cadre du mémoire, les transformations seront étudiées uniquement sous l'aspect des données. En guise d'extension, devrait être procédé ultérieurement à l'étude des règles permettant d'adapter les traitements en fonction des transformations effectuées sur les structures de données.

Les principes établis ci-dessus nous permettent de définir l'organisation de la première partie de ce mémoire. Les deux premiers chapitres rappellerons les concepts des modèles E/A et MAG utilisés pour exprimer respectivement le schéma conceptuel et le schéma logique des données. Le troisième chapitre exposera les éléments de la démarche proposée pour la conception d'une BD tandis que le quatrième introduira l'atelier logiciel support de la démarche.

L'exposé des trois premiers chapitres s'inspire directement de (HAINAUT,81), (HAINAUT,86a) et (CAD-MUL,85).

CHAPITRE 1

LE MODELE ENTITE/ASSOCIATION :

MODELE D'EXPRESSION DU SCHEMA CONCEPTUEL DES DONNEES.

Le modèle E/A est un modèle conceptuel de structuration des informations pouvant servir de point de départ dans la démarche de conception d'une BD. Son but est d'exprimer la sémantique des données à l'exclusion des problèmes de représentation physique (codage interne), d'accès à ces données et d'organisation du stockage. Il est sensé représenter l'information intrinsèque nécessaire et suffisante à la connaissance d'un Système d'Informations.

Ce modèle étant supposé connu, nous ne donnerons qu'un bref rappel de ses concepts essentiels. Le lecteur non encore familiarisé peut consulter les références suivantes : (BOD-PIGN, 83), (CHEN,78), (BENCI).

Les concepts élémentaires du modèle sont ceux d'ENTITE, d'ASSOCIATION et d'ATTRIBUT. Outre ces objets de base, le modèle précise aussi la notion de contrainte d'intégrité. Les contraintes d'intégrité formellement prises en charge sont les contraintes de cardinalité dans le schéma des types d'entités et types d'associations, les contraintes de domaine et de format d'attributs, les contraintes de connectivité des rôles joués par les types d'entités au sein des types d'associations, les contraintes de sous-typage des types d'entités au sens strict (héritage des attributs) ou au sens large (héritage des attributs et des types d'associations) et les dépendances fonctionnelles.

CHAPITRE 2

LE MODELE D'ACCES GENERALISE (MAG) :

MODELE D'EXPRESSION D'UN SCHEMA LOGIQUE.

Dans la démarche de conception d'une BD, le Modèle d'Accès Généralisé, variante de la classe des modèles relationnels binaires, sera considéré comme intermédiaire entre le modèle conceptuel et celui du SGD.

Même si les différents SGDs apparaissent très divers, ils mettent en oeuvre les mêmes concepts, et ce qui les distingue fondamentalement est l'ensemble des restrictions qu'ils imposent sur les assemblages possibles de ces concepts. 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 Système de Gestion de Données.

Le MAG est un modèle dont l'ensemble des concepts est le résultat de regroupements de concepts essentiels des SGDs. Il permet ainsi de décrire l'organisation des données de n'importe quel SGD.

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. 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.

Dans le présent chapitre nous donnons un rappel des concepts de base (définition, propriétés, représentation graphique) de ce modèle et une présentation très brève des principales contraintes d'intégrité qu'il permet de définir. Une description plus détaillée, des illustrations de ces concepts, ainsi qu'une introduction aux langages de désignation de données (LDA) et une définition des primitives de manipulation des structures de données du MAG sont données dans (HAINAUT,81) et (HAINAUT,86a).

2.1 OBJETS DE BASE DU MAG.

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

2.1.1 Article (RECORD) et type d'articles (RECORD-TYPE).

Un article est une unité d'information qui peut être créée, modifiée, supprimée et à laquelle il est possible d'accéder. Il constitue l'unité de communication entre la base de données et un programme.

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.

Un type d'articles est désigné par un nom qui l'identifie parmi l'ensemble des types d'articles de la BD.

Il sera représenté graphiquement par un nom inscrit dans un cadre rectangulaire.

2.1.2 Les fichiers.

Un fichier est une collection dynamique d'articles. Chaque article appartient à un et un seul fichier, mais les articles d'un même type peuvent être répartis dans plusieurs fichiers. Un fichier peut collectionner des articles de différents types.

Chaque fichier porte un nom qui l'identifie parmi tous les fichiers d'une même BD.

2.1.3 La base de données.

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.

2.1.4 Valeur d'ITEM et ITEM.

Notion.

Une valeur d'ITEM est un élément d'un type de données (appelé DOMAINE) qui peut être associé à un article. Il n'est possible d'accéder aux valeurs d'ITEMs qu'à partir de l'article auquel elles sont attachées. A tout instant, 0, 1 ou plusieurs valeurs d'ITEMs peuvent être associées à un article déterminé.

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. Chaque ITEM porte un nom qui l'identifie parmi les ITEMs d'un type d'articles.

Propriétés des ITEMs :

- ITEMs élémentaires et ITEMs décomposables.
  La valeur d'un ITEM élémentaire est atomique du point de vue de sa signification; la valeur d'un ITEM décomposable est une suite de valeurs significatives; chacune de ces dernières appartient à un ITEM qui est dit composant de l'ITEM décomposable(a). Un composant peut être lui-même décomposable. Un ITEM sera dit principal s'il a pour père le type d'articles dans lequel il est défini.

- ITEMs simples et ITEMs répétitifs.
  Un ITEM est simple si à chaque article ne peut être associée qu'une valeur de cet ITEM. Un ITEM est répétitif si plusieurs valeurs de cet ITEM peuvent être associées à un article. Dans ce dernier 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 déterminée) ou illimitée (ce cas étant essentiellement d'un intérêt théorique). Un ITEM à répétitivité non fixe est en général accompagné d'un ITEM spécial jouant le rôle de compteur de valeurs pour ce premier(b).

- ITEMs obligatoires et ITEMs facultatifs.
  La propriété pour un ITEM d'être obligatoire ou facultatif est à considérer non pas par rapport au type d'articles le contenant mais par rapport à son père. Un ITEM est facultatif s'il est permis de n'associer aucune valeur de cet ITEM à son père; il est obligatoire sinon.

- 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 notion est parfois locale à un fichier ou à un chemin (voir ci-après les identifiants composés).

(a) Nous parlerons également d'ITEMs fils et père, descendant et ancêtre. Un type d'associations entre un père (ITEM ou type d'articles) et un ITEM fils pourra être appelé "type d'associations CONTAINS".

(b) Nous parlerons également d'un "ITEM compteur" et d'un "ITEM compté" et d'une "liaison compteur" entre ces deux ITEMs.

Représentation graphique.

Graphiquement, l'ITEM sera représenté par son nom et son association à son père par un arc. On distinguera un ITEM simple identifiant (pas de symbole), un ITEM simple non identifiant (triangle avec pointe vers le fils), un ITEM répétitif identifiant (triangle avec pointe vers le père) et un ITEM répétitif non identifiant (diabolo). Une barre sur l'arête signifiera que l'ITEM est obligatoire.

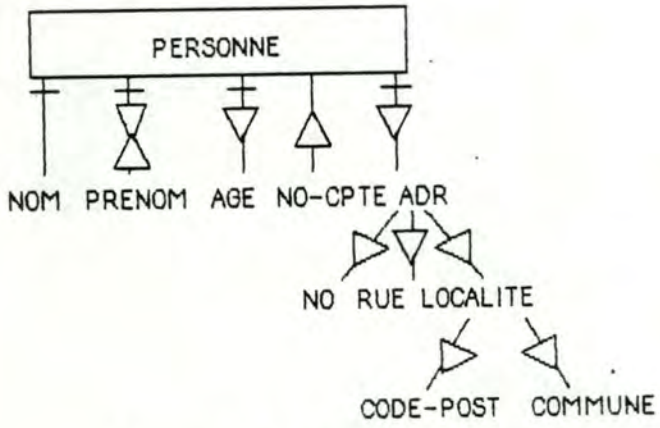


Figure 2.1: Exemple d'ITEMs.

2.1.5 Chemin d'accès et type de chemins d'accès (PATH-TYPE).

Notion.

En considérant uniquement le noyau sémantique du MAG, le chemin d'accès peut être défini comme mécanisme d'association entre articles. Cependant, lorque l'on considère le MAG dans son ensemble, le chemin d'accès peut également jouer le rôle de mécanisme d'accès, c'est-à-dire qu'il permet, à partir d'un article considéré comme origine de ce chemin, d'accéder successivement aux articles (0, 1 ou plusieurs) appelés cibles de ce chemin. On notera que le chemin d'accès définit un accès unidirectionnel.

Tout chemin appartient à un et un seul type de chemins qui en définit les propriétés communes.

A chaque type de chemins est associé un nom qui l'identifie parmi tous les types de chemins ayant même "origine" et même "cible".

Types de chemins particuliers :

- types de chemins inverses : si deux types de chemins sont déclarés inverses, alors pour toute cible C d'un type de chemins ayant O pour origine, il existe un chemin du deuxième type d'origine C dont O est une cible, et inversément.
  Bien que ces deux types de chemins expriment la même sémantique,

leur distinction est nécessaire puisqu'il n'offrent pas le même mécanisme d'accès.

- 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)(a).

- types de chemins récursifs : un même type d'articles est à la fois cible et origine.

Propriétés des types de chemins :

- classe fontionnelle d'un type de chemins.

  Les types de chemins 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. Ces classes fonctionnelles sont les suivantes :

  - "Un-à-Plusieurs" ou 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;

  - "Plusieurs-à-Un" ou 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;

  - "Un-à-Un" ou 1-1 : si un chemin ne peut contenir qu'une seule "cible" et une "cible" ne peut l'être que d'un seul chemin;

  - "Plusieurs-à-Plusieurs" ou N-M : si un chemin peut contenir un nombre quelconque de "cibles", et si une "cible" peut l'être d'un nombre quelconque de chemins;

- contrainte d'existence.

  Elle consiste à imposer à tout article d'un membre d'un type de chemins d'être associé à au moins un article de l'autre membre. On dira aussi que le type de chemins est obligatoire pour le membre sur lequel est posée cette contrainte.

  Remarque : par la suite, pour faciliter l'expression des règles de transformation, la présence ou non de contraintes d'existence dans un type de chemins et sa classe fonctionnelle seront assimilées respectivement aux connectivités minimale et maximale du modèle E/A.

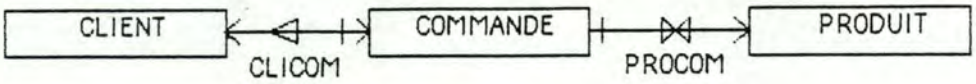
Représentation graphique.

Graphiquement, un type de chemins sera représenté par un arc étiqueté de son nom. Si un accès a été défini dans un type de chemins, cet arc sera

---

(a) Nous parlerons dans ce cas de types de chemins à extrémités multitypes ou de types de chemins multirôles.

orienté en partant de la représentation de l'origine et en aboutissant à celle de la cible. Ce principe sera généralisé aux types de chemins multitypes. Lorsque deux types de chemins sont inverses, on pourra les représenter par un arc bidirectionnel. La classe fonctionnelle sera représentée de la même façon que les propriétés d'identifiant et de répétitivité des ITEMs (triangle, diabolo ou absence de symbole).



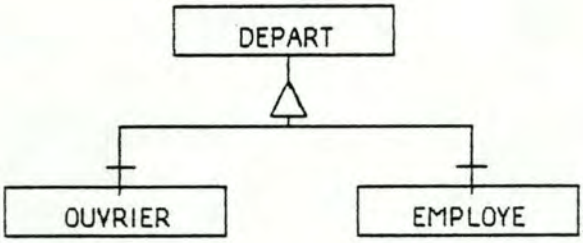


Figure 2.2: Exemple de types de chemins.

2.1.6 Les identifiants composés.

La notion d'identifiant introduite en 2.1.4. comme propriété d'un ITEM peut également être considérée comme propriété d'un type de chemins en ce sens qu'un type de chemins peut être identifiant d'une de ses extrémités (éventuellement des deux). La notion d'identifiant peut également être étendue à une combinaison d'ITEMs et/ou de types de chemins. Un identifiant composé sera représenté graphiquement par un parallélisme partiel des arcs reliant le type d'articles identifié à chaque composant de l'identifiant.

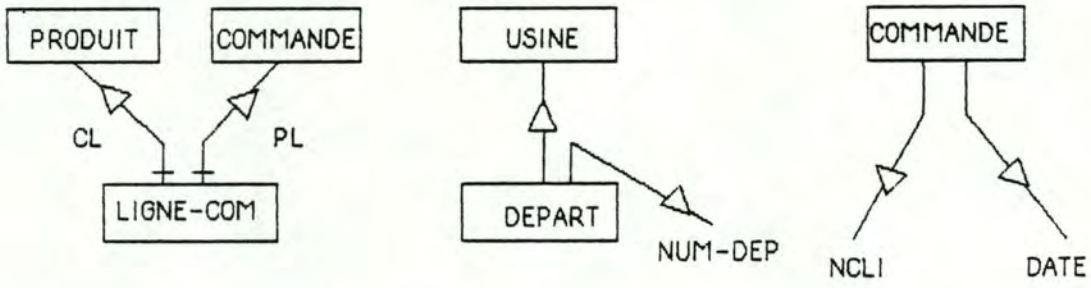


Figure 2.3: Exemple d'identifiants composés.

2.1.7 Les clés d'accès.

Notion.

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 chemin ou à un fichier(a).

Représentation graphique.

Graphiquement, une clé d'accès sera représentée par un arc orienté de la représentation de l'ITEM (ou groupe d'ITEMs) clé vers celle du type d'articles.

Figure 2.4: Exemple de clés d'accès.

2.1.8 Ordre d'une séquence d'articles.

Les principaux agrégats d'articles qui peuvent constituer une séquence pour le concepteur sont essentiellement la base de données, le fichier, le chemin d'accès et l'ensemble des articles associés à une valeur déterminée d'une clé.

Pour chacune de ces quatre constructions, l'accès séquentiel à ces articles s'effectue, en l'absence de spécification contraire, dans un ou plusieurs ordres prédéterminés.

Un ordre peut être local (s'il est défini sur une séquence d'articles d'un seul type) ou global (si la séquence sur laquelle il est défini contient des articles de différents types).

En se restreignant au premier type, le MAG permet de définir un ordre d'un des quatre types suivants :

- pas d'ordre significatif (random);
- ordre lié à l'instant de l'insertion dans la séquence : chronoloique (FIFO) ou anti-chronologique (LIFO);

(a) La notion de composant de clé d'accès ou d'identifiant sera généralisée aux types de chemins.

- ordre programmé : le prorammeur 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 clé de tri.

2.1.9 L'article système.

Toute base de données contient un et un seul article d'un type particulier, appelé article système. Cet article constitue un point d'entrée privilégié dans la base de données. Il peut être origine de chemins d'accès, mais ne peut pas en être cible.

2.2 CONTRAINTES D'INTEGRITE.

La structuration des données au moyen des concepts du MAG introduit en 2.1 conduit à des représentations trop générales des structures conceptuelles. Ces dernières sont astreintes à respecter des règles appelées contraintes d'intégrité. Il est donc nécessaire d'adjoindre à une description en termes des objets de base, un ensemble de règles qui traduisent les contraintes d'intègrité de la description conceptuelle.

Les contraintes les plus souvent utilisées (classes fonctionnelles, contraintes d'existence et identifiants) ont déjà été spécifiées en 2.1. Cependant, il sera souvent nécessaire d'exprimer des contraintes d'intégrité d'autres types tels que : les contraintes de cardinalité, de référence, d'expression de redondance, de dérivation ou d'équivalence entre schémas, ...

CHAPITRE 3

DEMARCHE DE CONCEPTION D'UNE BASE DE DONNEES.

Ce chapitre précise la démarche méthodologique de conception de bases de données dont la structure a été évoquée dans le traitement du cas (cfr. partie introductive) et dans l'introduction de cette première partie.

Cet exposé permettra de mettre en évidence les propriétés avantageuses de la démarche :

- elle prend en charge tous les critères de décision correspondant aux objectifs du système final (correction, performance, conformité aux contraintes d'un SGD, indépendance des programmes vis-à-vis des données, adaptabilité par rapport aux besoins futurs);
- elle permet de localiser les dépendances vis-à-vis des outils opérationnels sur lesquels le système sera implanté, et en particulier le SGD;
- bien que basée sur des principes intuitifs, cette démarche présente une assez grande rigueur.

3.1 ANALYSE CONCEPTUELLE.

L'analyse conceptuelle conduit à l'élaboration d'une description complète du Système d'Informations qui soit indépendante de la notion même d'outil informatique. Cette description est constituée d'un schéma conceptuel des données, d'un schéma conceptuel des traitements, ainsi que du relevé des quantifications.

3.2 LA CONCEPTION LOGIQUE.

Cette phase consiste à traduire la solution conceptuelle en une solution qui soit exécutable par une machine abstraite, c'est-à-dire strictement indépendante des machines réelles.

La conception logique s'avère nécessaire pour localiser et réduire les dépendances vis-à-vis d'un SGD et de langages de programmation particuliers puisqu'elle permet de retarder les décisions concernant ces composants.

Les modèles utilisés lors de cette phase sont le Modèle d'Accès Généralisé (MAG) pour la représentation des structures de données et des structures d'accès à ces données et le langage de description d'algorithmes (LDA). Ce langage est compatible avec le MAG (LDA/MAG) en ce sens que les algorithmes percevront les données au travers de structures MAG. IL permet d'exprimer les algorithmes de manière simple, quel que soit leur niveau de complexité, en particulier en ce qui concerne la désignation des données et leur manipulation. Ce langage offre non seulement des primitives de désignation et de manipulation de données comparables à celles offertes par les langages de programmation traditionnels, mais également des expressions de haut niveau comme celles qui sont offertes par certains SGDs relationnels.

La solution logique sera constituée :

- d'algorithmes qui satisfont à leur spécification et qui sont efficaces en ce qui concerne le nombre d'accès à la BD;

- des structures de données exprimées dans le MAG ayant la même sémantique que celles de la solution conceptuelle. A ces structures s'ajoute une spécification des accès jugés nécessaires pour l'exécution efficace des algorithmes.

Les étapes menant à cette solution peuvent être résumées comme suit :

- production d'un schéma des accès possibles (SAP).
  Ce schéma exprime dans le MAG une structure des données qui est une représentation aussi directe que possible du schéma conceptuel. Il ne contient donc aucune spécification d'accès. Son objectif est d'offrir une expression des données dans un formalisme adapté aux processus de conception logique.

- construction de l'architecture et des algorithmes (algorithmes prédicatifs).
  A partir de la spécification conceptuelle des traitements, est définie une architecture abstraite adéquate de modules. Chaque module y reçoit une spécification externe et se voit affecté un algorithme qui satisfait à cette spécification. Si cet algorithme accède à la BD, 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.

- développement d'algorithmes efficaces.
  On recherchera pour chaque algorithme prédicatif une stratégie d'accès efficace à la BD. Le seul critère de performance envisageable à ce stade est le nombre d'opérations logiques. Ce critère est nécessaire pour choisir entre les différents algorithmes effectifs équivalents qui peuvent être dérivés d'un algorithme prédicatif.

- dérivation du schéma des accès nécessaires (SAN).
  A partir du schéma SAP et de l'algorithme effectif retenu, on relève pour chaque traitement les structures de données et d'accès strictement nécessaires à une exécution efficace. On obtient ainsi un sous-schéma "effectif" relatif à chaque traitement. L'intégration de tous ces "sous-schémas" donne le schéma des accès nécessaires.

3.3 LA CONCEPTION PHYSIQUE.

Dans la dernière phase de la démarche de conception d'une BD la solution logique exécutable par une machine abstraite va être transformée et optimisée afin de s'adapter à un processeur réel, c'est-à-dire à un langage de programmation et à un SGD réel.

La solution physique sera constituée des schémas logique, interne et externes de la BD et des programmes d'application travaillant sur ces schémas.

La phase de conception physique se divise en deux sous-couches : l'une, dite abstraite, spécifie une solution conforme à la machine réelle mais encore exprimée en termes du MAG et LDA, et l'autre, dite concrète, conduit à la construction de la solution exécutable à partir de cette spécification.

CONCLUSION.

La démarche peut être définie comme un ensemble de processus de transformation de spécifications et de descriptions. Sa modularité permet de l'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, ...).

L'application de la démarche à des problèmes en vraie grandeur fait ressortir la nécessité de s'appuyer sur des outils qui facilitent le

passage à travers les différentes couches de la démarche. L'objectif de l'atelier logiciel orienté bases de données est de fournir à terme cet environnement.

Comme nous l'avons déjà signalé, nous ne prendrons en considération au niveau de la démarche et du support que les transformations de structures de données.

CHAPITRE 4

L'ATELIER LOGICIEL

SUPPORT DE LA DEMARCHE DE CONCEPTION D'UNE BASE DE DONNEES.

Un atelier logiciel se donnant pour objectif de fournir des outils de conception et d'aide à la conception de BDs sur machines réelles au départ de spécifications conceptuelles est en cours de développement à l'Institut d'Informatique de Namur. Cet atelier couvre donc les niveaux logique et physique de la démarche.

Pour faciliter la mise en oeuvre de ces outils un méta-schéma (appelé schéma de l'atelier), a été défini récemment qui, comme son nom l'indique, permet une représentation unique des schémas de données quel que soit le modèle utilisé (E/A, MAG, ...). Ce méta-schéma s'est avéré indispensable pour la conception de BDs en vraie grandeur. Il permet entre autres de passer plus rapidement de la solution conceptuelle à la solution logique et de définir les liaisons entre les différentes versions de schémas.

Dans un premier temps, les modèles conceptuels de départ seront supposés être les modèles offerts par IDA (tel que le modèle E/A décrit plus haut). IDA (Interactive Design Approach) est un atelier logiciel réalisé à l'Institut d'Informatique de Namur en copération avec le projet ISDOS, qui intègre des outils d'aide à la spécification de Systèmes d'Informations. Un bref aperçu des outils fournis par l'atelier logiciel orienté BD ainsi que de ses connections avec IDA est donné ci-dessous. Nous préciserons également la contribution à l'atelier du présent mémoire.

4.1 ARCHITECTURE GENERALE DE L'ATELIER LOGICIEL DE CONCEPTION DE BASES DE DONNEES.

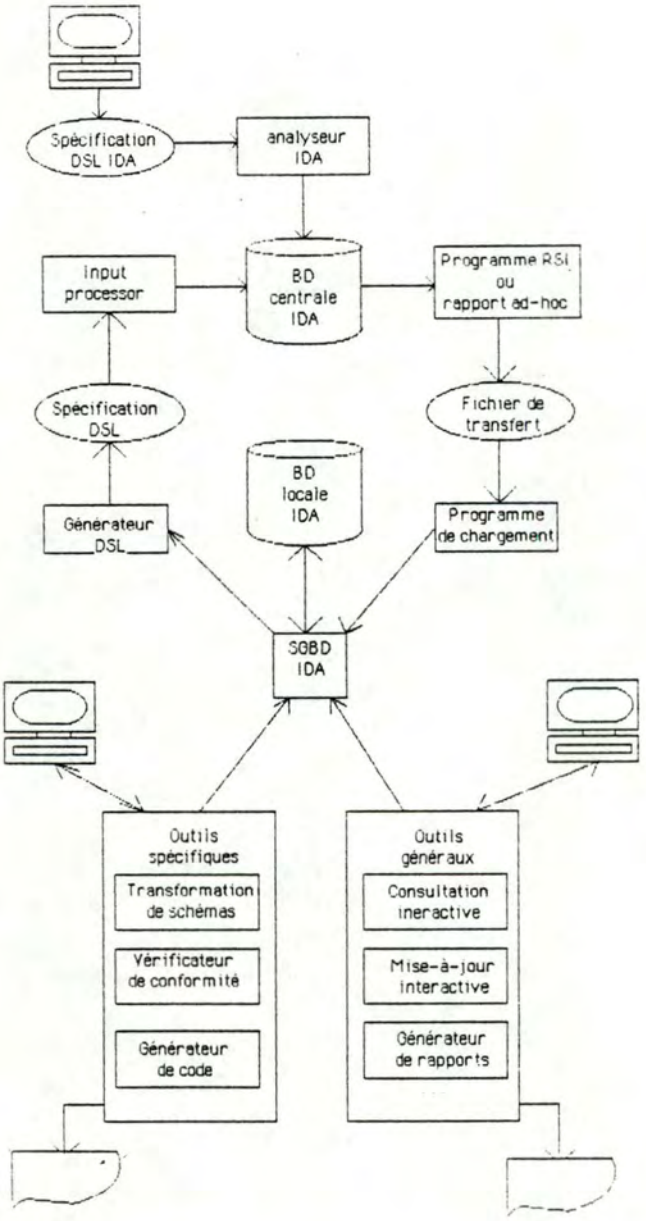


Figure 4.1: Architecture générale de l'atelier logiciel

Des instructions DSL IDA sont lues par l'analyseur DSA qui, après avoir procédé aux contrôles syntaxiques et sémantiques, les enregistre dans la base de données des spécifications conceptuelles (= BD centrale d'IDA). Ces spécifications pourront ensuite être extraites de la base centrale par des programmes de chargement pour être enregistrées dans une BD locale à l'atelier logiciel orienté Bases de Données. Ces spécifications pourront ainsi profiter des traitements effectués au moyen des outils qui seront offerts par cet atelier.

Cet atelier désire intègrer

- d'une part, des outils généraux tels que des outils de consultation et de mise-à-jour interactives, de génération de rapports documentaires ou d'analyse présentant sous des formes variées certains aspects du système spécifié, ...

- et d'autre part, des outils plus spécifiques tels qu'un vérificateur de conformité d'un schéma à un jeu de règles donné, un transformateur de schémas (conversion d'un schéma conceptuel en un texte SGD), générateur de code, ...

4.2 CONTRIBUTION DU MEMOIRE A L'ATELIER.

Dans le cadre de ce mémoire, nous analyserons les outils de transformations de schémas de données et de vérification de conformité aux SGDs COBOL, CODASYL et SQL qui assisteront le passage de la solution conceptuelle à la solution physique.

L'option a été choisie d'orienter les outils non pas vers des outils de conception mais vers des outils d'aide à la conception.

Pour adapter les outils à la propriété de modularité de la démarche, il est préférable de construire plusieurs transformateurs au lieu d'en concevoir un qui parcoure en une seule fois toute la chaine de conception.

Les outils analysés devront notamment :

- laisser au concepteur un maximum de liberté dans ses prises de décision;
- offrir une assistance maximum dans l'emploi des transformations;
- assurer l'équivalence entre la solution conceptuelle et la solution physique;
- être facilement extensible à :
  - l'automatisation (partielle ou complète) des transformations;
  - la prise en compte d'autres modèles conceptuels et SGDs;
- être facilement adaptables à la prise en charge ultérieure des traitements.

PARTIE II

TRANSFORMAITONS DE SCHEMAS

INTRODUCTION.

Selon la démarche proposée pour la conception d'une base de données, les transformations se subdivisent en trois grandes catégories :

- les transformations E/A --> MAG
- les transformations MAG --> MAG
- les transformations MAG --> DDL SGD.

En ce qui concerne la mise en oeuvre des transformations, il a été opté de laisser au concepteur toutes les prises de décision. Pour les transformations terminales (E/A --> MAG et MAG --> DDL SGD), le concepteur devra peu intervenir. Les transformations MAG --> MAG par contre requièrent un degré d'intervention assez élevé.

Comme nous l'avons déjà signalé, les transformations doivent assurer l'équivalence sémantique et éventuellement celle au niveau des accès. Cette équivalence ne sera pas prouvée formellement : un tel travail constitue une recherche d'une ampleur considérable et n'est pas l'objectif primordial de ce mémoire. Nous nous limiterons à mettre en évidence les transformations qui nous semblent intuitivement ne pas assurer cette équivalence.

Dans un premier temps, les schémas conceptuels de départ seront supposés cohérents et non redondants et nous ne retiendrons que des transformations irredondantes (qui n'introduisent pas de redondances supplémentaires). Précisons également que les contraintes d'intégrité autres que celles d'identifiant, d'existence, de classes fonctionnelles, et de cardinalité pourront ne pas être prises en compte dans le processus de transformation.

Pour chacune des trois catégories de transformations on étudiera dans cette deuxième partie, la définition des transformations, l'objectif qui leur est assigné et les règles qui les caractérisent.

Les transformations MAG --> MAG seront complètées en annexe par l'expression de leur algorithme. Le premier chapitre de cette partie (chapitre 5) est consacré à l'étude des transformations E/A --> MAG pour lesquelles on se limitera à l'exposé de l'élimination des structures du modèle E/A non compatibles avec le noyau sémantique du MAG.

Les transformations MAG --> MAG constituant le coeur de la démarche de conception d'une base de données occuperont la place majeure dans cette partie. L'étude proprement dite sera précédée de la définition de la conformité d'un schéma MAG à un SGD (chapitre 6). Le chapitre 7 est consacré à l'analyse détaillée des transformations MAG --> MAG.

On analysera successivement au cours de ce chapitre les transformations abstraites et les transformations concrètes : les transformations abstraites constituant des fondements théoriques par rapport auxquelles sont définies les transformations concrètes. Le huitième chapitre abordera brièvement la traduction dans le DDL propre à un SGD d'un schéma MAG qui lui est conforme.

CHAPITRE 5

TRANSFORMATIONS DE SCHEMAS E/A EN SCHEMAS MAG.

L'objet du présent chapitre concerne la traduction des structures de données résultant de l'analyse conceptuelle, supposées cohérentes et non redondantes, dans un formalisme mieux adapté au processus de conception logique. En particulier, il s'agit de décrire les règles de correspondance entre les concepts E/A et ceux du MAG. L'exposé de ces règles sera précédé de la définition de la forme canonique du modèle E/A qui sera retenue dans le processus de conception de BDs. Il est clair que la prise en charge d'autres modèles conceptuels que le modèle E/A réclame simplement un autre jeu de règles.

Ces règles permettront de produire, le plus rapidement possible, un schéma MAG qui soit sémantiquement équivalent et le plus proche possible du schéma conceptuel. En particulier, le schéma obtenu sera exprimé uniquement sur le noyau sémantique du MAG.

Lors du processus de transformation E/A --> MAG nous permettrons d'anticiper sur les futurs sens d'accès qui seront définis dans les types de chemins et éventuellement sur des transformations qui devront avoir lieu dans la phase de conception logique. Ces règles prevoyerons dès lors que le concepteur puisse choisir respectivement les noms des types de chemins et la solution de transformation (lorsque plusieurs techniques sont proposables).

5.1 FORME CANONIQUE RETENUE DU MODELE E/A.

Ci-dessous nous précisons les élargissements et restrictions de la forme canonique du modèle E/A qui sera retenue dans le cadre de ce mémoire par rapport à celle offerte par IDA. On trouvera un exposé détaillé de cette dernière forme dans (BER-GAT, 86).

Les élargissements par rapport aux éléments de cohérence exigés par IDA sont essentiellement les suivants :

- la règle d'unicité des noms d'objets imposant que tout objet aie un nom qui l'identifie parmi tous les objets du schéma (qu'ils soient ou

non de même type) sera assouplie en fonction des règles établies par le MAG. En particulier, deux noms de rôles pourront avoir le même nom s'ils sont assumés par des types d'entités distincts et/ou dans des types d'associations distincts.

- un type d'entités ou d'associations pourra ne pas avoir d'identifiant;
- l'identifiant d'un type d'associations pourra être formé d'un sous-ensemble, éventuellement vide, des rôles des types d'entités sur lesquels il est défini et d'attributs de ce type d'associations.

Les restrictions ont trait à la non prise en compte dans le processus de transformation de :

- de la notion de sous-typage entre types d'entités;
- les dépendances fonctionnelles autres que les identifiants.

Remarquons que, si l'on suppose que tout type d'associations n'est plus décomposable, toutes les contraintes d'intégrité prises en charge par IDA seront considérées lors de l'élaboration de la BD réelle. La connectivité maximale d'un rôle pourra représenter aussi bien une valeur indéterminée qu'une valeur déterminée. Dans ce dernier cas, et si cette valeur n'est pas 1, la connectivité maximale deviendra indéterminée et sera accompagnée d'une contrainte d'intégrité.

5.2 REGLES DE CONVERSION.

Ayant choisi le modèle E/A comme point de départ de la démarche, le jeu de règles de conversion sera simple puisque :

- d'une part, le modèle E/A a peu de concepts et
- d'autre part, le noyau sémantique du MAG peut être considéré comme un modèle conceptuel de type E/A restreint aux types d'associations binaires (types d'associations portant sur deux types d'entités et deux seulement).

Nous limiterons dès lors l'exposé des règles de conversion à l'élimination des structures du modèle E/A non reprises dans le MAG.

1. TYPES D'ASSOCIATIONS N-AIRES (N > 2).

Le type d'associations R (E1:r1, E2:r2, ... , En:rn) où n > 2 sera transformé en :

- un nouveau type d'entités : soit E' et
- n nouveaux types d'associations binaires Ri'(E',Ei) dans lesquels la connectivité de E' sera 1-N et celle de Ei (en supposant que R est non récursif) sera celle que ce type d'entités avait dans R. Si R était récursif, chacun de ses membres sera identifié par le nom du type d'entités membre et celui du rôle joué par ce type d'entités dans R.

Les valeurs par défaut des noms des nouveaux objets sont :

- pour le nouveau type d'entités : le nom du type d'associations R et
- pour les nouveaux types d'associations Ri' : les noms des rôles ri correspondants.

Des rôles pourront être définis dans les nouveaux types d'associations Ri'.

2. TYPES D'ASSOCIATIONS AVEC ATTRIBUTS.

Ces types d'associations seront transformés comme dans le point précédent, même s'ils sont binaires, et leurs attributs seront définis pour les nouveaux types d'entités correspondants.

Une autre possibilité de transformation sera proposée si le type d'associations est binaire et qu'au moins l'un de ses rôles a une connectivité i-1. Cette transformation consiste à transférer les attributs du type d'associations vers le type d'entités jouant ce rôle.

Remarque : cette transformation n'assure l'équivalence sémantique que si soit tous les attributs du type d'associations sont obligatoires, soit (non exclusif) i vaut 1.

3. TYPES D'ASSOCIATIONS BINAIRES RECURSIFS.

Pour anticiper sur l'élimination de la récursivité qui pourra être demandée dans la phase de conception logique, il peut être intéressant de proposer au concepteur de transformer les types d'associations récursifs en types d'entités comme dans le premier point.

4. IDENTIFIANTS DES TYPES D'ENTITES.

Comme nous l'avons déjà signalé lors de l'exposé de la forme canonique du modèle E/A retenue (cfr chapitre 1), un identifiant d'un type d'entités peut être composé uniquement d'attributs internes et/ou de rôles assumés par ce type d'entités au sein de types d'associations. Notons que dans les règles données ci-dessous on suppose que tout identifiant d'un type d'entités ou d'associations est minimal.

Uniquement les composants rôles joués dans des types d'associations devenus types d'entités devront être transformés.

Pour aider la compréhension du principe de conversion, considérons l'exemple suivant où un type d'entités A est identifié par un attribut interne IA via son rôle dans le type d'associations ternaire ABC transformé en un type d'entités.

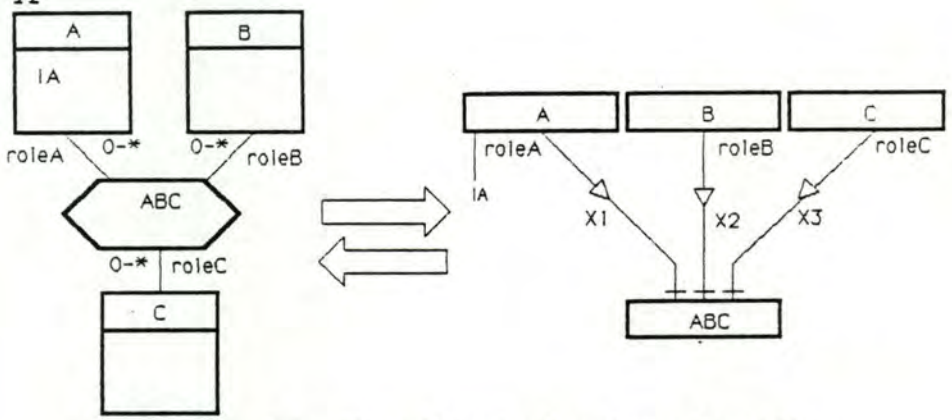


Figure 5.1: Identifiant d'un type d'entités.

Le nouveau type d'entités ABC ayant une connectivité maximale 1 dans tous les nouveaux types d'associations, le composant role-A de l'identifiant devra être remplacé par les deux rôles composés suivants :

- rôle de A dans X1 composé au rôle de ABC dans X2 et
- rôle de A dans X1 composé au rôle de ABC dans X3.

La notion de rôle composé est inexistante dans le MAG. Il pourrait cependant être possible d'éviter de passer par la notion de rôle composé en procédant par des copies d'identifiants. Nous n'entrons pas dans les détails ici et nous supposerons dans un premier temps qu'aucun identifiant d'un type d'entités n'est composé de l'un de ses rôles joués dans un type d'associations à transformer en un type d'entités.

5. IDENTIFIANTS DE TYPES D'ASSOCIATIONS.

Identifiants implicites.

En toute généralité, tout type d'associations est implicitement identifié par les types d'entités sur lesquels il porte. Par conséquent, pour tout type d'associations traduit en un type d'entités et une série de type d'associations, dans le cas où aucun identifiant explicite n'est défini pour ce type d'associations, il faudra lui définir un identifiant formé de l'ensemble de ses rôles dans les nouveaux types d'associations. Tout type d'associations étant supposé minimal, nous sommes assurés que cet identifiant sera strict.

Identifiants explicites.

Un identifiant explicite d'un type d'associations sera un identifiant composé d'au moins un attribut interne au type d'associations identifié.

CHAPITRE 6

NOTION DE CONFORMITE.

Comme nous l'avons déjà signalé, le MAG est un modèle de référence qui permet de décrire des structures de données et d'accès indépendamment d'un SGD particulier mais tout en étant proche de ses concepts intrinsèques. En toute généralité, un SGD sera plus restrictif que le MAG en ce qui concerne les constructions de description admises.

Un schéma MAG sera dit conforme à un SGD s'il ne contient aucune construction incompatible avec la spécification de ce SGD.

Le but de la phase de construction de schémas conformes étant d'éliminer les structures invalides(a), il est préférable de parler en termes de contraintes posées plutôt qu'en termes de règles de construction de schémas.

Dans un soucis d'indépendance à un SGD particulier, ces contraintes seront exprimées globalement dans les termes du MAG plutôt que particulièrement à chaque SGD(b).

A ce stade de production de schémas conformes, nous ne retiendrons que les contraintes logiques, les contraintes physiques intervenant lors de la phase de traduction du schéma conforme dans le langage de description de données (DDL) du SGD. Il est cependant parfois difficile de séparer les concepts logiques des concepts physiques (par exemple, la notion d'ordre des articles dans un référentiel donné, les formats et longueurs des items,...). Afin de rendre la phase de traduction dans le DDL d'un SGD la plus directe que possible, de tels concepts seront considérés comme concepts logiques.

Dans un soucis de repérer un maximum de restrictions qui peuvent être posées sur le MAG par un SGD, nous nous sommes placées dans le cadre de trois SGDs généralement considérés comme concurrents : un simple Système de Gestion de Fichiers (COBOL), un SGBD CODASYL (CODASYL 71) et un SGBD relationnel (SQL). Les règles de construction de schémas conformes COBOL, CODASYL et SQL peuvent être consultées dans (CLARINVAL,81), (HAINAUT,84), (HAINAUT,86a) et (OLLE,78).

Dans ce chapitre nous présentons quelques contraintes qui peuvent être posées par un SGD sur les structures de données et d'accès du MAG. Une liste complète de ces contraintes sera donnée dans le chapitre 10. Les

---

(a) Nous utiliserons le terme "invalide" selon l'acception suivante : "qui ne satisfait pas aux conditions légales".

(b) Dans un soucis de généralisation de la notion de conformité à un jeu de contraintes quelconque, nous pourrons par la suite parler de conformité plutôt que de conformité à un SGD.

listes des contraintes posées en particulier par chaque SGD retenu peuvent être trouvées en annexe.

Exemples de restrictions d'un SGD par rapport au MAG.

Les restrictions d'un SGD par rapport au MAG portent essentiellement sur les propriétés des ITEMs et des types de chemins, sur les concepts d'identifiant, de clé d'accès et d'ordre, sur le nombre de types d'articles dans un fichier et le nombre de fichiers par type d'articles, sur le format et l'unicité des noms d'objets.

Nous n'entrerons pas ici dans les détails des contraintes de formation des noms d'objets, des contraintes sur les notions d'ordre et de fichier, de même que des contraintes sur les notions d'identifiant et de clé d'accès trop spécifiques à un SGD particulier.

En ce qui concerne les ITEMs, la plupart des SGDs (en l'occurrence COBOL et SQL) imposent que tout type d'articles contienne au moins un ITEM. En toute généralité, un SGD pose une contrainte sur le nombre maximum de niveaux qu'il est permis d'atteindre dans le graphe de décomposition des ITEMs. Ce nombre vaut 49 pour COBOL, 99 pour CODASYL. SQL n'admettant pas les ITEMs décomposables, ce nombre vaudra bien évidemment 1 pour ce SGD. Certains SGDs n'admettent de plus pas qu'un ITEM soit répétitif. SQL est un exemple d'un tel SGD. COBOL et CODASYL par contre permettent aux ITEMs d'être répétitifs (de répétitivité fixe ou variable), mais sous certaines conditions et en l'occurrence, un ITEM ne pourra être de répétitivité variable que s'il est accompagné d'un ITEM compteur. COBOL n'admet de plus pas qu'un ITEM de répétitivité variable ne soit pas le dernier dans le graphe de décomposition. La plupart des SGDs et notamment COBOL et SQL ne permettent pas aux ITEMs d'être facultatifs. SQL offre cependant une valeur spéciale (NIL) permettant de représenter l'absence de valeur.

La notion de type de chemins n'est pas prise en compte par tous les SGDs. COBOL et SQL ignore d'ailleurs cette notion. Les SGDs qui la prennent en considération sont en général assez restrictifs en ce qui concerne les valeurs qui peuvent être prises par leurs propriétés de classe fonctionnelle et de contrainte d'existence. CODASYL par exemple impose que tout type de chemins soit de classe fonctionnelle 1-N. De plus en CODASYL, aucune contrainte d'existence ne peut être posée sur l'OWNER (type d'articles du côté 1 de la classe fonctionnelle). Des contraintes pourront également être posées sur le nombre de types d'articles sur lesquels les types de chemins peuvent porter et éventuellement sur le nombre de types d'articles qui peuvent composer une même extrémité. En CODASYL par exemple, le type de chemins peut être multitype mais pas du côté OWNER. Un SGD pourra également interdire qu'un même type d'articles fasse partie des deux extrémités du type de chemins. Tel est le cas par exemple pour CODASYL. En ce qui concerne le sens d'accès dans les types de chemins, CODASYL 78 considère que tous les types de chemins sont bidirectionnels. La version CODASYL 71 offre automatiquement l'accès dans les deux sens mais il est possible de restreindre l'accès de l'OWNER vers les MEMBERs au moyen de DYNAMIC SETs. Certains SGDs pourront aussi n'offrir que des types de chemins à sens unique et ne pas gérer les contraintes d'inverse entre types

de chemins.

En ce qui concerne les contraintes sur les notions d'identifiant et de clé d'accès, celles-ci sont souvent très spécifiques aux SGDs. Certaines de ces contraintes sont des limitations sur les combinaisons possibles de ces concepts, d'autres portent plus particulièrement sur le nombre et le type des composants, d'autres encore portent sur le nombre d'identifiants/clés d'accès, éventuellement dont les composants sont d'un type déterminé, qui peuvent être définis pour un même type d'articles. Les identifiants/clés d'accès peuvent de plus être locales à un sous-ensemble de fichiers. Nous ne donnerons pas ici toutes les contraintes qui sont posées en particulier par les SGDs COBOL, CODASYL et SQL, mais nous nous limiterons à en donner quelques exemples.

Les notions d'identifiant, de clé d'accès et de clé d'ordre sont souvent fortement liées pour la plupart des SGDs. Tel est le cas plus particulièrement pour COBOL et SQL. Pour ces SGDs, il n'est pas possible de définir des identifiants qui ne soient pas clés d'accès. De plus, parmi les clés d'accès COBOL, au moins une doit être identifiante. En CODASYL, uniquement les identifiants dans un type de chemins peuvent ne pas être clés d'accès. Notons que dans CODASYL 78, cette possibilité de clé d'accès non identifiante disparait complètement. En ce qui concerne les contraintes portant sur le nombre de composants que peut avoir un identifiant/clé d'accès, COBOL impose que ce dernier soit composé d'un unique composant ITEM. En CODASYL, tout identifiant ne pourra pas avoir plus d'un composant type de chemins mais pourra avoir un nombre quelconque de composants ITEMs. Ces derniers pourront, contrairement à ce que permet COBOL, être situés n'importe où dans le graphe de décomposition. COBOL n'admet en effet pas que deux identifiants/clés d'accès définis pour un même type d'articles commencent à la position. Pour ces deux SGDs aucune clé d'accès ne peut être constituée d'ITEMs répétitifs. En ce qui concerne le nombre d'identifiants/clés d'accès par type d'articles, COBOL et SQL ne poseent apparamment aucune restriction. CODASYL par contre n'admet pas plus d'un identifiant par type d'articles qui ne soit composé que d'ITEMs.

Conclusion.

Le schéma MAG directement dérivé du schéma conceptuel devra subir des transformations de manière à éliminer toute construction non conforme.

Pour aider le concepteur à repérer dans son schéma les constructions à éliminer, il serait opportun d'automatiser ce processus dans un outil logiciel qui sera appelé vérificateur de conformité.

La définition et la mise en oeuvre du processus de transformation seront étudiées dans les chapitres suivants. En particulier, le chapitre 7 est consacré à l'analyse des règles de transformation tandis que le chapitre 12 propose un sens de parcours efficace pour le vérificateur de conformité.

CHAPITRE 7

TRANSFORMATIONS DE SCHEMAS MAG EN SCHEMAS MAG.

Comme nous l'avons déjà indiqué lors de l'analyse de la démarche de conception de bases de données, le schéma MAG, directement obtenu du schéma conceptuel, doit subir des transformations puisqu'il ne tient compte ni des contraintes spécifiques à un SGD, ni des critères de performance liées à l'organisation de la BD.

L'objectif de ce chapitre concerne la définition de transformations qui prennent en compte ces contraintes et critères.

Le schéma MAG directement obtenu par transformation du schéma conceptuel ne prend pas en considération les notions d'accès exposées au chapitre 3 décrivant le Modèle d'Accès Généralisé. Précisons cependant que, notre processeur de transformation étant indépendant de la construction des algorithmes, la spécification des accès nécessaires ne pourra se faire qu'interactivement via de simples demandes de mise à jour de la BD.

Plutôt que de se placer directement au niveau de l'analyste pour imaginer les transformations qui peuvent lui être utiles, il s'est avéré indispensable de faire précéder l'approche pratique d'une approche théorique. Cette dernière mettra en évidence un ensemble de transformations de base (appelées transformations abstraites ou théoriques) pour lesquelles la préservation de l'équivalence peut être démontrée formellement. Pour une raison essentiellement pédagogique, nous nous limiterons à donner une idée de base de chaque transformation abstraite. Cet exposé fera l'objet de la première section de ce chapitre. Une liste étendue de transformations abstraites a été étudiée. Seul un sous-ensemble reprenant les plus utiles (appelées transformations concrètes) seront analysées plus en détail dans la deuxième section de ce chapitre. D'autres, considérées comme moins utiles mais cependant intéressantes pour augmenter les possibilités de choix du concepteur, seront simplement mentionnées. Les transformations concrètes seront analysées sous l'angle de la conformité puis sous celui de l'affinage de schémas de données.

Dans le cadre de ce mémoire, nous ne considérons que des transformations assurant l'équivalence sémantique (et donc réversibles) et irredondantes (c'est-à-dire qui n'amènent pas de redondances supplèmentaires). Ces transformations seront également étudiées du point de vue de l'équivalence des accès. Nous mettrons en évidence les quelques situations où celle-ci n'est pas assurée.

7.1 LES TRANSFORMATIONS ABSTRAITES.

Dans cette section nous proposons un ensemble de transformations de base permettant de passer d'un schéma MAG à un autre équivalent au premier.

Ces transformations se classent en différentes familles exprimées dans des termes génériques en vue d'un meilleur regroupement. En particulier, on parlera de type d'objets (resp. type d'associations) lorsque le principe de base de la transformation ne se soucie pas de la distinction entre ITEM et type d'articles (respectivement CONTAINS et type de chemins). Graphiquement un type d'objets sera représenté par un cercle et un type d'associations par un arc.

Les deux principales familles sont la création/suppression d'un type d'articles intermédiaire dans un type d'associations et la rotation/rotation inverse d'un ou de plusieurs types d'associations.

On ne donnera ci-dessous qu'un bref aperçu de chaque famille ainsi que des problèmes rencontrés qui ont amené à des limitations sur les constructions MAG finalement prises en compte lors du processus de transformation. Le lecteur intéressé touvera dans les documents de travail une analyse plus détaillée de ces familles, sauf pour celles qui seront présentées dans les points 7.1.1.4 et 7.1.1.5.

Les transformations devant être applicables aussi bien pour des schémas avec accès que pour des schémas sans accès, l'équivalence sera d'abord étudiée du point de vue de la sémantique, puis du point de vue des accès.

Notons également qu'une transformation applicable pour un type de chemins de classe fonctionnelle N-M (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).

7.1.1 ANALYSE DES FAMILLES DE TRANSFORMATIONS.

Rappelons que les conventions de représentation suivantes ont été prises : un cercle représentera un type d'objets (ITEM ou type d'articles) et un arc représentera un type d'associations (CONTAINS ou type de chemins). La propriété de classe fonctionnelle d'un type de chemins sera étendue au type d'associations.

Dans cet exposé nous donnons pour chaque transformation abstraite : sa définition exprimée essentiellement en termes d'une représentation graphique de ses effets généraux, ses objectifs afin de déterminer son niveau d'utilité pratique, ses préconditions, ainsi qu'une idée de base grossière des règles de la transformation en séparant l'aspect sémantique de celui des accès.

7.1.1.1 CREATION/SUPPRESSION D'UN TYPE D'ARTICLES INTERMEDIAIRE DANS UN TYPE D'ASSOCIATIONS.

Définition.

Ces transformations sont basées sur le principe universellement connu d'agrégation/désagrégation et consistent essentiellement à remplacer un type d'associations par un type d'articles et inversément.

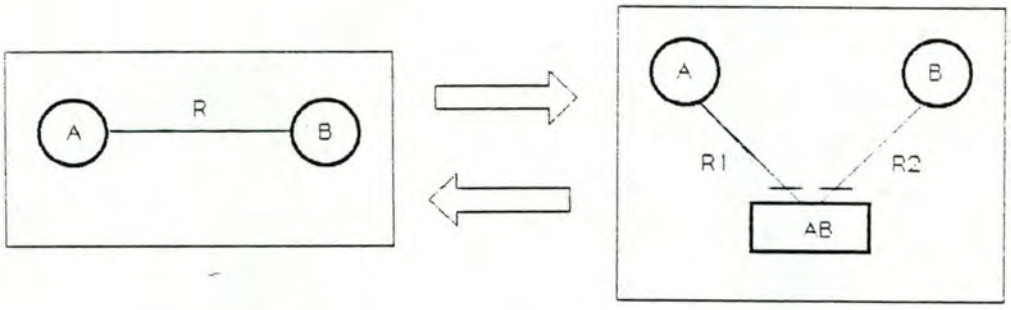


Figure 7.1: Création/suppression d'un type d'articles.

Création d'un type d'articles.

Objectif.

L'introduction d'un nouveau type d'articles dans un type d'associations est un procédé classique d'élimination de types de chemins N-M ou récursifs, d'identifiants ou de clés d'accès secondaires, de clés d'accès non identifiantes.

Précondition.

A ou B est un type d'articles.

Aspect sémantique.

Si deux types d'objets A et B sont reliés entre eux par un type d'associations R, celui-ci peut être remplacé par un type d'articles et deux nouveaux types d'associations R1 et R2 qui seront obligatoires pour le nouveau type d'articles et de plus l'identifieront. Les propriétés de R1 et R2 sont directement déductibles des propriétés de R sauf pour leur classe fonctionnelle puisque plusieurs solutions pourront être proposées pour la traduction de la classe fonctionnelle de R.

Gestion des accès.

Les accès dans les nouveaux types d'associations sont directement déductibles des accès dans R.

Une décision devra cependant être prise concernant la traduction des clés d'accès ayant R comme composant non unique. Considérons par exemple le cas où itb est clé d'accès dans le type de chemins R.

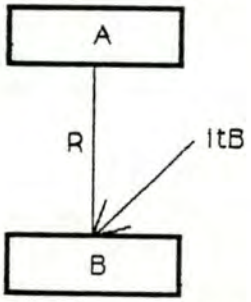


Figure 7.2: Clé d'accès dans un type de chemins à transformer.

Si on insère un type d'articles dans R, le schéma devient :

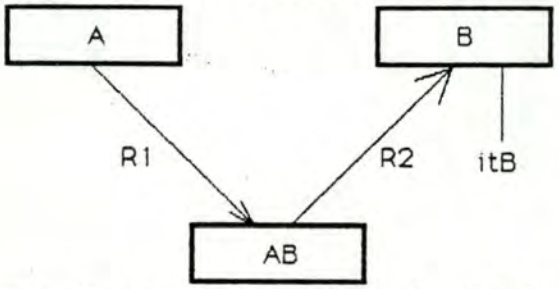


Figure 7.3: Clé d'accès dans un type de chemins transformé.

Le problème est de savoir comment convertir la clé d'accès dans le type de chemins transformé. Doit-elle être transformée indépendamment du fait qu'elle soit ou non identifiante ?

Si nous considérons que non, la seule solution possible est de remplacer dans la clé d'accès le composant R par un composant correspondant à la composition de R1 et R2. Cependant, la notion de type d'associations composé n'existe pas actuellement dans le MAG.

Si nous considérons que la clé d'accès doit être transformée en fonction de sa qualité d'être ou non identifiante, la traduction pourra dépendre de la solution qui aura été choisie pour la traduction de la classe fonctionnelle de R. Nous n'entrons pas dans les détails ici. Le lecteur intéressé par des propositions de traduction de telles clés d'accès est reporté aux documents de travail.

Suppression d'un type d'articles.

Objectif.

Cette transformation est essentiellement utile pour augmenter les performances d'accès, pour diminuer la modularité des structures de données ou pour permettre au concepteur un retour en arrière dans ses prises de décision.

Préconditions.

Le type d'articles à supprimer AB peut être considéré comme né de la transformation inverse analysée dans le point précédent. Plus précisément,

- aucun type d'associations autre que R1 et R2 ne porte sur AB;
- R1 et R2 sont obligatoires pour AB et de plus l'identifient;
- les combinaisons des classes fonctionnelles de R1 et R2 sont compatibles avec les règles de traduction de la transformation inverse.

Aspect sémantique.

La transformtion de création d'un type d'articles intermédiaire assurant l'équivalence sémantique, il en est de même pour son inverse. Il n'existera ici qu'une seule possibilité de transformation et dès lors toutes les propriétés du nouveau type de chemins sont directement déductibles de celles de R1 et R2.

Gestion des accès.

En supposant que les accès dans les types d'associations R1 et R2 sont compatibles avec les règles de traduction des accès dans le cas de la transformation inverse, les accès dans le nouveau type d'associations R sont directement déductibles des accès dans l'un des types d'associations R1 ou R2.

7.1.1.2 ROTATION/ROTATION INVERSE D'UN OU DE PLUSIEURS TYPES D'ASSOCIATIONS.

Définition.

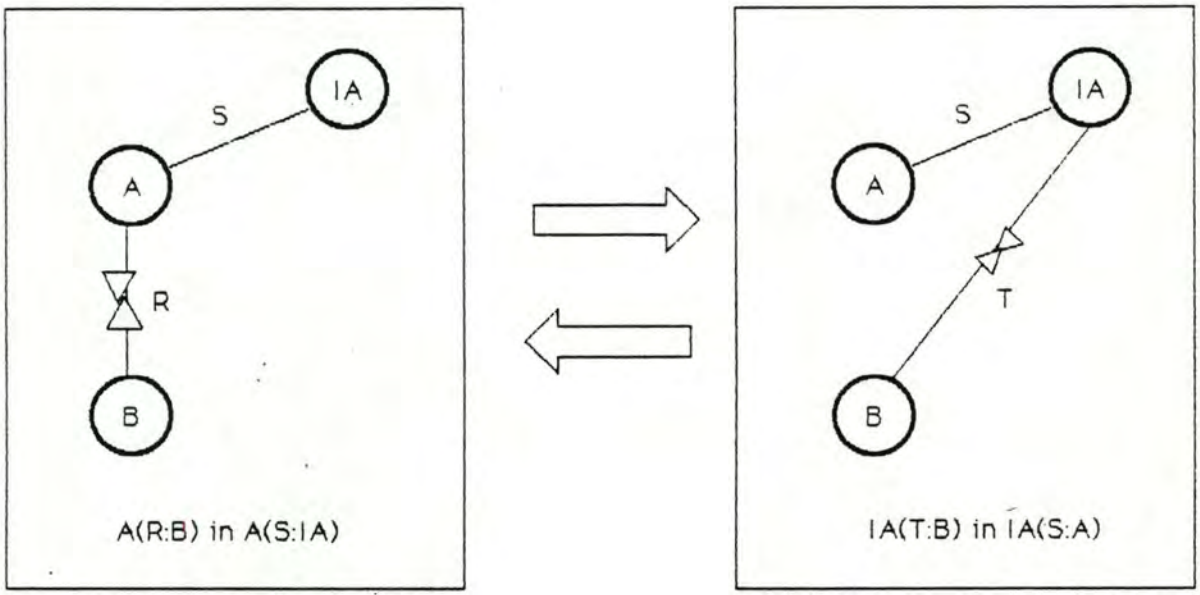


Figure 7.4: Rotation de types d'associations.

Si IA est un identifiant de A et si B est associé à A, on peut aussi bien l'associer à IA.

Il est à remarquer que IA peut avoir plusieurs composants ITEMs ou types de chemins. Dans un tel cas la rotation est dite MULTIPLE et pour que l'équivalence sémantique soit assurée, la classe fonctionnelle de R(A,B) doit être 1-1 ou 1-N. Si IA n'a qu'un seul composant on parle d'une rotation SIMPLE et R(A,B) peut alors avoir une classe fonctionnelle N-M.

Notons de plus que l'inverse d'une rotation simple est une rotation simple, tandis que l'inverse d'une rotation multiple d'un type d'associations est une rotation de plusieurs types d'associations en même temps.

Nous n'examinerons pas ici l'inverse d'une rotation multiple.

Objectif.

En prenant un identifiant IA composé uniquement d'ITEMs, cette famille servira notamment à éliminer des types de chemins . Si IA contient au moins un composant type de chemins, la rotation ne semble à priori utile que pour augmenter les performances d'accès.

Préconditions.

- A est un type d'articles;
- si B est un ITEM, chaque composant de IA est un type de chemins;
- si un des composants de IA est un ITEM, B est un type d'articles;
- si IA a plusieurs composants, R(A,B) a une classe fonctionnelle 1-1 ou N-1;
- il existe une contrainte d'intégrité exprimant que l'ensemble des A reliés à au moins un B est inclu dans l'ensemble des A qui ont une valeur pour tous les composants ITEMs de IA et qui participent dans tous les types de chemins formant les autres composants de IA. Remarquons que cette précondition est automatiquement vérifiée si tous les composants de IA sont obligatoires pour A.

Aspect sémantique.

Les propriétés des nouveaux types d'associations sont directement déductibles des propriétés de R. Une contrainte d'intégrité supplémentaire devra cependant être créée. Celle-ci, en supposant que IA n'a qu'un seul composant, pourra se formuler comme suit : l'ensemble des IA reliés à B est inclu dans l'ensemble des IA reliés à A. Le signe d'inclusion devra être remplacé par le signe d'égalité si R était obligatoire pour A. Cette contrainte n'aura de plus pas à être créée si S était obligatoire pour IA (en supposant que le S soit un type de chemins).

Gestion des accès.

Les accès dans les nouveaux types d'associations sont directement déductibles des accès dans R.

Pour que l'équivalence d'accès soit assurée, le type d'associations entre IA et A doit être bidirectionnel.

7.1.1.3 CREATION/SUPPRESSION D'UNE CONTRAINTE D'INTEGRITE.

Définition.

Ces transformations consistent à transférer la gestion d'une contrainte d'intégrité du SGD vers les programmes d'application (suppression), et inversément (création).

Objectif.

La suppression d'une contrainte d'intégrité d'un type donné est indispensable pour permettre la conformité à un SGD ne gérant pas un tel type de contraintes. La transformation inverse n'apparait a priori pas utile si ce n'est que pour permettre un retour en arrière dans les prises de décision.

Les contraintes d'intégrité pour lesquelles il pourrait être utile de prévoir ces transformations sont :

- la propriété pour un type d'associations d'être obligatoire ou facultatif pour l'un des objets sur lesquels il porte;
- la classe fonctionnelle d'un type d'associations;
- les identifiants formés de plus de x composants, éventuellement d'un type donné (ITEM ou type de chemins);
- les contraintes exprimant que deux types d'associations sont inverses.

Aspect sémantique.

Toute contrainte d'intégrité exprimée dans le schéma peut être supprimée sans perte de sémantique si l'on retient que celle-ci doit être gérée par programme d'application (et inversément).

7.1.1.4 CREATION/SUPPRESSION D'UN DOUBLE D'UN TYPE D'ARTICLES.

Définition.

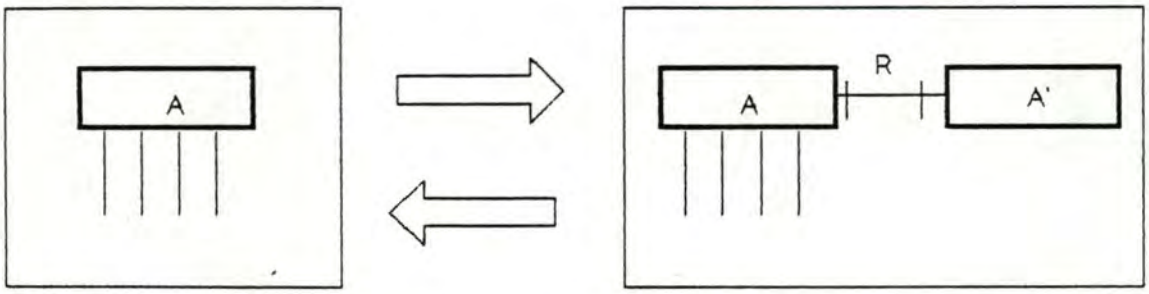


Figure 7.5: Création/suppression d'un double d'un type d'articles.

Ces transformations consistent à créer/supprimer un type d'articles qui n'est associé à aucun objet autre que sa duplication sémantique.

Objectif.

Il s'agit de transformations purement théoriques qui préparent/clôturent les transformations de transfert de types d'associations entre les deux types d'articles de même sémantique.

Création d'un double d'un type d'articles.

Précondition.

Cette transformation n'a de sens que si au moins deux types d'associations portent sur le type d'articles à dupliquer.

Aspect sémantique.

Le nouveau type de chemins R qui reliera A et A' sera un type de chemins "is-a". En d'autres termes, R sera de classe fonctionnelle 1-1 et obligatoire pour A et A'.

Notons que cette transformation introduit de la redondance sémantique qui ne disparaîtra que lorsqu'au moins un type d'associations sera transféré de A vers A'.

Gestion des accès.

L'équivalence d'accès est assurée quels que soient les accès définis dans R.

Suppression d'un double d'un type d'articles.

Cette transformation étant l'inverse de la précédente et de plus trop élémentaire, nous nous limiterons à en donner les préconditions pour que l'équivalence sémantique soit assurée.

Préconditions.

- le type d'articles à supprimer A' est relié à un type d'articles (soit A) par un type de chemins "is-a" (soit R);
- R est le seul type d'associations portant sur A'.

7.1.1.5 TRANSFERT DE TYPES D'ASSOCIATIONS ENTRE TYPES D'ARTICLES DOUBLES.

Définition.

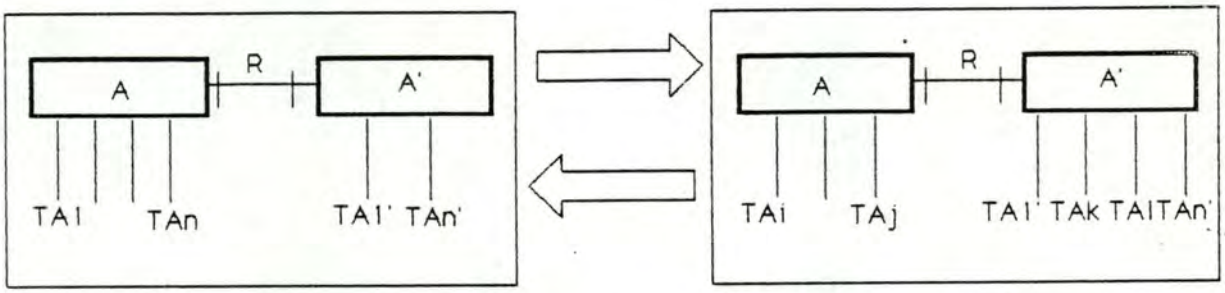


Figure 7.6: Transfert de types d'associations.

Cette transformation effectue un transfert de types d'associations entre types d'articles reliés par un type de chemins "is-a". Elle peut donc être considérée comme complémentaire aux transformations examinées dans le point précédent.

Objectif.

Le transfert de types d'associations entre types d'articles doubles servira notamment à éliminer des identifiants ou clés d'accès doubles, à augmenter/diminuer la modularité des structures de données si ce transfert suit/précède une création d'un double d'un type d'articles.

Préconditions.

- le type d'articles récepteur A' est relié par un type de chemins "is-a" au type d'articles émetteur A;
- si un type d'associations à transférer est composant d'un identifiant/clé d'accés, tous les autres composants de ce dernier sont transférés en même temps (par la même opération).

Aspect sémantique.

Les propriétés des types d'associations transférés sont inchangées si ce n'est qu'ils ne portent plus sur A mais sur A'.

Gestion des accès.

Les accès dans les types d'associations sont inchangés. Pour assurer l'équivalence, les accès dans le type de chemins "is-a" devront être ajustés en fonction de ceux dans le type d'associations à transférer. Cet ajustement sera cependant inutile si la convention a été prise de rendre ce type de chemins bidirectionnel lors de la transformation de création d'un double d'un type d'articles.

7.1.1.6 REGROUPEMENT/APLATISSEMENT D'ITEMs.

Ces transformations peuvent directement être considérées comme transformations concrètes. Etant analysées plus en détail dans la section suivante de ce chapitre, nous n'en donnerons ici qu'une idée de base.

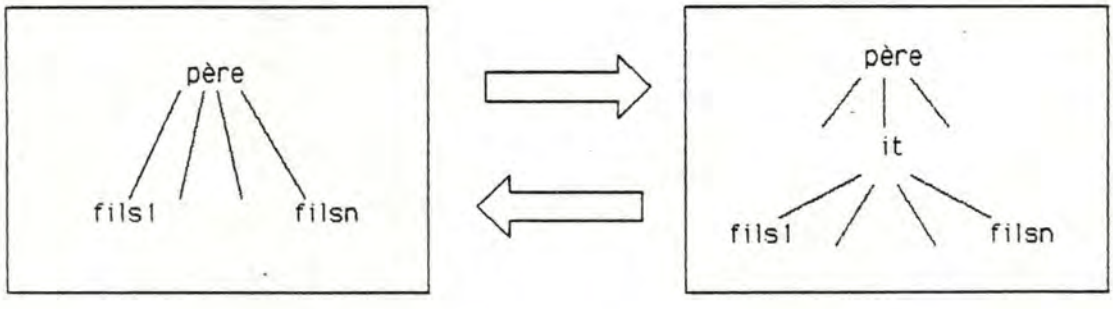


Figure 7.7: Regroupement/décomposition d'ITEMs.

Cette famille consiste à regrouper plusieurs ITEMs ayant même père en un ITEM décomposable et inversément, à aplatir un ITEM décomposable. Pour que l'équivalence sémantique soit assurée, l'ITEM décomposable sera simple et obligatoire.

7.1.1.7 ECLATEMENT/FUSION DE TYPES DE CHEMINS.

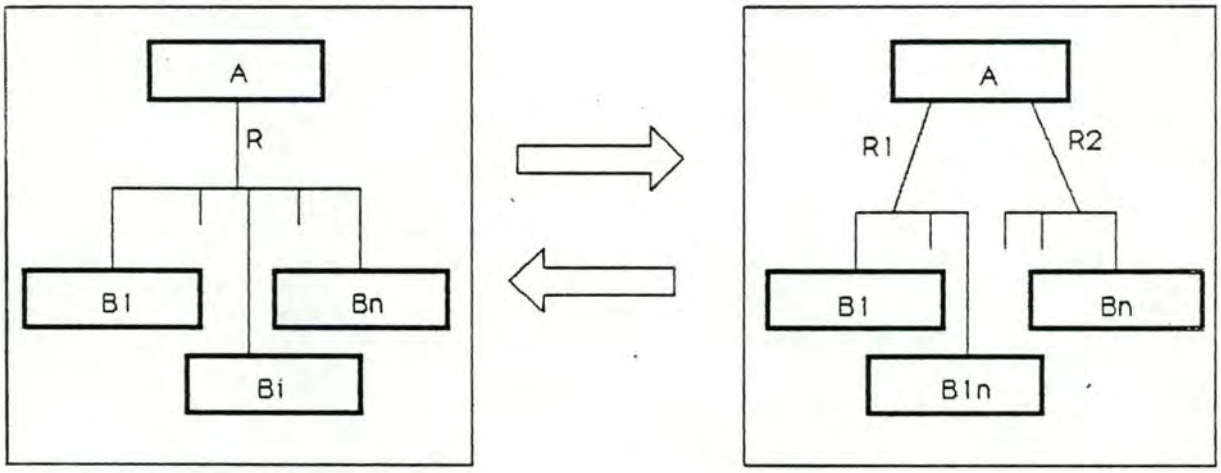


Figure 7.8: Eclatement/fusion de types de chemins.

Ces transformations ont pour effet d'éclater un type de chemins multirôle en deux types de chemins (éventuellement multirôles) et inversément, de fusionner deux types de chemins ayant une extrémité commune en un type de chemins multirôles. Remarquons que la transformation d'éclatement nécessite la gestion de contraintes d'exclusion (respectivement d'inclusion) si l'extrémité à éclater a une connectivité maximale égale à un (respectivement si le type de chemins est obligatoire pour l'extrémité à éclater).

7.1.1.8 TRANSFORMATIONS DE TYPES D'ASSOCIATIONS INVERSES.

Deux types d'associations inverses peuvent être transformés soit globalement (= perception sémantique) soit séparément (= perception d'accès). Dans ce dernier cas, lors de la transformation d'un type d'associations ayant un inverse, il sera nécessaire de distinguer les cas où le type d'associations inverse a déjà été transformé ou non et, dans l'affirmative, si c'était au moyen de la même technique. Cette nécessité vient du fait qu'il faudra traduire la contrainte d'intégrité exprimant la redondance initiale (deux types d'associations inverses expriment la même sémantique).

Ces transformations sont essentiellement utiles pour augmenter les performances d'accès et pour supprimer des clés d'accès secondaires.

7.1.2 EXTENSION DE LA LISTE DES FAMILLES DE TRANSFORMATIONS.

Ci-dessous sont proposées quelques familles de transformations que l'on pourrait ajouter à celles présentées jusqu'à présent, mais qui n'ont pas été considérées lors de la recherche des transformations concrètes pour l'une des raisons suivantes :

- soit qu'elles n'assurent l'équivalence sémantique que dans des conditions trop spécifiques les rendant inutiles;
- soit que leur étude serait d'une ampleur incompatible avec le présent mémoire.

On donnera ensuite quelques transformations préservant l'équivalence mais pour lesquelles celle-ci ne sera pas assurée lors de la mise en oeuvre. Ces transformations seront dès lors appelées opérations de mise à jour de la base de spécification.

Autres familles de transformations.

1. "Eclatement d'un type d'articles en deux types d'articles de sémantique plus spécifique et inversément, fusion de deux types d'articles en un seul moins spécifique".

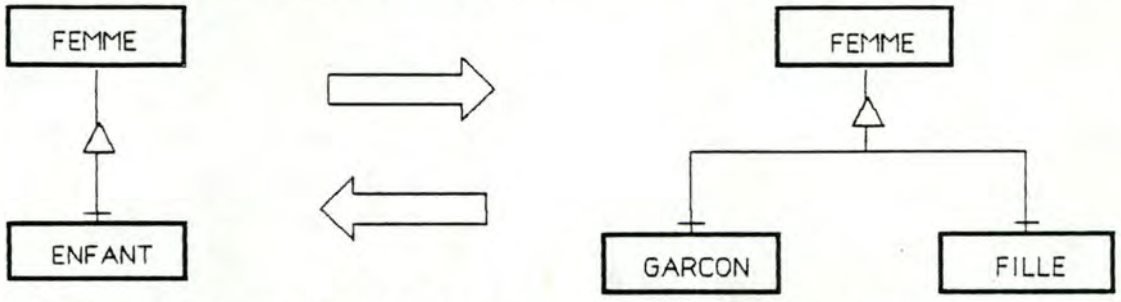


Figure 7.9: Exemple d'éclatement/fusion de types d'articles.

Il est à remarquer que l'équivalence sémantique pourrait être assurée si le type d'articles **ENFANT** était caractérisé par un **ITEM** permettant de distinguer les **FILLE**s des **GARCON**s.

Cette technique n'étant qu'un procédé parmi d'autres de spécialisation/généralisation, nous ne l'avons pas regrettablement pas analysée.

2. "Report de la répétitivité d'un ITEM sur ses fils et inversément."

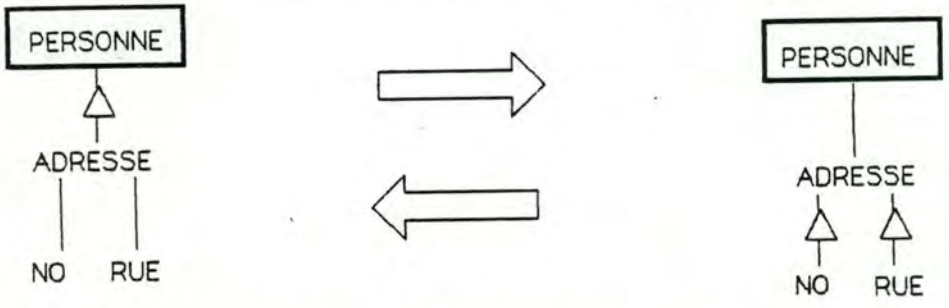


Figure 7.10: Exemple de report de répétitivité.

La correspondance entre le NO et la RUE d'une même ADRESSE est perdue si l'on reporte la répétitivité d'ADRESSE sur NO et RUE.

Pour éviter de perdre l'information de couplage entre fils d'un ITEM répétitif, la transformation de report de répétitivité ne pourra être appliquée que si l'ITEM répétitif n'a qu'un seul fils. Devant réduire cette transformation à ce cas, il n'y a aucune utilité pratique à l'analyser.

Opérations de mise à jour.

1. Création/suppression d'un objet technique (ITEM, fichier, identifiant, clé d'accès, clé d'ordre, ITEM compteur, ROLE, ...).

2. Changement de la valeur d'une propriété d'un objet dont l'effet n'entraine aucune modification de la sémantique du schéma.

   La propriété à modifier pourra en l'occurrence, être l'une des suivantes :

   - le nom d'un objet (type d'articles, type de chemins, ITEM, ...);
   - le format d'un ITEM;
   - la loi d'ordre d'une clé d'ordre;
   - la loi d'ordre des doubles d'une clé d'accès ou d'ordre;
   - l'emplacement d'un objet dans un ensemble ordonné;
     un tel ensemble pourra être formé :
     - de tous les ITEMs ayant le même père;
     - de tous les composants ITEM ou type de chemins d'une clé d'ordre.

3. Insertion/retrait d'un type d'articles d'un fichier.

4. Ajout/suppression d'un sens d'accès dans un type de chemins.

5. Création/suppression d'une clé d'accès ou d'ordre.

6. Ajout/suppression d'un composant ITEM/type de chemins à un identifiant technique, une clé d'accès ou une clé d'ordre.

7.1.3 CONCEPTS MAG NON PRIS EN COMPTE.

Etant donné l'ampleur et la difficulté de trouver des règles de transformation tenant compte de tous les concepts MAG et de tous les cas théoriques possibles, nous avons décidé :

1. de limiter l'application des transformations à des cas d'utilité pratique.

   Considérons par exemple la transformation de création d'un type d'articles intermédiaire dans un type d'associations. Il s'agit d'une transformation qu'il sera indispensable de mettre en oeuvre puisqu'elle est la seule qui permette notamment d'éliminer des ITEMs répétitifs.

   Un problème se pose si le type d'articles intermédiaire doit être introduit dans un type d'associations CONTAINS entre un ITEM et un type d'articles et que cet ITEM est composant non unique d'un identifiant ou d'une clé d'accès.

   Lors de l'étude des transformations concrètes, nous poserons l'absence de tels identifiants/clés d'accès comme précondition d'application d'une telle transformation. Il est à remarquer que cette restriction n'est pas la seule possible. On aurait pu tout aussi bien interdire au concepteur de définir des identifiants/clés d'accès répétitifs ou de ne lui permettre d'en définir que si l'ITEM répétitif en est l'unique composant.

2. de ne pas prendre en compte dans les règles de transformation des concepts suivants : sous-schéma, ordre, fichier, identifiant/clé d'accès global, contraintes d'intégrité autres que celles prises en compte en particulier par chaque famille.

CONCLUSION.

Le jeu de transformations élémentaires sur le MAG proposé ci-dessus est vaste mais n'est peut-être pas complet. Il couvre cependant l'essentiel des besoins pouvant se présenter en pratique. La plupart des structures non validées par COBOL, CODASYL et SQL peuvent être éliminées au moyen des deux grandes familles de base à savoir, la création d'un type d'articles intermédiaire dans un type d'associations et la rotation.

Ce chapitre ayant essentiellement pour but de découvrir un ensemble le plus complet possible de transformations assurant l'équivalence sémantique et éventuellement celle au niveau des accès, ces transformations ont été exprimées en utilisant des termes génériques. Pour éviter de lasser le lecteur par la lourdeur d'expression des règles détaillées, nous n'avons donné ici qu'une idée de base assez grossière de chaque famille de transformation théorique. La section suivante analyse les transformations plus en profondeur dans les termes du concepteur.

7.2 LES TRANSFORMATIONS CONCRETES.

Ayant mis en évidence un ensemble assez vaste de transformations de base assurant l'équivalence sémantique et dans la plupart des cas l'équivalence d'accès, nous définissons les transformations qui seront mises à la disposition du concepteur. Elles sont appelées transformations concrètes par différence avec les transformations abstraites ou théoriques analysées dans la section précédente. Il sera important de donner à chaque opérateur de transformation concrète une expression à la fois simple, précise et brève.

On ne reprendra pas ici les transformations qui seraient réalisées par de simples primitives de mise à jour de la base de spécification telles que la création/suppression d'ITEMs techniques éventuellement identifiants ou de fichiers, l'insertion/retrait d'un ou plusieurs types d'articles dans un fichier, la modification des noms d'objets.

Toutes les constructions MAG non conformes mises en évidence dans le chapitre 6 (Notion de conformité) peuvent a priori être éliminées en utilisant uniquement les transformations concrètes suivantes :

- transformation d'un ITEM en un type d'articles;
- aplatissement d'un ITEM décomposable;
- regroupement d'ITEMs en un ITEM décomposable;
- transformation d'un type de chemins en un type d'articles;
- transformation d'un type de chemins par duplication d'ITEMs;
- transformation d'un accès par clé en un accès par type de chemins;
- remplacement d'une connectivité restrictive d'un membre d'un type de chemins par une contrainte d'intégrité à gérer par programme d'application;
- transformation d'un ITEM obligatoire en un ITEM facultatif accompagné d'une contrainte d'intégrité spécifiant la possibilité d'absence de valeur.

Les transformations particulièrement utiles pour permettre la conformité à un SGD seront appelées transformations de conformité.

Le concepteur désirera effectuer des transformations, non seulement pour une raison de conformité au SGD, mais également pour satisfaire à des critères tels que la clarté des structures de données, la minimisation du temps d'accès et de mise à jour de la BD ou de la place mémoire occupée. Les transformations de conformité et celles permettant essentiellement de répondre à ces critères seront toutes appelées transformations d'affinage de schémas.

Dans le premier paragraphe de cette section on analysera en détail les transformations reprises dans la liste donnée ci-dessus. Dans un second paragraphe on établira une correspondance entre les restrictions qui peuvent être posées par un SGD sur les structures de données MAG et ces transformations. L'étude de cette correspondance montrera que cette liste, bien que non exhaustive, est suffisante pour satisfaire l'objectif

primordial de conformité aux SGDs COBOL, CODASYL et SQL. On en profitera pour mettre en évidence, en fonction de chacun de ces SGDs en particulier, quelques macro-transformations construites à partir de ces dernières qui éviteront d'introduire de nouvelles structures non conformes. Dans le troisième paragraphe, nous proposerons d'autres transformations d'affinage de schémas.

Nous tenons à nous excuser d'une redondance possible entre ces trois sections. La lecture en est cependant moins ardue.

7.2.1 ETUDE DES TRANSFORMATIONS CONCRETES.

Dans cette section, on donnera à chaque transformation concrète de la liste donnée dans l'introduction : sa définition en termes de transformations abstraites, son objectif ou utilité exprimé essentiellement par rapport à l'objectif primordial de conformité, ses données en entrée et préconditions d'application pour que l'équivalence sémantique et d'accès soit assurée, une représentation graphique accompagnée des règles de dérivation des propriétés des nouveaux objets. Les règles introduites dans la représentation graphique de la transformation pourront ne pas être reprises dans ce dernier point. Les préconditions portant sur l'identification des objets qui constituent les données en entrée (unicité des noms des objets à créer par la transformation, relations entre les objets intervenant dans la transformation) pourront également être omises. La rédaction de l'exposé des transformations concrètes s'inspire de (BRES, 85).

7.2.1.1 Transformation d'un ITEM en un type d'articles.

Définition.

Cette transformation est un cas particulier de l'introduction d'un type d'articles intermédiaire dans un type d'associations. Dans ce cas particulier, il s'agit d'un type d'associations entre un type d'articles et un ITEM principal.

Objectif.

La transformation d'un ITEM en un type d'articles est principalement utile pour éliminer des ITEMs répétitifs ou facultatifs, des clés d'accès non identifiantes et des identifiants ou clés d'accès secondaires.

Données en entrée.

- IA : nom de l'ITEM à transformer;
- A : nom du type d'articles contenant IA;
- B : nom du nouveau type d'articles;
- R : nom du nouveau type de chemins;
- si IA est non identifiant, s'il garde ou non cette propriété dans B;
- si IA est répétitif, s'il garde ou non cette propriété dans B.

Préconditions.

- IA est un ITEM principal;
- si IA fait partie d'un identifiant ou d'une clé d'accès, il en est l'unique composant.

Représentation graphique.

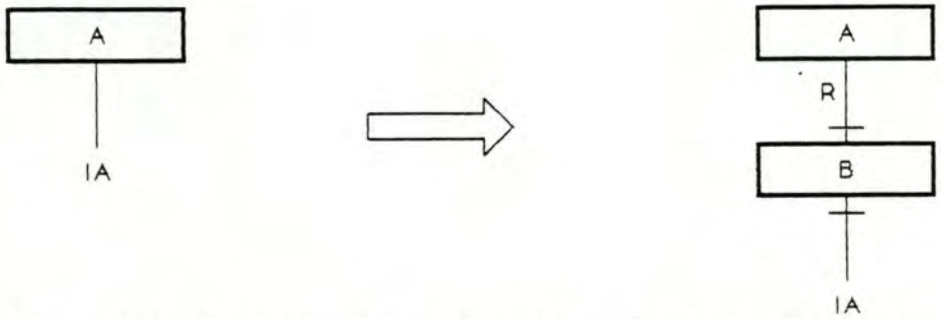


Figure 7.11: Transformation d'un ITEM en un type d'articles.

Règles.

Règles générales.

- R et IA sont obligatoires pour B;
- Si IA (ou un sous-ensemble de ses descendants) était identifiant de A, il le devient pour B et la connectivité maximale de B dans R (CmaxB) est 1;
  sinon, soit IA est non identifiant dans B et CmaxB = 1, soit IA devient identifiant de B et CmaxB = N
- Si IA était non répétitif dans A, il garde cette propriété dans B et CmaxA = 1,
  sinon soit IA garde sa répétitivité dans B et CmaxA = 1, soit IA est non répétitif dans B et CmaxA = N.
  Dans ce dernier cas, si la répétitivité de IA dans A n'est pas illimitée, les connectivités de A devront être adaptées.

Remarque.

Nous ne proposons pas ici de conserver IA dans A puisque ceci reviendrait à transformer uniquement son éventuelle qualité de clé d'accès et que cette possibilité sera offerte par la transformation d'un accès par clé en un accès par type de chemins analysée en

7.2.1.6.

Gestion des accès.

- R servira toujours d'accès pour passer de A à B;
- si IA était clé d'accès dans A, alors IA sera clé d'accès dans B et R servira d'accès pour passer de B à A.

Identifiants supplémentaires.

Si IA était non identifiant et répétitif dans A et ne garde dans B que la première de ces propriétés, IA devient identifiant de B dans le nouveau type de chemins R.

Remarque : la création d'un tel identifiant est toujours correcte parce que le MAG est un modèle ensembliste. Cet identifiant ne serait pas toujours justifié si nous supposions qu'un ITEM répétitif peut prendre plusieurs fois la même valeur pour un même article. Un exemple d'une telle situation est donné par la figure suivante :

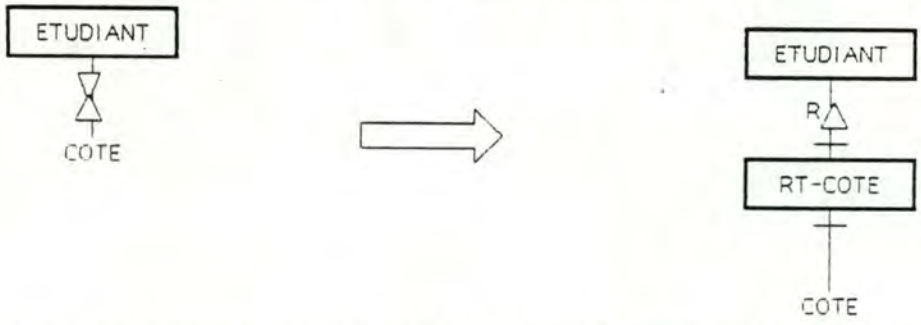


Figure 7.12: Cas d'un ITEM répétitif à valeurs eventuellement doubles

Gestion de l'éventuel ITEM compteur de répétitivité variable.

Si IA est un ITEM répétitif dont la valeur de la répétitivité est donnée pour chaque article A au moyen d'un ITEM compteur, cette information devra être traduite dans le nouveau schéma. La traduction de cet ITEM compteur devra tenir compte des remarques importantes suivantes :

- la définition du MAG impose à l'ITEM compteur de se trouver dans le même type d'articles que l'ITEM compté;
- il est possible qu'un ITEM soit compteur pour plusieurs autres ITEMs;
- la suppression d'un ITEM compteur n'assure l'équivalence sémantique que si celui-ci n'est compteur que d'un seul ITEM.

7.2.1.2 Aplatissement d'un ITEM décomposable.

Définition.

Cette transformation est en relation biunivoque avec la transformation abstraite de même nom.

Objectif.

L'aplatissement d'un ITEM décomposable est primordiale pour permettre la mise à plat d'un type d'articles requise par un SGD tel que SQL. Elle consiste principalement à remplacer un ITEM par ses composants.

Données en entrée.

- IA : nom de l'ITEM à aplatir;
- A : nom du type d'articles contenant IA.

Précondition.

IA est un ITEM obligatoire et non répétitif.

Représentation graphique.

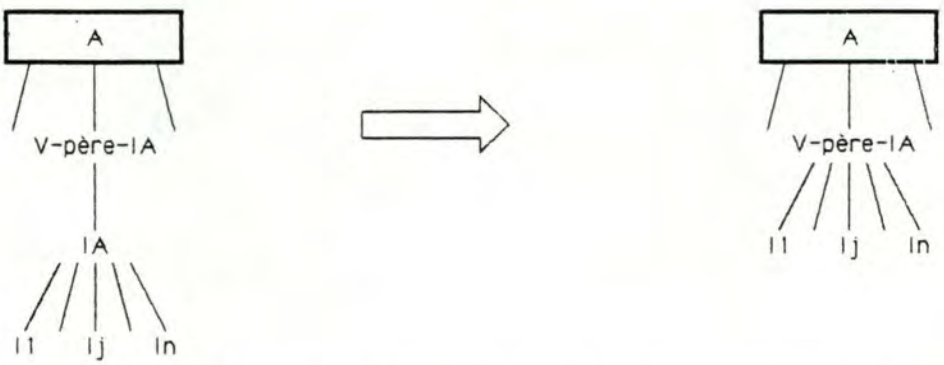


Figure 7.13: Aplatissement d'un ITEM décomposable.

Règles.

- Les fils de IA deviennent fils du père de IA et s'insèrent dans la séquence à l'endroit où se trouvait IA.
- IA est remplacé par ses fils dans tous les identifiants et clés d'accès qui l'ont comme composant.

7.2.1.3 Regroupement d'ITEMs en un ITEM décomposable.

Définition.

Cette transformation est l'inverse de la précédente et est également en relation biunivoque avec la transformation abstraite de même nom.

Objectif.

Le regroupement d'ITEMs en un ITEM décomposable est principalement utile pour éliminer des identifiants/clés d'accès formés de plusieurs ITEMs.

Données en entrée.

- IA : nom du nouvel ITEM décomposable;
- A : nom du type d'articles contenant IA;
- I1 I2 ... In : ITEMs à regrouper en IA;
- choix de la nouvelle composition des identifiants/clés d'accès formés de l'ensemble des ITEMs I1 I2 ... In.

Précondition.

Les ITEMs I1 I2 ... et In ont le même père.

Représentation graphique.

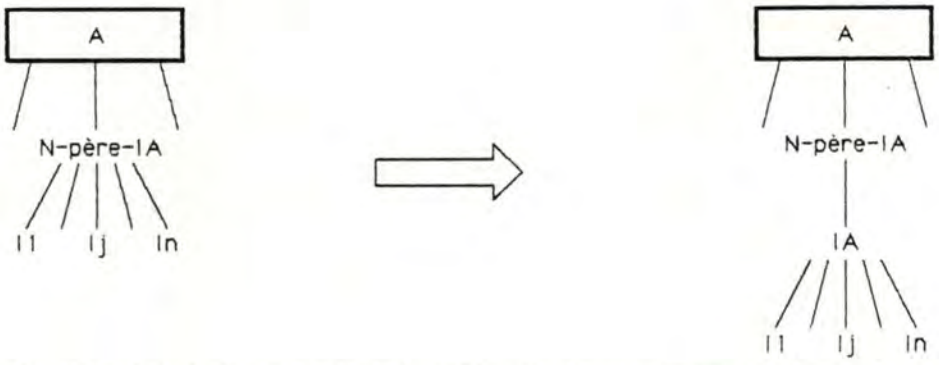


Figure 7.14: Regroupement d'ITEMs en un ITEM décomposable.

Règles.

- IA sera obligatoire et non répétitif.
- I1 I2 ... In conservent leurs propriétés.
- Dans chaque identifiant/clé d'accès formé de I1 I2 ... In, l'ensemble de ces composants pourra être remplacé par un composant unique correspondant à IA.

7.2.1.4 Transformation d'un type de chemins en un type d'articles.

Définition.

Cette transformation est un cas particulier de la création d'un type d'articles intermédiaire dans un type d'associations.

Objectif.

La transformation d'un type de chemins en un type d'articles sert essentiellement à éliminer des types de chemins "Plusieurs-à-Plusieurs" et des types de chemins récursifs.

Données en entrée.

- R(A,B) : identification du type de chemins à transformer (le nom du type de chemins et les noms des deux types d'articles membres);
- R1 : nom du nouveau type de chemins portant sur A et le nouveau type d'articles correspondant à R;
- R2 : nom du nouveau type de chemins portant sur B et le nouveau type d'articles correspondant à R;
- éventuellement, choix de la traduction de la classe fonctionnelle de R (si celle-ci n'est pas 1-1).

Un nom par défaut sera donné au nouveau type d'articles : soit RTI.

Préconditions.

Il pourra exister des préconditions sur les identifiants/clés d'accès définis dans R. Ces préconditions dépendront uniquement de la solution qui aura été choisie pour la traduction de la classe fonctionnelle de R. Nous n'entrons pas dans les détails ici. Le lecteur intéressé est reporté aux documents de travail.

Règles.

Etude de la classe fonctionnelle de R1 et R2.

Différentes solutions seront proposées en fonction de la classe fonctionnelle de R.

Si R est "Plusieurs-à-Plusieurs".

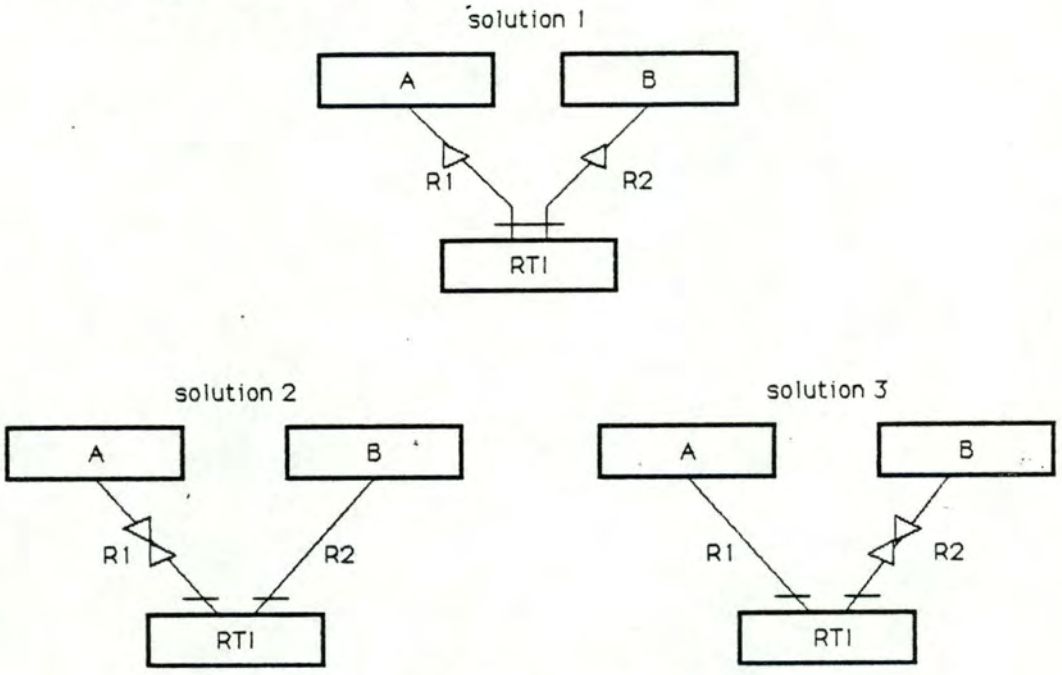


Figure 7.15: Cas d'un type de chemins N-M.

Si R est "Un-à-Plusieurs".
Supposons A le membre de connectivité maximale "Plusieurs".

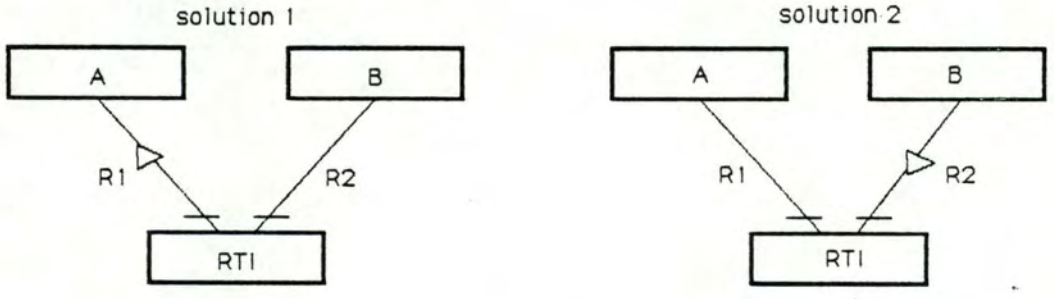


Figure 7.16: Cas d'un type de chemins 1-N.

Si R est "Un-à-Un".

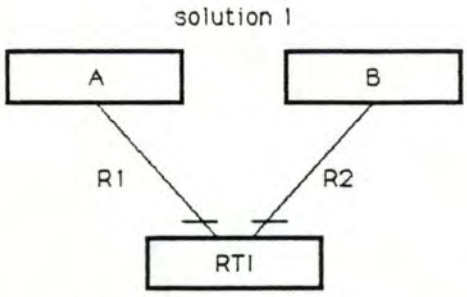


Figure 7.17: Cas d'un type de chemins 1-1.

Etude des contraintes d'existence dans R1 et R2.

- R1 et R2 seront obligatoires pour RTI.
- R1 (respectiviement R2) sera obligatoire pour A (respectivement pour B) si R était obligatoire pour A (respectivement pour B).

Nouveaux identifiants.

Un identifiant formé de R1 et R2 sera créé pour RTI si R était de classe fonctionnelle N-M et que la solution choisie rend les classes fonctionnelles de R1 et R2 1-N.

Gestion des accès.

Si R servait d'accès pour passer de A à B (respectivement de B à A), R1 servira d'accès pour passer de A à RTI (respectivement de RTI à A) et R2 servira d'accès pour passer de RTI à B (respectivement de B à RTI).

Remarque.

Si R est récursif, il sera nécessaire d'identifier les membres A et B par leur ROLE respectif dans R.

7.2.1.5 Transformation d'un type de chemins par duplication d'ITEMs.

Définition.

Cette transformation est un cas particulier de la rotation d'un type d'associations. Dans le cas présent, le type d'associations est un type de chemins et l'identifiant qui sert de base à sa rotation est un ITEM (ou groupe d'ITEMs)(a).

Objectif.

La transformation d'un type de chemins par duplication d'ITEMs est particulièrement utile pour éliminer des types de chemins et pour réduire une hiérarchie.

Données en entrée.

- IA : nom de l'ITEM à dupliquer (IA peut ne pas exister);
- A : nom du type d'articles contenant IA;
- R(A,B) : identification du type de chemins à transformer;
- IA' : nom du nouvel ITEM, duplication de IA.

Précondition.

IA, s'il existe, est un ITEM simple, obligatoire, identifiant et clé d'accès. La contrainte que IA soit obligatoire a été posée pour éviter de vérifier une précondition plus complexe qui impose l'existence d'une contrainte d'intégrité spécifiant que tout A relié par R à B possède une valeur pour IA.

Représentation graphique.

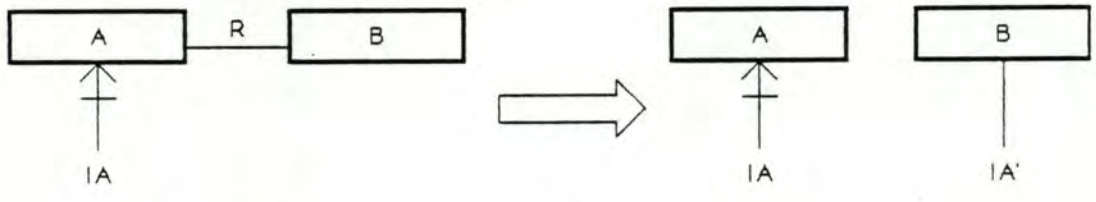


Figure 7.18: Transformation d'un type de chemins par duplication d'ITEMs.

(a) Dans cet exposé on supposera que l'identifiant n'est composé que d'un seul ITEM.

Règles.

Règles générales.

- Si IA n'existe pas dans A, il sera créé comme ITEM simple, obligatoire, identifiant et clé d'accès.

- Propriétés de IA'.

  - IA' est obligatoire si et seulement si R est obligatoire pour B (sachant que IA est obligatoire pour A).

  - IA' est répétitif (de répétitivité variable illimitée) si la connectivité maximale de B dans R est "Plusieurs". Il sera simple sinon.

  - IA' est identifiant si la connectivité maximale de A dans R est "Un".

Gestion des accès.

IA' sera clé d'accès si R servait d'accès pour passer de A à B.

Il est à remarquer que, étant donné la précondition que IA est clé d'accès dans A, la transformation de R par duplication d'ITEMs n'assure pas l'équivalence d'accès si R sert d'accès pour passer de A à B mais que l'accès inverse n'existe pas.

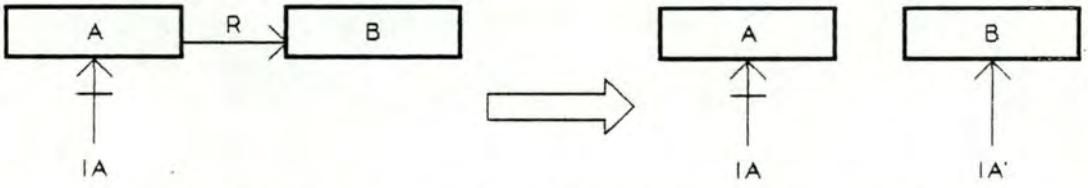


Figure 7.19: Cas où R ne sert d'accès que de A vers B.

En effet, même si A n'est pas accessible à partir de B dans R, il pourra être accédé à partir de B dans le schéma transformé en utilisant l'accès implicite de B vers IA' et la clé d'accès IA.

Contrainte d'intégrité supplémentaire.

L'ensemble des valeurs prises par IA' devra être inclu dans celui de IA. Le signe d'inclusion sera remplacé par le signe d'égalité si R est obligatoire pour A.

Remarques.

1. Si R est bidirectionnel, cette transformation pourrait également proposer une solution où l'identifiant de chaque membre de R est dupliqué.

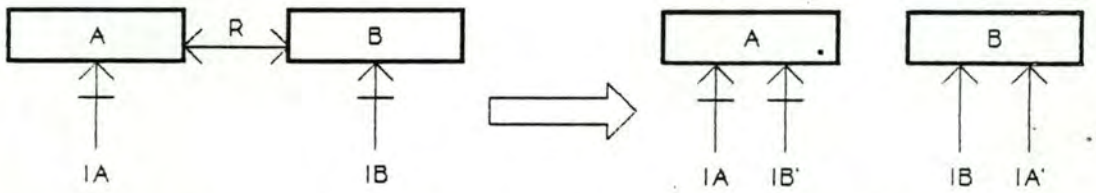


Figure 7.20: Duplication des identifiants de chaque membre de R.

Une telle solution n'a pas été proposée ici puisqu'elle pourra être obtenue en utilisant les transformations de types de chemins inverses qui consistent à transformer séparément chaque sens d'accès (cfr. les transformations d'affinage).

2. Etude du nom par défaut de IA'.

IA' pourra par défaut avoir le même nom que IA. Ceci ne sera cependant pas permis si R est récursif puisque IA et IA' appartiendront au même type d'articles. La donnée en entrée "nom du type d'articles A contenant IA" n'est dans ce cas particulier plus pertinente pour déterminer le sens de la duplication. Si R est récursif, cette donnée doit être remplacée par le nom du rôle que doit représenter le nouvel ITEM.

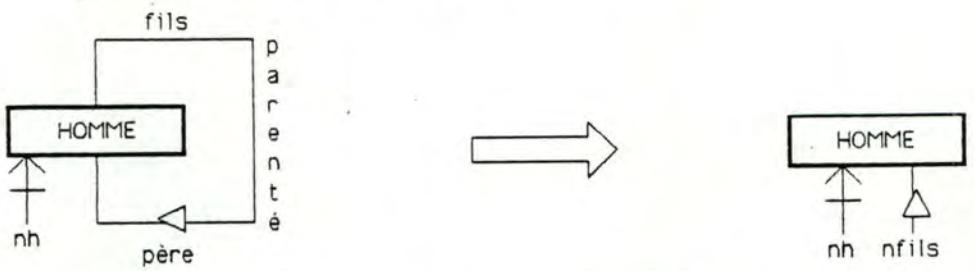


Figure 7.21: Cas d'un type de chemins récursif.

Dans cet exemple, le nouvel ITEM pourra représenter soit le rôle père soit le rôle fils.

Si R est récursif, nous proposons pour la formation du nom par défaut du nouvel ITEM d'ajouter le préfixe "n" au nom du rôle qu'il représente (en supposant qu'aucun ITEM de B ne porte ce nom). Cette convention ne pourra cependant pas être appliquée si deux types de chemins récursifs définis pour un même type d'articles peuvent avoir un nom de rôle identique. Si une telle situation est acceptée, la règle d'unicité des noms d'ITEMs risque en effet d'être violée.

Si l'on désire une convention qui soit applicable dans tous les cas nous devrons :

- soit imposer comme précondition d'application de la transformation d'une part, que deux types de chemins récursifs définis sur un même type d'articles aient des noms de rôles distincts, et d'autre part l'absence de type de chemins symétrique (types de chemins dans lequ le rôle de chaque extrémité est identique),

- soit prendre un nom par défaut pour le nouvel ITEM qui soit la concaténation du nom du type de chemins et du nom du rôle qu'il représente.

La forme canonique du modèle E/A offerte par IDA assurant que deux rôles, même de types d'associations différents n'auront pas le même nom, nous suggérons la première proposition.

3. Si R est récursif, les règles définies ci-dessus doivent être reformulées en remplaçant A par le nom du rôle de R représenté par le nouvel ITEM et B par l'autre rôle.

7.2.1.6 Transformation d'un accès par clé en un accès par type de chemins.

Définition.

Cette transformation est un cas particulier d'introduction d'un type d'articles intermédiaire dans un type d'associations inverse. La qualité de clé d'accès d'un ITEM constitue ici le type d'associations inverse.

Objectif.

La transformation d'un accès par clé en un accès par type de chemins est particulièrement utile pour éliminer des clés d'accès secondaires. Elle n'est cependant pas indispensable puisque la transformation d'un ITEM en un type d'articles (analysée en 7.2.1.1.) peut être utilisée pour ces mêmes raisons. Les solutions de ces deux transformations pourraient de plus être facilement fusionnées en étendant la transformation d'un ITEM en un type d'articles de faon à donner la possibilités de le conserver dans son type d'articles initial s'il est clé d'accès.

Données en entrée.

- IA : nom de l'ITEM clé d'accès à transformer;
- A : nom du type d'articles contenant IA;
- B : nom du nouveau type d'articles;
- R : nom du nouveau type de chemins;
- IA' : nom du nouvel ITEM (par défaut, celui de IA).

Préconditions.

Les préconditions sont les mêmes que celles de la transformation d'un ITEM en un type d'articles à l'exclusion de la dernière. Celle-ci n'a en effet plus de raison d'être puisque IA sera conservé dans A.

Représentation graphique.

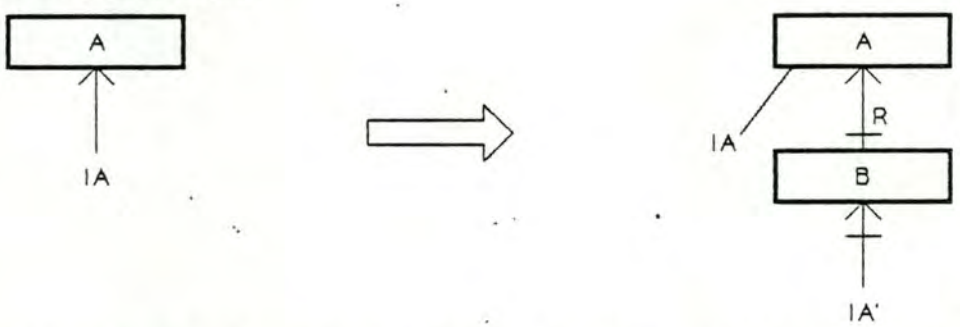


Figure 7.22: Transformation d'un accès par clé en un accès par type de chemins.

Règles.

Les règles seront presque les mêmes que celles pour la transformation d'un ITEM en un type d'articles. Les seules différences résideront dans la gestion des accès, dans le fait que IA sera conservé dans A et qu'une contrainte de référence entre IA et IA' devra être créée.

Gestion des accès.

- IA perd sa qualité de clé d'accès et IA' est clé d'accès à B;
- R sert d'accès pour passer de B à A

Contrainte d'intégrité supplémentaire.

La contrainte de référence suivante doit être créée : "l'ensemble des valeurs prises par IA' est égal à l'ensemble des valeurs prises par IA".

Remarques.

Rappelons que cette transformation introduit de la redondance d'accès puisque les ITEMs IA et IA', porteurs de la même sémantique, sont tous deux accessibles à partir de B. La contrainte de référence donnée ci-dessus n'est donc pas suffisante puisqu'elle n'exprime pas que

"pour tout article de type B (soit b), l'ensemble des valeurs de IA' doit être égal à l'ensemble des valeurs prises par IA pour l'article de type A relié à b.

La transformation d'un accès par clé en un accès par type de chemins a été analysée ci-dessus pour la technique de création d'un type d'articles intermédiaire dans un type d'associations inverse représenté par la qualité pour un ITEM d'être clé d'accès. Notons que si le type d'articles A est relié à un autre type d'articles (soit B) par un type de chemins bidirectionnel, de classe fonctionnelle 1-1 et obligatoire pour B, l'accès de IA vers A peut également être transformé en un accès par type de chemins en utilisant la technique de rotation pour le type d'associations inverse représenté par la clé d'accès comme le montre la figure suivante.

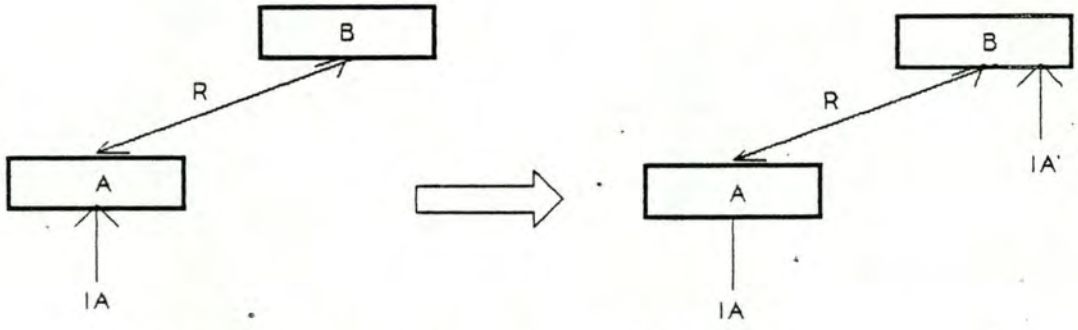


Figure 7.23: Rotation d'une clé d'accès.

Remarquons que cette transformation introduit également de la redondance d'accès puisque IA et IA' sont tous deux accessibles à partir de A.

7.2.2 TRANSFORMATIONS DE CONFORMITE.

L'objectif primordial des transformations de structures de données est de permettre d'aboutir à un schéma conforme à un SGD. Les principales constructions MAG qui peuvent ne pas être validées par un SGD ont éte mises en évidence dans le chapitre 6 (Notion de conformité).

Pour aider le concepteur à choisir les transformations à appliquer, nous lui fournissons dans ce pararaphe une correspondance entre les constructions MAG non conformes et les transformations permettant de les éliminer, celles-ci étant à choisir dans l'ensemble des transformations analysées ci-dessus.

L'étude de cette correspondance permettra également demontrer que toutes les constructions non conformes peuvent être éliminées en appliquant uniquement des transformations appartenant à cet ensemble.

Il est important de remarquer qu'une transformation éliminant une structure invalide peut, en fonction du SGD, introduire de nouvelles incompatibilités. La correspondance tient compte de ce fait et propose pour COBOL,CODASYL et SQL des solutions qui permettent un rapprochement strict vers la conformité.

Tout comme dans le chapitre "Notion de conformite", les contraintes sur la formation des noms d'objets, sur les notions d'ordre et de fichier, de même que les contraintes sur les notions d'identifiant/clé d'accès trop spécifiques à un SGD particulier, ne sont pas considérées.

7.2.2.1 Contrainte sur la notion de type d'articles.

Certains SGDs imposent que tout type d'articles contienne au moins un ITEM. Pour permettre de satisfaire à cette contrainte, il suffit de prévoir une transformation d'ajout d'un ITEM technique à un type d'articles. Cette transformation n'a pas été insérée dans la liste de celles présentées dans le paragraphe précédent parce qu'il s'agit d'une simple primitive de mise à jour de la base des spécifications.

Si le type d'articles sans ITEM est relié à un autre au moyen d'un type de chemins 1-1, obligatoire pour ces deux types d'articles, il pourrait être intéressant de proposer l'agrégation de ces derniers. Cette transformation assure l'équivalence sémantique et d'accès puisqu'en termes théoriques elle correspond à la fusion de deux types d'articles doubles (transformation analysée dans le point 7.2.1.4.). Il est à remarquer que cette transformation a été omise volontairement dans la liste proposée dans l'introduction parce qu'elle n'est pas indispensable pour permettre la conformité à un SGD.

7.2.2.2 Contraintes sur la notion d'ITEM.

LES ITEMS DECOMPOSABLES.

Les ITEMs décomposables pourront être éliminés au moyen de la transformation d'aplatissement d'ITEMs.

LES ITEMS FACULTATIFS.

La transformation d'un ITEM en un type d'articles permet d'éliminer des ITEMs facultatifs tout en introduisant un nouveau type de chemins. Cette transformation ne permet pas un rapprochement strict vers la conformité à des SGDs tels que COBOL et SQL qui ne gèrent pas la notion de type de chemins. La transformation suivante, plus rapide, est dès lors également proposée : rendre obligatoire les ITEMs facultatifs et leur associer une contrainte d'intégrité spécifiant la possibilité d'absence de valeur.

Remarquons que cette dernière transformation, bien que reprise dans la liste des transformations donnée dans l'introduction, pourrait être reportée au moment de la traduction du schéma dit "conforme" en un texte SGD puisque son seul effet est de transférer la gestion d'une contrainte d'intégrité vers les programmes d'application.

LES ITEMS REPETITIFS.

La répétitivité d'un ITEM peut facilement être éliminée en transformant cet ITEM en un type d'articles. Il suffit de choisir une solution où la répétitivité est reportée sur le nouveau type de chemins.

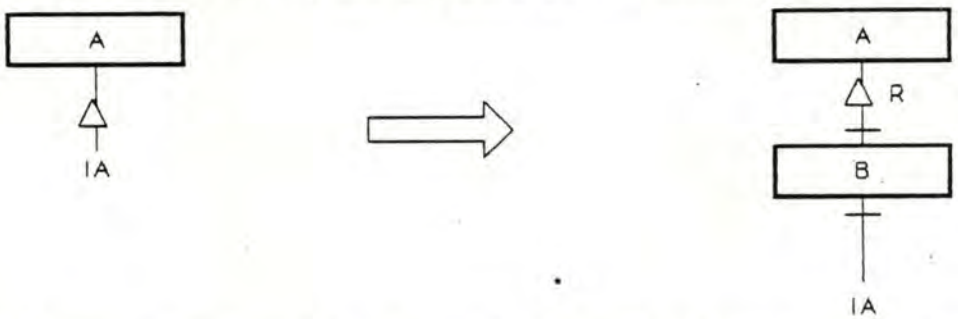


Figure 7.24: Transformation d'un ITEM répétitif en un type d'articles.

Pour la même raison que dans le point précédent, cette transformation introduit une nouvelle structure non validée par des SGDs tels que COBOL et SQL. COBOL admettant les ITEMs répétitifs, le problème de leur élimination ne se pose pas fondamentalement pour ce SGD. Par contre, SQL n'offre la possibilité que de définir des ITEMs simples. Nous proposerons dès lors une macro-transformation fusionnant les effets de la transformation d'un ITEM répétitif en un type d'articles (dans les mêmes conditions que celles données ci-dessus) avec l'élimination par duplication d'ITEMs du type de chemins créé lors de la transformation précédente. Cette macro-transformation aura notamment l'avantage d'éviter la création d'un type de chemins qui devra être éliminé par la suite.

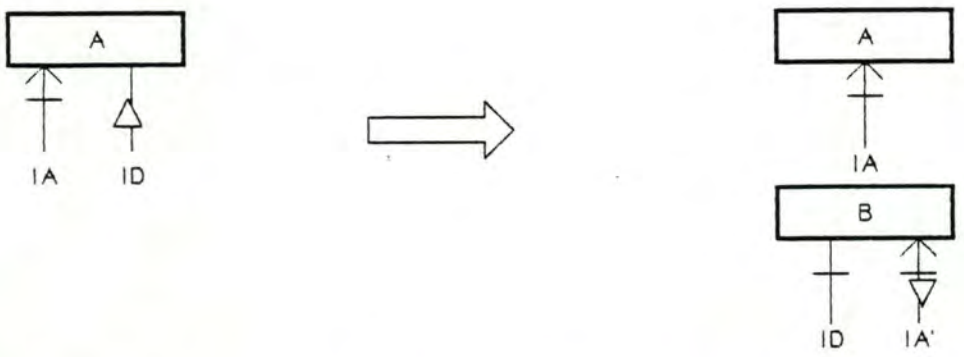


Figure 7.25: Elimination de la répétitivité sans créer un type de chemins.

LES ITEMS DE REPETITIVITE VARIABLE SANS ITEM COMPTEUR.

Les transformations proposées pour l'élimination de la répétitivité peuvent être reprises içi.

La transformation de création d'un ITEM compteur pour l'ITEM de répétitivité variable concerné étant plus directe, sera également proposée.

7.2.2.3 Contraintes sur la notion de type de chemins.:

"PAS DE TYPES DE CHEMINS"

Pour permettre l'élimination d'un type de chemins, les transformations suivantes sont proposées :

1. Transformation d'un type de chemins par duplication d'ITEMs.

2. Si le type de chemins est bidirectionnel, transformation de chaque accès du type de chemins pris séparément en remplaçant chaque sens d'accès par une clé d'accès.

Remarques.

Cette transformation n'a volontairement pas été reprise dans la liste des transformations analysées dans le paragraphe précédent. Il s'agit en effet d'une transformation essentiellement utile en pratique pour augmenter les performances d'accès.

La transformation d'un accès par type de chemins en un accès par clé peut être considérée, en termes pratiques, comme la transformation inverse à celle qui consiste à transformer un accès par clé en un accès par type de chemins analysée en 7.2.1.6. En termes théoriques elle pourra être l'application soit du principe de suppression d'un type d'articles

intermédiaire dans un type d'associations CONTAINS entre un type d'articles et un ITEM principal, soit de celui de la rotation d'un type de chemins inverse en prenant un identifiant servant de base à la rotation qui soit composé uniquement d'ITEMs. N'ayant pas recherché en profondeur les effets des techniques de transformation spécialement liées à la notion d'accès, nous n'entrons pas dans les détails.

3. Macro-transformation fusionnant les effets de la transformation du type de chemins en un type d'articles et de l'élimination des types de chemins créés par duplication d'ITEMs.

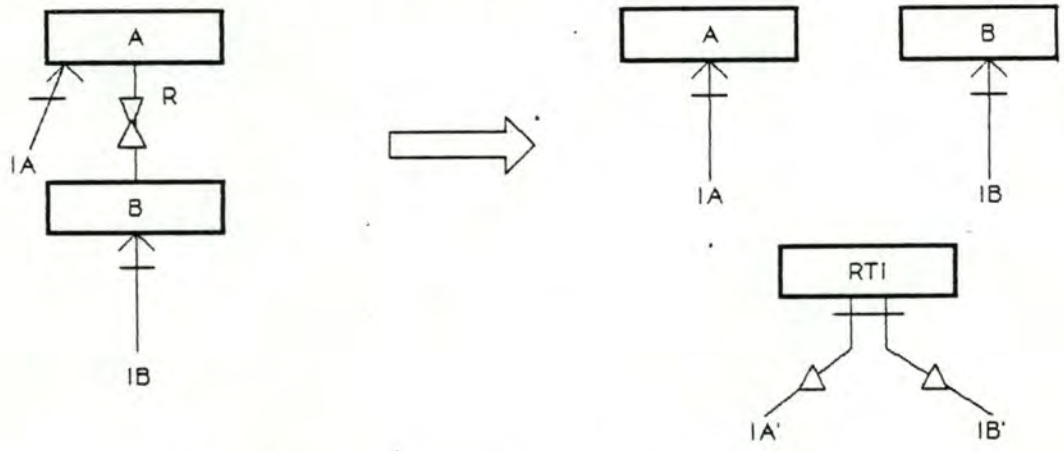


Figure 7.26: Macro-transformation : cas d'un type de chemins N-M.

Cette transformation est particulièrement intéressante si le type de chemins à éliminer a une classe fonctionnelle N-M et si la solution choisie pour la transformation du type de chemins en un type d'articles est celle qui évite la création d'un type de chemins N-M. Ces conditions sont en effet nécessaires pour éviter la création d'une nouvelle clé d'accès répétitive.

"PAS DE TYPES DE CHEMINS RECURSIFS"

Les techniques permettant d'éliminer des types de chemins récursifs seront explicitées pour l'exemple donné par la figure suivante :

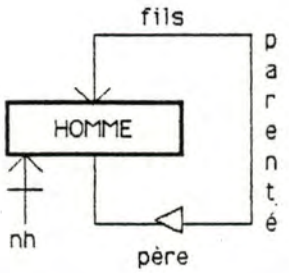


Figure 7.27: Exemple d'un type de chemins récursif.

Pour l'élimination de types de chemins récursifs, les transformations suivantes sont proposées :

1. Transformation du type de chemins récursif en un type d'articles.

Figure 7.28: Transformation en un type d'articles.

2. Elimination du type de chemins récursif par duplication d'ITEMs.

Figure 7.29: Elimination de la récursivité par duplication d'ITEMs.

3. Macro-transformation fusionnant les effets de la transformation du type de chemins en un type d'articles et de l'élimination d'un ou des deux types de chemins créés par duplication d'ITEMs.

Figure 7.30: Macro-transformation : cas d'un type de chemins récursif.

"PAS DE TYPES DE CHEMINS Plusieurs-à-Plusieurs".

Nous proposons de transformer les types de chemins de classe fonctionnelle N-M en types d'articles tout en sélectionnant la solution suivante :

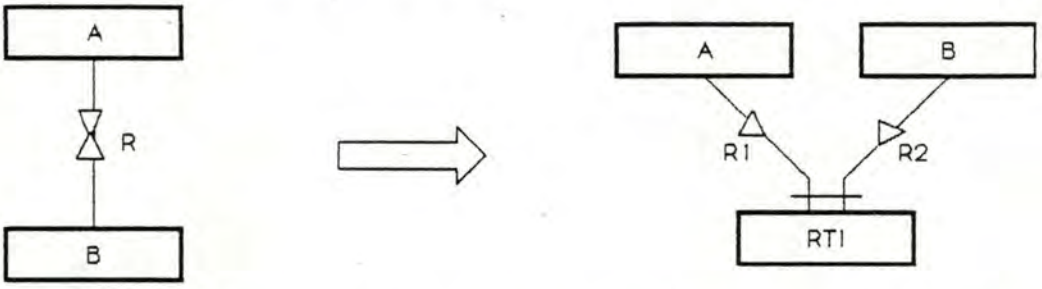


Figure 7.31: Transformation d'un type de chemins N-M en un type d'articles.

Une autre solution, qui parait cependant moins probable, serait d'éliminer ce type de chemins par duplication d'ITEMs. Dans le cas particulier de type de chemins N-M, si aucun des deux types d'articles membres n'a un identifiant formé que d'un seul ITEM, il sera nécessaire de créer un identifiant technique répondant à cette condition.

Il pourrait être intéressant de proposer également une macro-transformation qui fusionne les effets de la transformation précédente avec l'élimination des nouveaux types de chemins par duplication d'ITEMs.

Pour SQL cette solution est plus intéressante que l'élimination du type de chemins par duplication d'ITEMs puisqu'elle évite la création d'un ITEM répétitif qui devrait être éliminé par la suite.

"PAS DE TYPES DE CHEMINS Un-à-Un"

Pour éliminer un type de chemins R de classe fonctionnelle 1-1, la transformation la plus rapide et applicable sans restriction est de rendre "Plusieurs" la connectivité maximale de l'un des membres (pour CODASYL, ce membre sera de préférence celui pour lequel n'est pas définie une contrainte d'existence) et de la relier à une contrainte de cardinalité.

Si R est obligatoire pour ses deux types d'articles membres, il peut être considéré comme un type de chemins "is-a". Dans ce cas particulier, la fusion de ces deux types d'articles peut également être proposée. Remarquons que cette transformation assure l'équivalence sémantique. Elle correspond en termes théoriques à combiner une série de transferts de types d'associations entre types d'articles porteurs de la même sémantique (dont le principe est analysé dans le point 7.1.1.4.) à la transformation de suppression d'un double d'un type d'articles (analysée dans le point 7.1.1.5.).

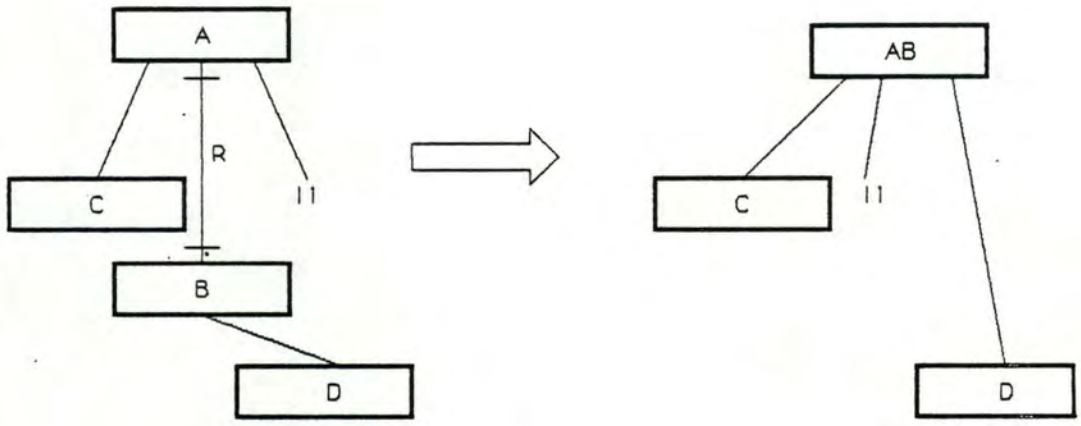


Figure 7.32: Fusion de deux types d'articles de même sémantique.

"PAS DE TYPES DE CHEMINS 1-N AYANT UNE CONTRAINTE D'EXISTENCE DU COTE N".

La solution la plus simple est de remplacer cette contrainte d'existence par une contrainte d'intégrité gérée par programmes d'application.

7.2.2.4 Contraintes sur les notions d'identifiants et de clés d'accès.

"PAS PLUS D'UN IDENTIFIANT PAR TYPE D'ARTICLES COMPOSE UNIQUEMENT D'ITEMs".

Pour éliminer un identifiant secondaire n'ayant que des composants ITEMs, il suffira d'utiliser le type d'articles SYSTEM comme support de l'identifiant. Cette transformation consistera essentiellement à augmenter ce dernier d'un composant correspondant au type de chemins qui sera créé entre le type d'articles SYSTEM et le type d'articles sur lequel est défini l'identifiant.

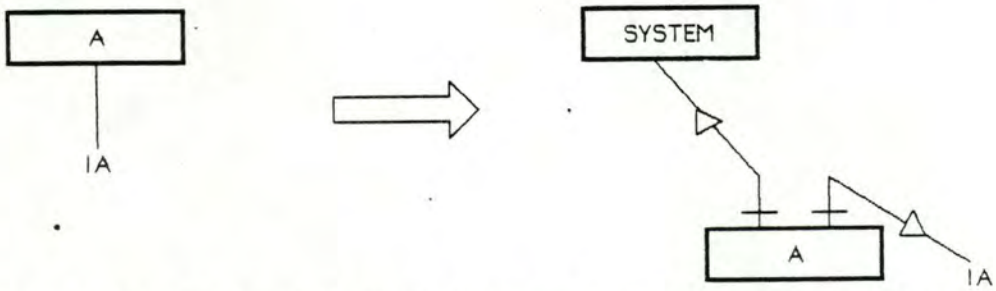


Figure 7.33: Le type d'articles SYSTEM comme support d'un identifiant.

Notons que cette transformation ne peut être proposée que pour les SGDs qui offrent un type d'articles SYSTEM. La transformation d'un ITEM en un type d'articles permet également d'éliminer des identifiants secondaires. Cette transformation a l'avantage d'être proposable pour tous les SGDs gérant la notion de type de chemins.

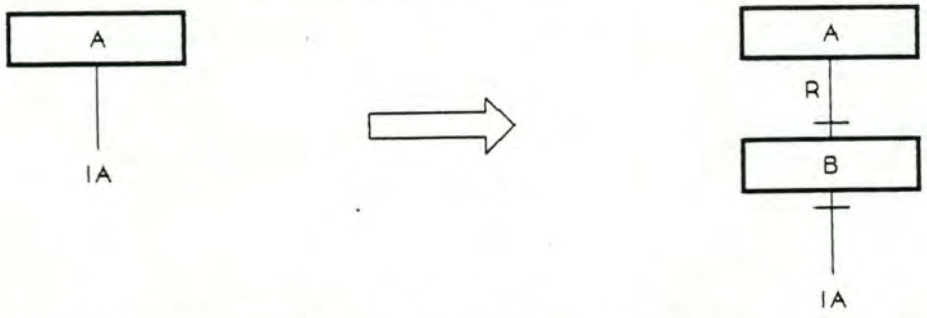


Figure 7.34: Transformation d'un ITEM identifiant en un type d'articles.

Pour pouvoir effectuer cette dernière transformation, l'identifiant ne doit être formé que d'un ITEM qui soit de plus principal. Si cette condition n'est pas satisfaite, il sera nécessaire de procéder à des regroupements/aplatissements d'ITEMs.

Pour les SGDs ignorant la notion de type de chemins, il serait particulièrement intéressant de proposer également une macro-transformation qui fusionne les effets de la transformation de l'identifiant en un type d'articles avec l'élimination par duplication d'ITEMs du type de chemins créé par la transformation précédente.

"PAS PLUS D'UNE CLE D'ACCES PAR TYPE D'ARTICLES".

Les transformations proposées pour l'élimination des identifiants secondaires sont également applicables pour éliminer les clés d'accès secondaires. A ces transformations, nous en ajouterons une troisième qui consiste à transformer uniquement la qualité de clé d'accès d'un ITEM.

1. Utilisation du type d'articles SYSTEM comme support de clé d'accès.

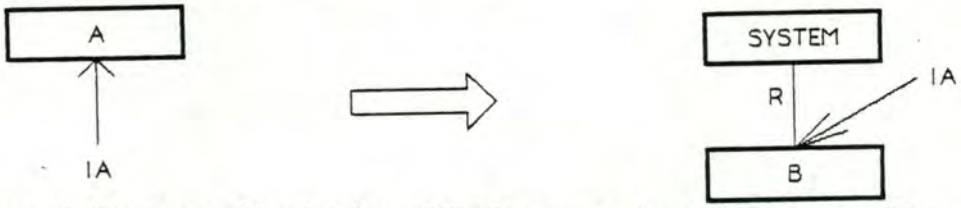


Figure 7.35: Type d'articles SYSTEM comme support de clés d'accès.

IA conservera toutes ses propriétés.

2. Transformation de l'ITEM clé d'accès en un type d'articles.

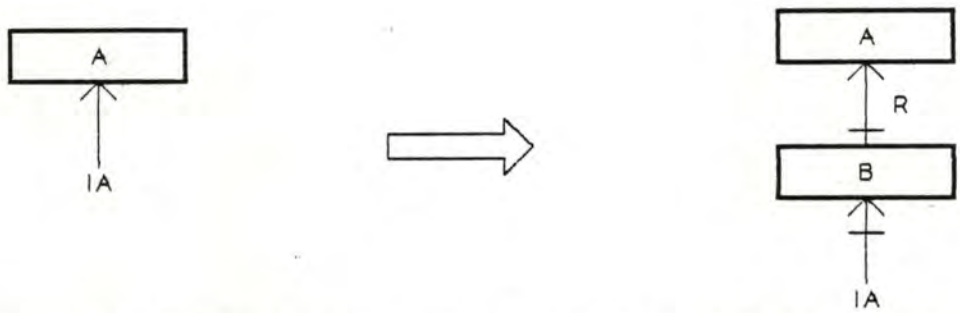


Figure 7.36: Transformation d'un ITEM clé d'accès en un type d'articles.

Toutes les solutions possibles en fonction des propriétés de IA (identifiant ou non, répétitif ou non) peuvent être proposées.

3. Transformation d'un accès par clé en un accès par type de chemins.

Nous ne reprenons ici que la technique de création d'un type d'articles intermédiaire dans un tpe d'associations inverse représenté par la qualité pour un ITEM d'être clé d'accès. Les techniques de rotation sont un peu plus délicates à gérer pour éviter la création de nouvelles clés d'accès secondaires.

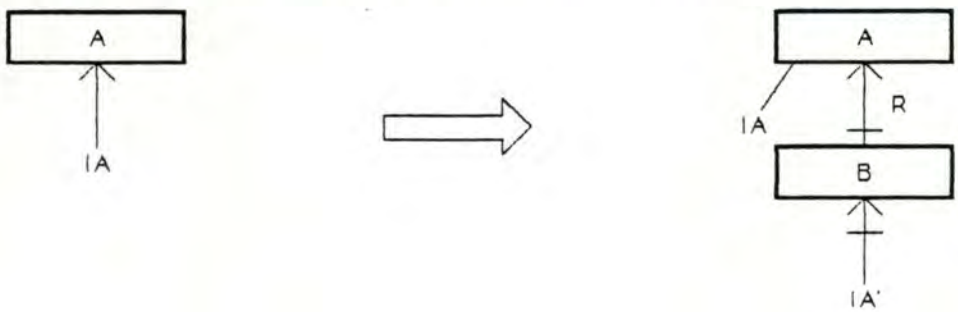


Figure 7.37: Transformation d'une clé d'accès en un type de chemins.

Comme pour la transformation donnée dans le point précédent, toutes les solutions possibles en fonction des propriétés de IA peuvent être proposées.

"PAS D'IDENTIFIANT QUI NE SOIT PAS CLE D'ACCES".

Il suffira d'ajouter à l'identifiant la qualité de clé d'accès.

Si l'un des composants de l'identifiant est déjà clé d'accès, il suffira d'ajouter à cette dernière les autres composants de l'identifiant. Cependant, l'ordre des composants ne sera pas quelconque : les nouveaux composants devront être les derniers.

En COBOL, tout identifiant/clé d'accès ne peut être formé que d'un seul ITEM. Il pourrait dès lors être intéressant de proposer également une macro-transformation permettant de rassembler les ITEMs composants de

l'identifiant/clé d'accès en un ITEM décomposable avant d'effectuer la transformation citée ci-dessus.

"PARMI LES CLES D'ACCES, AU MOINS UNE EST IDENTIFIANTE".

Une telle incompatibilité peut être éliminée simplement par la création d'un ITEM technique identifiant et clé d'accès.

Pour éviter la création d'ITEMs techniques, certains concepteurs pourraient préférer éliminer toutes les clés d'accès non identifiantes. Les transformations proposées pour l'élimination de clés d'accès secondaires sont également utiles ici. La seule différence est qu'il sera primordial d'éviter la création de nouvelles clés d'accès non identifiantes pour des types d'articles n'ayant pas de clé d'accès identifiante.

"UN IDENTIFIANT/CLE D'ACCES EST COMPOSE D'UN SEUL ITEM".

L'élimination des identifiants/clés d'accès composés de plus d'un ITEM nécessite le regroupement des ITEMs formant l'identifiant/clé d'accès en un ITEM décomposable.

"PAS DE CLE D'ACCES REPETITIVE"

L'élimination d'une clé d'accès répétitive pourra s'effectuer soit en transformant l'ITEM clé d'accès en un type d'articles, soit en transformant uniquement l'accès par clé en un accès par type de chemins. Dans les deux cas, il sera important de reporter la répétitivité de l'ITEM au niveau du nouveau type de chemins pour éviter la création d'une nouvelle clé d'accès répétitive.

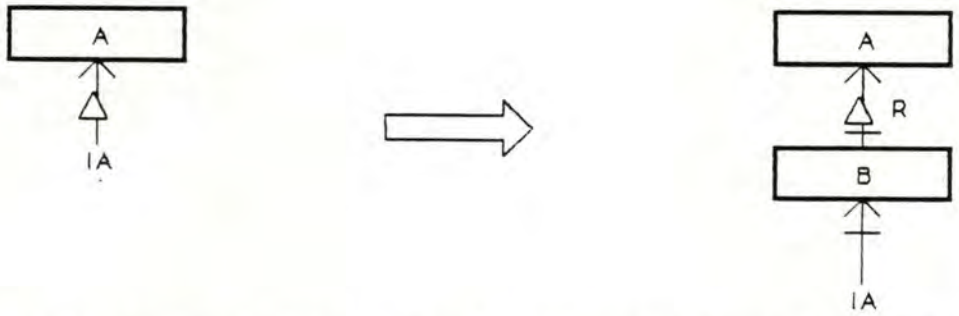


Figure 7.38: Remplacement d'une clé d'accès répétitive en types d'articles.

"UN IDENTIFIANT A AU PLUS UN COMPOSANT TYPE DE CHEMINS".

L'élimination d'un identifiant ayant plus d'un composant type de chemins peut être obtenue simplement en transférant la gestion de la contrainte d'identifiant vers les programmes d'application. Une solution

qui évite la gestion de contraintes d'identifiant par programmes d'application est d'éliminer le(s) type(s) de chemins composant l'identifiant par duplication d'ITEMs, le nouvel ITEM remplaçant le type de chemins dans l'identifiant.

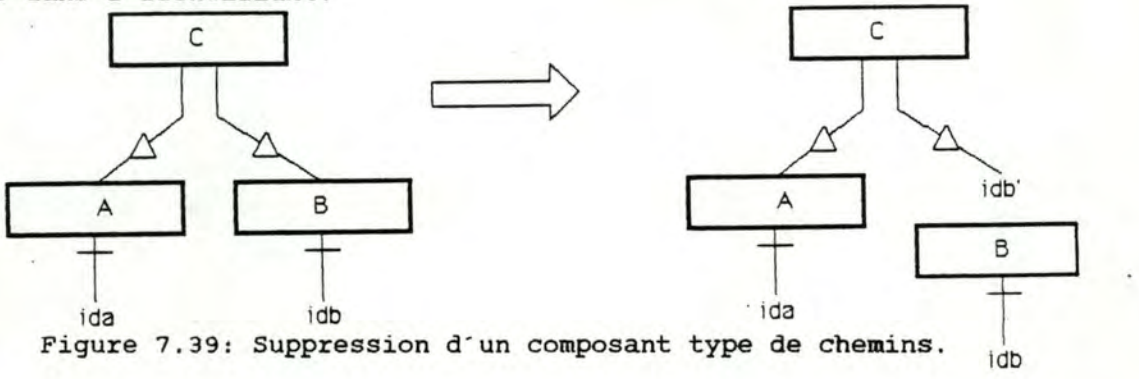


Figure 7.39: Suppression d'un composant type de chemins.

Conclusion.

L'étude de la correspondance entre les structures MAG non conformes et les transformations permettant de les éliminer montre que toutes les incompatibilités à un SGD peuvent disparaitre par application uniquement :

- de transformations appartenant à la liste qui a été proposée dans l'introduction de la deuxième section de ce chapitre (Les transformations concrètes) et,

- de simples primitives de mise à jour de la base des spécifications telles que la création/suppression d'ITEMs techniques éventuellement identifiants et/ou clés d'accès, l'utilisation du type d'articles SYSTEM comme support d'identifiants et de clés d'accès.

Toute autre transformation qui a été mise en évidence lors de l'analyse de cette correspondance peut être considérée comme une macro-transformation construite à partir des précédentes.

En faisant abstraction à la notion de conformité, toutes les transformations proposées jusqu'à présent peuvent être considérées comme transformations d'affinage de schémas de données.

7.2.3 TRANSFORMATIONS D'AFFINAGE.

Les transformations proposées jusqu'à présent sont particulièrement utiles pour éliminer les constructions MAG non validées par un SGD. Le concepteur pourra cependant désirer effectuer des transformations, non seulement dans un objectif de conformité, mais également pour répondre à des critères tels que la clarté et la modularisation des structures de données, la minimisation du temps d'accès et de mise à jour de la base des spécifications ou la minimisation de la place mémoire occupée.

Pour satisfaire un maximum des besoins du concepteur, il peut être intéressant d'étendre la liste des transformations proposées jusqu'à présent par :

- des transformations inverses pour permettre au concepteur un retour en arrière dans ses décisions.

  Comme exemples, citons :

  - la transformation d'un type d'articles en un type de chemins;
  - la transformation d'un type d'articles en un ITEM;
  - la reconstitution d'un type de chemins récursif;
  - la reconstitution d'un type de chemins éliminé par duplication d'ITEMs.

- des transformations de types de chemins inverses, éventuellement en tenant compte que les deux sens d'accès peuvent être transformés de façon différente.
  Par exemple, un sens d'accès dans un type de chemins serait transformé en un type d'articles et l'autre serait éliminé par duplication d'ITEMs.

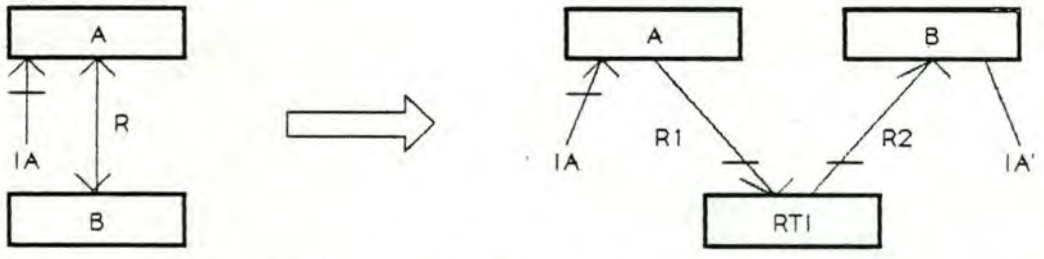


Figure 7.40: Exemple de transformation d'un type de chemins inverse.

Dans cet exemple, la transformation de l'accès de A vers B introduit un nouveau type d'articles tandis que l'accès de B vers A est éliminé par duplication de l'ITEM IA.

Remarquons que les transformations de types de chemins inverses ne sont pas indispensables pour permettre la conformité à un SGD mais ont essentiellement pour but d'augmenter les performances d'accès.

- des transformations concernant les types de chemins multitypes telles que :

  - leur éclatement (en vue d'éliminer les rôles multitypes ou simplement de les transformer) et inversément, la fusion de types de chemins ayant une extrémité commune en un type de chemins multitype (en supposant bien sûr que ces deux types de chemins sont porteurs de la même sémantique). La figure suivante illustre le principe de ces transformations.

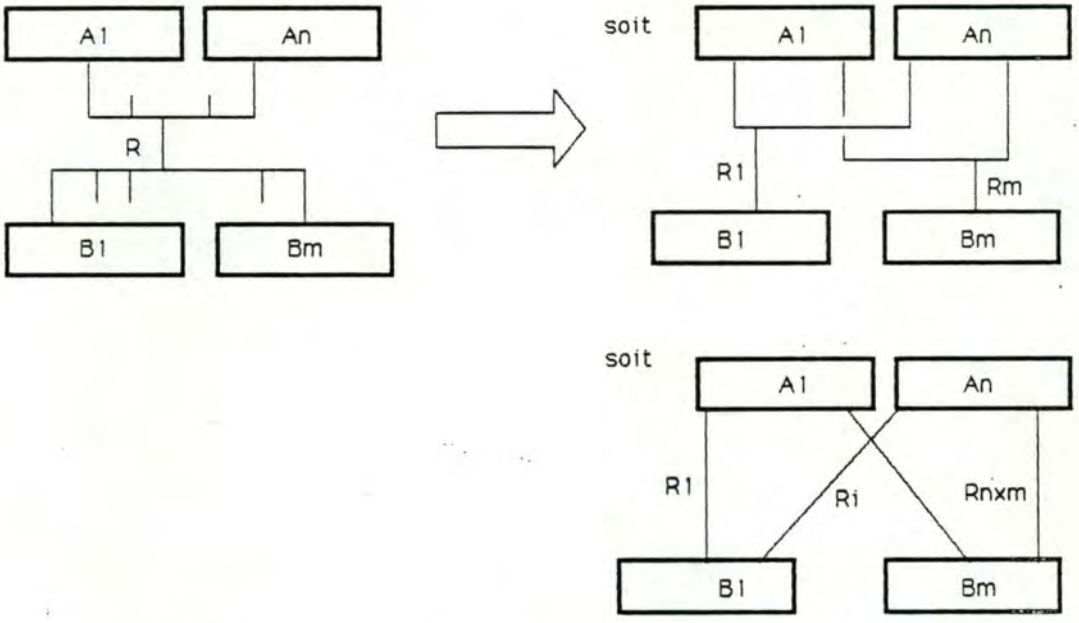


Figure 7.41: Eclatement/fusion de types de chemins.

- la création/suppression d'un type d'articles intermédiaire dans un type de chemins multitype. Cette transformation est particulièrement utile dans le cas où les deux rôles du type de chemins sont multitypes. La figure suivante illustre le principe de ces transformations.

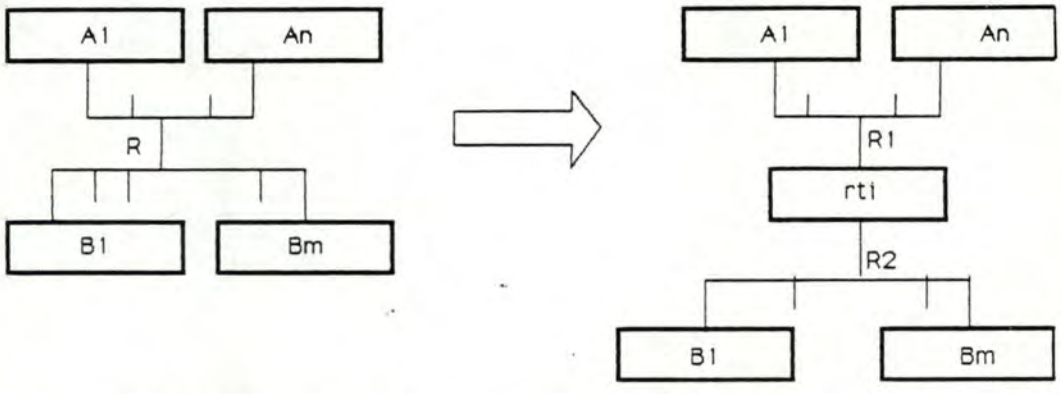


Figure 7.42: Type d'articles intermédiaire dans un multitype.

Notons que la notion de rôle multitype n'est pas reprise dans la forme canonique du modèle E/A offerte par IDA. Si nous ne permettons pas de

fusionner deux types de chemins en un multitype, le problème de leur élimination ne se pose pas.

- autres transformations telles que :

  - l'éclatement d'un type d'articles en deux types d'articles de même sémantique et inversément, la fusion de deux types d'articles de même sémantique:

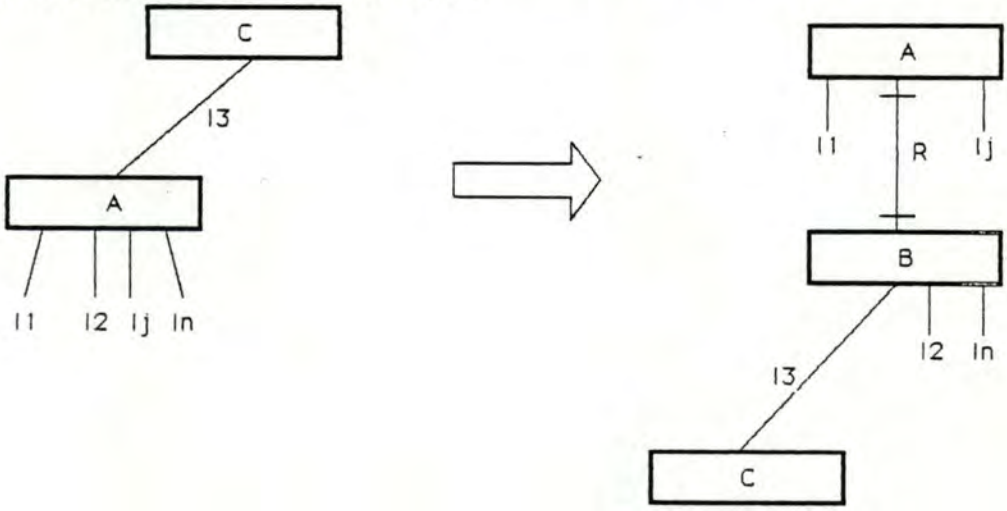


Figure 7.43: Eclatement/fusion de types d'articles.

Ces transformations permettraient par exemple de modifier la confidentialité des données, d'augmenter la performance au niveau des accès à ces données en l'occurrence en accroissant les possibilités de parallélisme.

- transfert d'un ITEM ou d'un type de chemins d'un type d'articles vers sa duplication sémantique;

- transformation d'un type de chemins par liaison de l'une de ses extrémités à l'identifiant de l'autre, cet identifiant ayant au moins un composant type de chemins (et inversément).

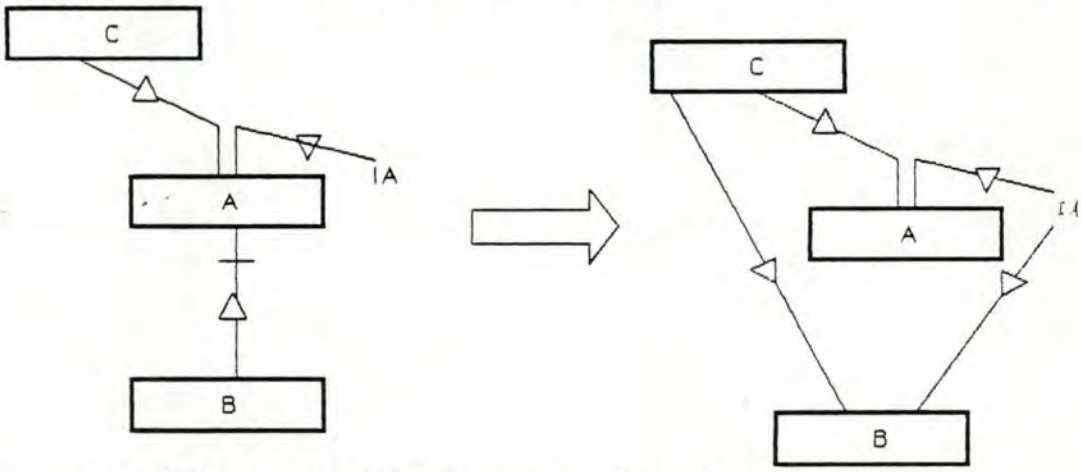


Figure 7.44: Liaison d'une extrémité à l'identifiant de l'autre.

Remarque : cette transformation est à rapprocher de l'élimination

de types de chemins par duplication d'ITEMs et s'en différencie uniquement parce que l'identifiant dont il est question aura au moins un composant type de chemins.

CONCLUSION.

L'analyse des transformations abstraites et concrètes montre que, parmi les transformations proposées, seule la transformation d'un type de chemins par duplication d'ITEMs n'assure pas toujours l'équivalence d'accès, ou si elle l'assure, ce n'est que moyennant des préconditions d'application supplémentaires. En particulier l'équivalence d'accès n'est pas assurée, soit si les ITEMs à dupliquer ne sont pas clés d'accès, soit si le type de chemins à transformer n'est accessible qu'à partir du type d'articles qui contient l'ITEM à dupliquer. La transformation sera effectuée de façon à conserver les anciens accès.

L'analyse a également permis de mettre en évidence la nécessité d'une définition plus précise du concept d'équivalence d'accès. Est-ce que conserver l'équivalence d'accès pour une transformation signifie que cette transformation se limite à offrir les mêmes possibilités d'accès ou faut-il que la transformation assure que le schéma transformé soit aussi performant du point de vue des d'accès que le schéma initial ? La transformation des clés d'accès a été effectuée en considérant la première interprétation.

La traduction des identifiants et clés d'accès qui ont d'autres composants que l'objet à transformer (ITEM ou type de chemins) n'a pas été analysée dans ce mémoire pour les transformations de types de chemins ou d'ITEMs en types d'articles, de même que pour la transformation de types de chemins par duplication d'ITEMs. Nous considérons en effet qu'il s'agit de cas que l'on rencontre rarement. Nous les avons cependant étudiées et présentons l'exposé des résultats dans les documents de travail.

L'analyse a surtout permis de mettre en évidence une grande variété de transformations applicables sur des structures MAG. Cette variété est le reflet des différentes situations envisagées. La liste des transformations abstraites pouvant ne pas être complète, il serait sans doute opportun d'entreprendre une recherche en vue de la définition d'un ensemble fini et minimal de primitives de transformations à partir desquelles on pourrait construire par composition toute transformation utile. Bien qu'il soit possible d'imaginer d'autres transformations concrètes (l'ensemble de celles-ci apparait a priori non exhaustif), il est assez agréable de constater que la plupart des constructions non conformes à un SGD peuvent être éliminées au moyen d'un ensemble assez restreint de transformations qui de plus ne semblent pas trop compliquées à réaliser.

Pour éliminer une construction MAG donnée, différentes techniques de transformation peuvent dans la plupart des cas être proposées. Les propositions peuvent dépendre des autres contraintes posées par le SGD. En effet, une transformation permettant d'éliminer une structure invalide

peut, en fonction du SGD, introduire de nouvelles incompatibilités. L'étude effectuée ci-dessus donne, pour chaque SGD en particulier, des solutions permettant de se rapprocher strictement de la solution conforme.

L'outil d'aide au processus de transformation qui sera développé laissera au concepteur la plus grande liberté de décision quant au choix et au séquencement des transformations. Il devra donc permettre de transformer un schéma même si celui-ci est déjà conforme.

Il importe que les transformations soient présentées au concepteur de façon simple, concise et intuitive. Pour une compréhension rapide des effets des diverses transformations, leur expression doit être claire et non ambigüe.

On pourrait réduire quelque peu la variété des transformations en procédant au regroupement de certaines d'entres elles. Par exemple, la transformation d'un accès par clé en un accès par type de chemins pourrait être regroupée avec la transformation d'un ITEM en un type d'articles. Il suffirait d'ajouter dans cette dernière une option de "redondance" pour l'ITEM à transformer qui ne pourra être prise que si cet ITEM est clé d'accès. La modularité maximale des transformations utiles à été volontairement décidée, d'une part pour éviter de compliquer l'analyse des transformations, et d'autre part pour permettre de mieux orienter le concepteur en fonction des effets des transformations. Cette modularité sera de plus intéressante lors de la mise en oeuvre des transformations.

La variété des transformations conduit à la nécessité de les structurer, par exemple, en fonction des constructions MAG à éliminer. Ceci permettrait de mieux orienter le concepteur vers la conformité désirée. Cette structuration dépendant essentiellement des choix de mise en oeuvre, elle sera abordée dans la troisième partie de ce travail.

Le repérage dans un schéma MAG des constructions non conformes peut être effectué, soit manuellement, soit de façon automatique au moyen d'un vérificateur de conformité. Le repérage par un support logiciel a été choisi car il soulage le concepteur de la connaissance de toutes les restrictions imposées par les SGDs sur le MAG. Ce vérificateur de conformité sera également analysé dans la troisième partie de ce mémoire.

CHAPITRE 8

TRANSFORMATION DE SCHEMAS MAG CONFORMES EN TEXTES SGD.

Cette phase consiste à coder dans le langage de description de données (DDL) d'un SGD un schéma MAG qui lui est conforme .

Le code DDL est généralement divisé en trois parties :

- le schéma logique,
- le schéma interne (physique) et
- les schémas externes.

Le schéma logique décrit les structures de données selon les concepts habituels des SGDs : il correspond à la perception d'un programmeur d'application. Sa production à partir du schéma MAG conforme est pour l'essentiel systématique.

Le schéma physique s'obtient en complétant le schéma logique par des paramètres techniques qui dépendent fortement d'un SGD à l'autre.

Les schéma externes sont obtenus à partir des sous-schémas. Leur construction nécessite donc l'extension du processus de transformation à la prise en compte de la notion de sous-schémas.

Les règles de conversion de schémas MAG conformes en textes SGD n'ont pas été étudiées dans le cadre de ce mémoire. Elles feront l'objet de recherches ultérieures.

PARTIE III

PROPOSITIONS DE MISE OEUVRE

INTRODUCTION

Dans le cadre de ce mémoire, l'option a été choisie d'orienter le support méthodologique et logiciel du processus de transformation non pas vers un outil de conception, mais plutôt vers un outil d'aide à la conception qui laissera la plus grande liberté de décision possible au concepteur. Dès lors, plutôt que de lui proposer une transformation automatique de son schéma conceptuel en un texte SGD, on séparera les trois étapes de transformation, à savoir E/A vers MAG, MAG vers MAG et MAG conforme vers DDL SGD.

L'analyse des transformations a permis de mettre en évidence le degré d'automaticité élevé des étapes "extrémales". En particulier, la première étape doit produire un schéma MAG le plus rapidement possible et en restant le plus proche possible de la solution conceptuelle. Le noyau sémantique du MAG pouvant être considéré comme un modèle de type E/A restreint aux types d'associations binaires sans attributs, le séquencement des transformations n'aura aucune influence sur le résultat. Chaque objet ne subira en effet pas plus d'une transformation. L'étape de production d'un premier schéma MAG pourra dès lors être supportée par l'outil de la manière suivante : le transformateur repère dans le schéma E/A toutes les constructions non validées par le MAG et le concepteur intervient uniquement pour déterminer les noms des nouveaux objets et la solution de transformation désirée lorsque plusieurs techniques lui sont proposées.

La prise de décision sera surtout importante pour l'étape de transformation à l'intérieur du MAG. Le but de cette étape est d'intégrer dans le schéma les choix de représentation qui auront été effectués non seulement en fonction des contraintes de conformité, mais également pour répondre à des critères d'organisation tels que la clarté de représentation, la performance d'accès et de stockage, la modularisation des structures pour augmenter les possibilités de parallélisme. Il apparait dès lors réaliste que ce soit le concepteur qui décide du séquencement des transformations puisque ce dernier, même si le concepteur se donne la conformité pour unique objectif, risque d'influencer la représentation finale des structures de données et d'accès.

Le support méthodologique et logiciel analysé dans le cadre de ce mémoire pour les transformations à l'intérieur du MAG consiste essentiellement à :

- offrir un répertoire de transformations adéquatement exprimées et structurées de façon à orienter au mieux le concepteur vers les transformations désirées;
- exécuter les transformations en fonction des choix du concepteur.

Pour que le transformateur ne soit pas dépendant de la notion de conformité, l'option a été prise de séparer le processus de transformation de celui de la vérification de conformité. Cette option a l'avantage de laisser au concepteur un maximum de liberté de transformation. Le support méthodologique et logiciel qui sera offert sera donc constitué de deux outils séparés : un premier (le transformateur) qui laisse le concepteur transformer selon ses désirs, et un second (appelé vérificateur de conformité) qui repèrera dans le schéma toutes les constructions non

conformes au SGD. Le scénario général du principe d'utilisation de ce support en vue de produire un schéma conforme peut se schématiser comme suit :

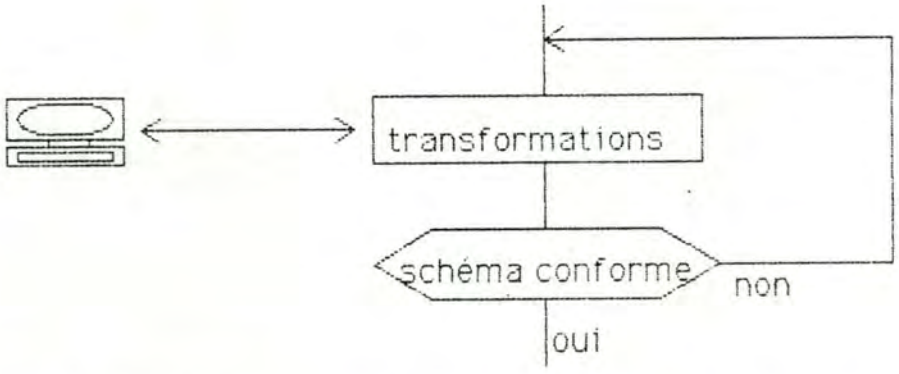


Figure 8.1: Scénario général de production d'un schéma conforme.

L'aide qui sera offerte au concepteur pour les trois étapes de transformation se traduira donc essentiellement par une assistance dans l'emploi des transformations.

Il pourrait de plus être intéressant, pour l'aider à mieux comprendre les effets spécifiques à chaque transformation proposée, de visualiser des informations telles que :

- le schéma graphique de la transformation,
- une liste de ses règles et des opérations qu'elle effectue sur le schéma,
- une liste des types de contraintes d'intégrité prises en compte pour l'avertir de la perte des contraintes d'intégrité d'autres types.

L'aspect ergonomique n'a pas été abordé dans le cadre de ce mémoire.

Une étude des valeurs par défaut pour les noms des nouveaux objets et le choix des solutions de transformation standard (lorsque la technique de transformation donne plusieurs propositions), permettra de diminuer la nécessité d'intervention du concepteur.

En ce qui concerne l'implémentation des outils, les principes à respecter sont essentiellement les suivants :

- l'extension à la fois à d'autres schémas conceptuels, à d'autres SGDs et à d'autres travaux tels qu'un processeur de conformité automatique;

- la transparence la plus grande possible par rapport aux SGDs pour minimiser le coût de modification suite à l'ajout de nouvelles contraintes ou à l'élimination de contraintes SGDs. A cet effet, on orientera vers la paramétrisation du vérificateur de conformité et vers une structuration des transformations en un répertoire unique ne faisant pas apparaître explicitement la notion de SGD;

- la modularité, en vue de faciliter et de mieux contrôler le développement et la maintenance des prorammes, et de minimiser les

modifications à apporter pour tenir compte des extensions.

La variété des transformations utiles oblige à les présenter au concepteur selon une structuration de classification. Le premier chapitre de cette troisième partie proposera quelques critères de structuration envisageables. Ayant opté pour un vérificateur de conformité paramétrable, il est nécessaire de constituer une liste paramétrable des contraintes qui peuvent être posées sur le MAG, en l'occurrence par un SGD. Une telle liste, qui inclut toutes les contraintes COBOL, CODASYL et SQL, sera exposée dans le chapitre 10. Pour la mise en oeuvre des transformations, il s'est avéré indispensable de disposer d'un schéma de l'atelier permettant une représentation unique des spécifications quel que soit leur niveau (conceptuel, logique ou physique). Un bref aperçu des types d'objets de base du schéma de l'atelier (limité à l'aspect des données) ainsi que de ses avantages est donné dans le chapitre 11 (Quelques remarques d'implémentation). Ce chapitre discutera également de deux approches de construction de schémas (approche progressive et approche par copie). La recherche pour le vérificateur de conformité d'un sens de parcours efficace de la base de spécification sera effectuée dans le dernier chapitre (chapitre 12).

CHAPITRE 9

STRUCTURATION DES TRANSFORMATIONS.

Comme nous l'avons déjà signalé, la grande variété des transformations oblige à les structurer adéquatement de façon à aider le concepteur à sélectionner celles à appliquer. Le critère de structuration sera essentiellement influencé par la guidance désirée.

Dans ce chapitre nous proposons quelques critères sur lesquels peuvent reposer la fonction de guidance et présentons pour chacun d'eux une structuration des transformations qui parait a priori la plus adéquate.

La guidance désirée pourra essentiellement être l'une des suivantes :

- guidance en fonction des types d'objets (ou concepts de base MAG) à transformer et éventuellement en fonction des propriétés et valeurs de propriétes spécifiques à chacun de ces types d'objets;
- guidance en fonction des structures à éliminer éventuellement pour atteindre la conformité à un SGD particulier;
- absence d'une guidance particulière.

Si l'on ne désire aucune guidance particulière, la structuration des transformations pourra donc être aléatoire. Il est cependant plus opportun de regrouper chaque transformation avec celle qui offre les effets inverses pour faciliter au concepteur les retours en arriére dans ses prises de décision.

Il semblerait avantageux de disposer d'un répertoire reprenant toutes les transformations qui peuvent être demandées sur les structures MAG. Un tel répertoire offrirait une grande liberté de décision au concepteur puisque ce dernier pourrait transformer son schéma même s'il est déjà conforme. Ceci est une conséquence directe du fait qu'un répertoire de transformations unique ne fait pas apparaitre explicitement la notion de SGD.

Pour un répertoire reprenant toutes les transformations, la structuration qui semble la plus adéquate permettra de répondre à un critère de guidance, d'abord en fonction du type d'objets ou concept de base à transformer, ensuite en fonction de chacune de ses propriétés. Ici aussi il est préférable de regrouper les transformations inverses même si elles ne transforment pas a priori le même type d'objets. Un exemple de transformations inverses qui ne portent pas visiblement sur le même type d'objets sont la transformation d'un type de chemins (respectivement d'un ITEM) en un type d'articles et la transformation inverse d'un type d'articles en un type de chemins (respectivement en un ITEM).

Une proposition de structure hiérarchique des transformations offrant une guidance en fonction des types d'objets et de leurs propriétés à transformer est donnée ci-dessous.

Structuration des transformations en fonction des concepts MAG.

- transformation d'ITEMs :
  - transformations générales :
    - transformation d'un ITEM en un type d'articles et inversément
    - transfert d'un ITEM d'un type d'articles vers sa duplication sémantique
  - transformations liées à la notion de décomposabilité :
    - regroupement d'ITEMs en un ITEM décomposable
    - aplatissement d'un ITEM décomposable
  - transformations liées à la notion de répétitivité :
    - transformations générales telles que la transformation d'un ITEM en un type d'articles de façon à éliminer la répétitivité
    - transformations spécifiques à la répétitivité variable illimitée telles que l'établissement d'une borne supérieure de répétitivité variable
    - création/suppression d'un ITEM compteur de répétitivité variable
  - transformations particulièrement liées à la notion d'ITEMs facultatifs
  - changement de la séquence des ITEMs
- transformation de types de chemins :
  - transformations générales telles que :
    - élimination d'un type de chemins par duplication d'ITEMs et inversément, reconstitution d'un type de chemins éliminés par duplication d'ITEMs
    - transformation d'un type de chemins en un type d'articles et inversément
  - transformations liées à la notion de récursivité :
    - soit le type de chemins à transformer est récursif,
    - soit le type de chemins obtenu après transformation sera récursif,
    - soit d'autres transformations dans lesquelles intervient un type de chemins récursif
  - transformation liées à la notion de rôle multitype
  - transformations des classes fonctionnelles
  - transformations des contraintes d'existence

- transformation de types d'articles :
  - éclatement d'un type d'articles en deux types d'articles de même sémantique et inversément
- transformation concernant la notion d'identifiant :
  - ajout/suppression de la qualité de clé d'accès à un identifiant
  - utilisation du type d'articles SYSTEM comme support d'un identifiant et inversément
  - remplacement d'un ensemble de composants ITEMs par un composant unique correspondant à leur père
  - transfert de la gestion d'une contrainte d'identifiant vers les programmes d'application
- transformation concernant la notion d'accès :
  - transformation d'un accès par clé en un accès par type de chemins et inversément
  - transformations d'un type de chemins inverse
  - utilisation du type d'articles SYSTEM comme support d'une clé d'accès et inversément
  - ajout/suppression d'un composant à une clé d'accès
  - création/suppression d'une clé d'accès
  - ajout/suppression d'un accès dans un type de chemins

La structuration en fonction des objets MAG et de leurs propriétés offre une bonne guidance et parait a postériori la plus réaliste pour un répertoire reprenant toutes les transformations possibles.

Il serait avantageux de structurer toutes les transformations utiles en un répertoire unique de façon à faire apparaitre la correspondance entre les restrictions qui peuvent être posées sur les constructions MAG et les transformations permettant d'éliminer ces constructions. Le résultat de l'étude des transformations de conformité effectuée dans le chapitre 7 pourrait alors être considéré comme le reflet d'une telle structuration. Ci-dessous, nous montrons pourquoi une telle structuration n'est pas réaliste de façon à donner un maximum de possiblités au concepteur et qu'elle nous ramène de plus à la structuration exposée précédemment.

La guidance en fonction des restrictions sur le MAG impose de structurer ces restrictions en fonction des types d'objets et des propriétés sur lesquels elles portent. Les transformations seront alors

répertoriées en fonction des types d'objets et propriétés particulières qu'elles permettent d'éliminer. Or, les transformations doivent pouvoir être utilisées pour répondre à des critéres autres que la conformité et certaines transformations n'ont pas pour effet d'éliminer une structure particulière, mais se limitent à la transformer. Par exemple, l'utilisateur pourrait vouloir transformer un ITEM répétitif en un type d'articles, non pas pour éliminer le caractère répétitif de cet ITEM, mais pour simplement augmenter la modularité des données.

Si l'on veut offrir au concepteur un maximum de possibilités de transformation, cette constatation oblige à considérer toutes les transformations dans le répertoire sans se soucier de leur objectif. Etant donné la variété des transformations, la structuration en fonction des constructions à éliminer devra être accompagnée d'une structuration en fonction des types d'objets et propriétés non pas à éliminer mais simplement à transformer. Ces deux types de structuration pourront être facilement fusionnés puisqu'ils portent sur le même critère (types d'objets et propriétés). En ne les fusionnant pas, ceci reviendrait à proposer au concepteur deux répertoires de transformations indépendants : un pour éliminer des structures non conformes et un pour effectuer des transformations d'affinage. Par la fusion de ces deux types de structuration, on en revient à la structure hiérarchique préalablement proposée.

Une guidance en fonction des contraintes sur le MAG posées par un SGD particulier peut être obtenue facilement en construisant un répertoire de transformations particulier pour ce SGD, structuré en fonction de ses contraintes particulières. Il suffit d'adapter le résultat de l'étude des transformations de conformité (deuxième section du chapitre 7) aux particularités de chaque SGD. Si seuls des répertoires particuliers à chaque SGD sont offerts au concepteur, ce dernier ne pourra pas demander l'application de transformations dans un autre but que la conformité aux SGDs pour lesquels a été constitué un répertoire.

CHAPITRE 10

STRUCTURATION DES CONTRAINTES.

La liste donnée ci-dessous expose d'une façon structurée la plupart des contraintes qui peuvent être posées par un SGD sur le MAG. Les contraintes pourront être plus ou moins restrictives par rapport à celles particulières à un SGD. Tout sous-ensemble de cette liste pourra servir de point d'entrée pour le vérificateur de conformité paramétrable qui sera analysé dans le chapitre 12. Pour permettre la conformité aux SGDs COBOL, CODASYL et SQL, la liste incluera toutes leurs contraintes particulières. Si elle semble a priori incomplète pour ces SGDs, c'est parce qu'elle exclut les contraintes moins restrictives par rapport à la définition même du MAG. Par exemple, la contrainte CODASYL qui impose que tout type d'articles aie un nom l'identifiant parmi tous les types d'articles de la BD, ne sera pas considérée puisqu'elle est moins restrictive que la règle d'unicité des noms d'objets établie par le MAG (cfr chapitre 2). Comme nous l'avons déjà signalé dans le chapitre 6 (Notion de conformité), les listes des contraintes particulières aux trois SGDs retenus sont données en annexe. Ces listes ont l'avantage de mieux cerner leurs contraintes puisqu'elles tiennent moins compte des restrictions déjà posées par la définition du MAG. Dans la liste exposée dans le présent chapitre, de même que pour celles en annexe, les contraintes sont réparties en fonction des concepts MAG sur lesquels elles portent. Les contraintes sur les notions d'identifiant, de clé d'accès et de clé d'ordre portent en général sur une combinaison de ces notions pour lesquelles il est difficile de déterminer le concept principal. On les regroupera sous le concept d'IKO ("Identifier-Key-Order"). Ces contraintes étant souvent spécifiques à chaque SGD, elles seront données séparément pour COBOL, CODASYL et SQL.

- Les types d'articles

  - Pas de type d'articles sans ITEMs (COBOL, SQL).

- Les ITEMs.

  - Pas d'ITEM facultatif (COBOL, SQL).
  - Pas d'ITEM dont le niveau de décomposition est supérieur à une valeur déterminée. Cette valeur vaut 49 pour COBOL, 99 pour CODASYL et 1 pour SQL.
  - Pas d'ITEM répétitif (SQL).
  - Pas d'ITEM de répétitivité variable sans ITEM compteur (COBOL, CODASYL).

- Pas d'ITEM de répétitivité variable ayant un ancêtre répétitif (COBOL, peut-être CODASYL).
- Pas d'ITEM de répétitivité variable qui ne soit pas le dernier dans le graphe de décomposition (COBOL).
- Le nombre d'ancêtres répétitifs d'un ITEM répétitif est limité. Ce nombre vaut 3 pour COBOL et est apparemment indéterminé pour CODASYL.
- Contraintes de format d'ITEM.

- Les types de chemins.

  - Pas de type de chemins (COBOL, SQL).
  - Pas de type de chemins de classe fonctionnelle
    - 1-1 (CODASYL)
    - N-M (CODASYL)
    - N-1 de l'origine vers la cible n'ayant pas d'inverse (CODASYL).
  - Pas de type d'associations sans accès (CODASYL).
  - Pas de type de chemins 1-N multi-origines (CODASYL).
  - Pas de type de chemins récursif (CODASYL).

- Les fichiers.

  - Pas de fichier ne contenant aucun type d'articles (COBOL).
  - Pas de type d'articles qui ne soit pas contenu dans un fichier (COBOL, CODASYL à l'exclusion du type d'articles SYSTEM).

    Remarque : en COBOL, chaque type d'articles doit de plus être contenu dans au plus un fichier.

- Les noms d'objets.

  - Contraintes de "format".
  - Le nom du type de chemins l'identifie parmi l'ensemble des types de chemins de la base de données (CODASYL).
  - Le nom de l'ITEM l'identifie parmi l'ensemble des ITEMs de la base de données (CODASYL).
  - Tout ITEM doit avoir un nom différent de celui du type d'articles dans lequel il est contenu (COBOL).

- Les IKOs.

  Les contraintes sur les IKOs étant souvent spécifiques à un SGD, il paraît difficile de les regrouper. Il n'y a d'ailleurs que deux contraintes pour les trois SGDs retenus dont l'expression est la même ou similaire : il s'agit des contraintes "pas d'identifiant qui ne soit pas clé d'accès" commune aux SGDs COBOL, et SQL et "pas de clé d'accès répétitive" posée par COBOL et CODASYL.

  Les IKOs en COBOL.

- Si une ou plusieurs clés d'accès sont définies sur un type d'articles, toute clé d'ordre définie sur ce type d'articles doit être clé d'accès.
- Les seuls ordres possibles si la clé d'ordre n'est pas clé d'accès sont l'ordre FIFO et l'ordre aléatoire et, si la clé d'ordre est clé d'accès, seul l'ordre trié par valeur croissante de la clé de tri est permis.
- Pas d'identifiant qui ne soit pas clé d'accès.
- Si une ou plusieurs clés d'accès sont définies sur un type d'articles, une au moins est identifiante.
- Une clé d'accès est constituée d'un seul ITEM.
- Pas de clé d'accès répétitive.
- Pas de clé d'accès ayant un ancêtre répétitif.
- Dans un même type d'articles, deux clés d'accès ne peuvent pas commencer à la même position.

Les IKOs en CODASYL.

- Pas plus d'une clé d'accès par type d'articles dans un fichier.
- Pas plus d'un identifiant par type d'articles dans un fichier.
- Pas de clé d'accès répétitive.
- Toute clé d'ordre est définie dans un type de chemins 1-N.
- Si un identifiant n'est pas clé d'accès, alors il est défini dans un type de chemins 1-N.
- Pas d'identifiant ayant plus d'un composant type de chemins.
- Aucun IKO de référentiel type de chemins ne peut être défini sur le type d'articles origine.

Les IKOs en SQL.

- Pas d'identifiant qui ne soit pas clé d'accès.
- Pas de clé d'ordre qui ne soit pas clé d'accès.

CHAPITRE 11

QUELQUES REMARQUES D'IMPLEMENTATION.

11.1 SCHEMA DE LA BASE DE DONNEES DE SPECIFICATION DE L'ATELIER.

Dans l'atelier logiciel orienté BDs, un schéma de la base de données des spécifications a récemment été défini, Ce schéma offre notamment une représentation unique des structures de données quel que soit le modèle utilisé (E/A, MAG, ...).

Ce schéma permet la stabilité des outils développés dans l'atelier de conception de bases de données. En effet, il constitue un compromis entre la généralité des concepts des différents modèles de représentation et la précision des définitions qui leur sont spécifiques. En ce qui concerne la généralité des concepts, un ENTITY-T pourra tout aussi bien représenter un type d'entités du modèle E/A, un type d'articles du modèle MAG ou une table SQL. En ce qui concerne la précision des définitions, la base des spécifications sera exprimée dans les termes du schéma de l'atelier en fonction du modèle utilisé. Le schéma de l'atelier sera donc accompagné. pour chaque modèle, d'un ensemble de règles de construction restreignant ainsi ses possibilités d'expression en fonction du modèle. Par exemple, si un LINK représente un type d'associations du modèle E/A, il pourra être relié à des ATTRIBUTEs, ce qui lui sera interdit s'il représente un type de chemins. La plupart des types d'objets du schéma sont également caractérisés par une série de paramètres dont l'utilité pourra être spécifique à un modèle.

Ce schéma facilitera donc la mise en oeuvre des outils de transformation de schémas. Il permettra entre autres, grâce à la représentation unique des structures de données, de passer plus rapidement de la solution conceptuelle à la solution logique et de définir les liaisons entre les différentes versions de schémas.

De plus, il acceptera toute extension ultérieure à la prise en compte des traitements puisqu'il permet d'établir les liens entre les programmes d'application et les différentes versions de schémas.

Nous ne donnerons ici qu'une partie du schéma conceptuel du sous-système de la base constitué de la description des données. Une description plus détaillée de ces concepts ainsi qu'une description du sous-système de la base constituée de la description des traitements peuvent être consultées dans (HAINAUT, 86b).

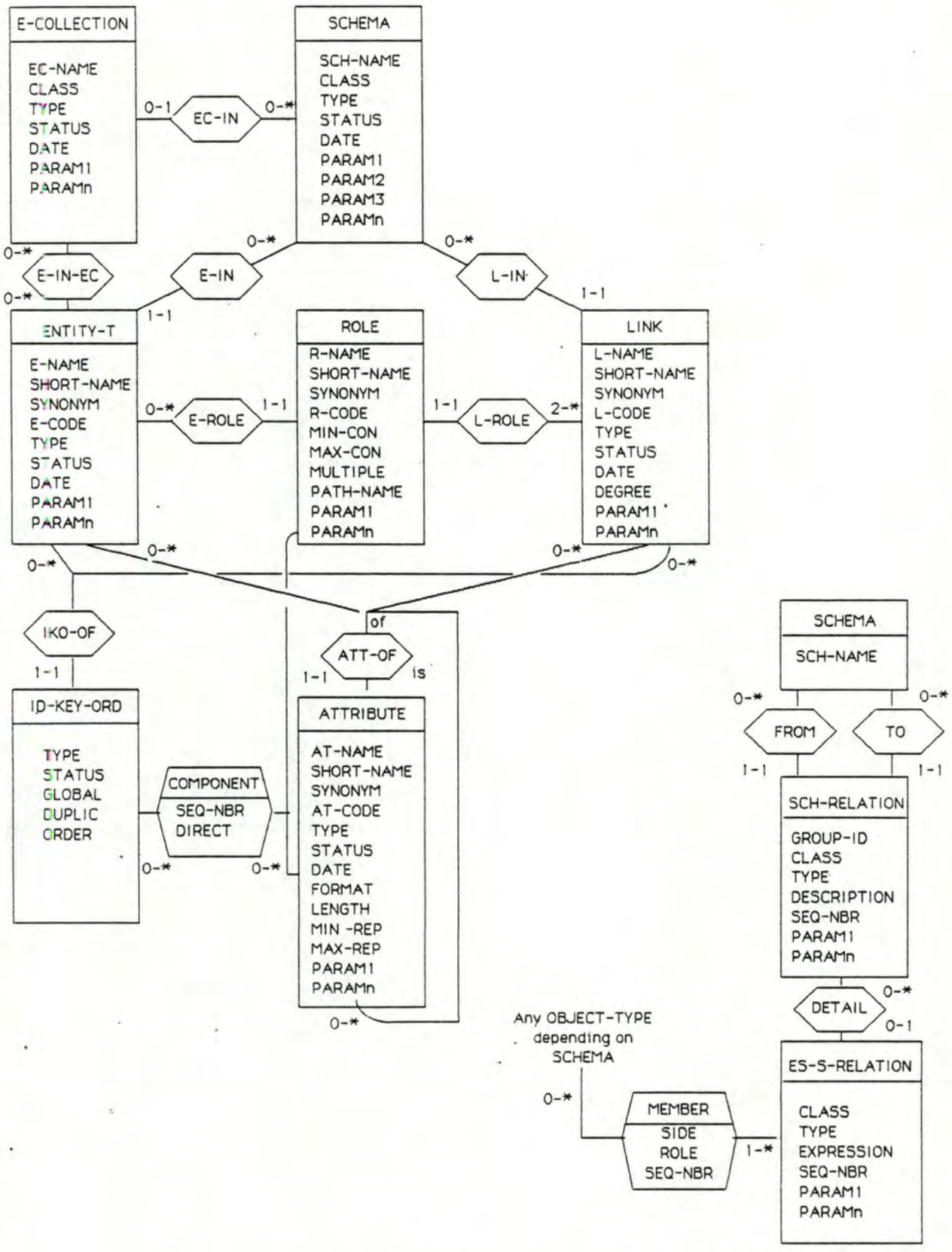


Figure 11.1: Sous-schéma de la BD des spécifications.

Le schéma de l'atelier offrant une représentation unique des structures de données quel que soit le modèle utilisé, la transformation E/A vers MAG peut être réduite à une transformation pour conformité. On pourrait dès lors penser à concevoir un vérificateur de conformité qui considère des restrictions, non pas par rapport aux constructions permises par le MAG, mais par rapport à toutes les constructions possibles dans le schéma de l'atelier. Le vérificateur de conformité pourrait ainsi repérer non seulement les constructions MAG non permises par un SGD mais également les constructions E/A non prises en compte par le MAG. Cette orientation n'a pas été prise car, contrairement au support méthodologique et logiciel fourni pour les transformations MAG vers MAG, le transformateur E/A vers MAG combine la recherche des concepts E/A non pris en compte par le MAG et leur traduction.

11.2 METHODE DE CONSTRUCITON DE SCHEMAS.

En ce qui concerne la méthode de construction de schémas, il est possible de suivre soit une approche progressive, soit une approche par copie.

Dans l'approche dite PROGRESSIVE le schéma est vide au départ - le transformateur désigne tous les objets à partir du schéma à transformer et le concepteur décide comment il veut les voir apparaître dans le nouveau schéma. Chaque session donnera lieu à une nouvelle version. Dans le schéma de l'atelier, chaque version sera reliée à la précédente par un type d'objets SCH-RELATION qui spécifiera les buts de la transformation. Les composants des deux schémas seront reliés au moyen d'associations élémentaires (EL-SCH-RELATION) qui exprimeront les règles d'équivalence. A ce niveau, la convention suivante pourra être prise : une absence de liaison entre un composant du deuxième schéma et un composant du premier exprimera que ce dernier a été intégralement recopié dans le nouveau schéma.

Dans l'approche PAR COPIE, plutôt que de construire progressivement le schéma final à partir du schéma initial, on copie ce dernier et on effectue les transformations directement sur la version copiée.

Remarquons que les relations de départ peuvent être générées automatiquement lors de la production d'un premier schéma MAG à partir du schéma conceptuel E/A. En effet, les règles permettant de passer d'un schéma conceptuel n-aire à un schéma conceptuel binaire sont complètement automatisables.

CHAPITRE 12

LE VERIFICATEUR DE CONFORMITE.

Ce chapitre présente un outil d'aide à la conception de bases de données qui permettra de repérer dans un schéma MAG toutes les constructions non conformes aux règles établies par un SGD. Cet outil est appelé vérificateur de conformité. Il permettra de soulager le concepteur, non seulement de la connaissance des règles propres à chaque SGD, mais également de la recherche dans son schéma des constructions non conformes. Pour des problèmes de taille importante, cet outil s'avère indispensable. Le concepteur, aidé de la liste des incompatibilités présentes dans son schéma qui lui sera donnée par le vérificateur de conformité, pourra ainsi facilement décider des transformations à appliquer. Un outil logiciel de vérification de conformité permet en effet de présenter la liste des constructions non conformes d'une façon plus rigoureusement structurée.

Il est possible d'envisager un vérificateur de conformité par SGD ou de n'en concevoir qu'un seul qui soit capable de repérer dans un schéma MAG toutes les constructions non validées par un jeu de contraintes quelconque. Quelle que soit l'option choisie, l'outil doit être capable de répérer dans un minimum de temps toutes les structures invalides de la base de données. Cette exigence oblige à rechercher un sens de parcours de la base qui soit efficace, c'est-à-dire au moyen duquel il sera possible de vérifier toutes les contraintes en accédant un minimun de fois à chaque objet présent dans la base.

L' optique d'un unique vérificateur de conformité a été choisie puisqu'elle lui permet d'être indépendant d'un SGD particulier. Le concepteur pourra ainsi demander la conformité à un jeu de contraintes quelconque, même si celui-ci est plus ou moins restrictif par rapport à celui d'un SGD réel. L'extension de l'outil à la prise en compte d'un autre SGD ne nécessitera pas l'étude d'un nouveau vérificateur mais une adaptation de l'unique vérificateur en ne considérant pour ce SGD que les contraintes qui ne sont pas encore prises en compte.

Ayant opté pour la paramétrisation du vérificateur, le sens de parcours a été conçu à partir de la liste fusionnant les contraintes qui peuvent être posées sur le MAG qui a été donnée dans le chapitre 10 (Structuration des contraintes). Dans le présent chapitre, nous recherchons un sens de parcours pour le vérificateur de conformité qui soit efficace quel que soit le jeu de contraintes donné en entrée. Nous donnerons également une idée de la façon dont les contraintes peuvent être efficacement vérifiées. Le lecteur intéressé par une étude plus algorithmique est reporté en annexe. L'efficacité relative de ce vérificateur paramétrable par rapport à un vérificateur particulier à chaque SGD ne sera pas prouvée formellement puisque celle-ci dépend non seulement du sens de parcours permettant de vérifier toutes les contraintes

mais également de la mise en oeuvre que nous n'avons pas eu le temps d'aborder.

SENS DE PARCOURS DU VERIFICATEUR DE CONFORMITE.

Afin d'évaluer plus facilement le sens de parcours du vérificateur, on supposera dans un premier temps que la base de données ne sera pas copiée par partie en mémoire centrale.

Il est possible de concevoir différents types de sens de parcours. Dans un souci essentiellement pédagogique, nous présentons le sens de parcours de façon évolutive.

PREMIERE IDEE.

Vérifier chaque contrainte séparément , c'est-à-dire trouver un sens de parcours indépendant pour chaque contrainte.

Par exemple, supposons la vérification des trois contraintes suivantes :

- pas d'ITEM décomposable;
- pas d'ITEM répétitif;
- pas d'ITEM facultatif.

Cette vérification nécessite trois accès à chaque ITEM alors qu'un seul suffirait si ces contraintes étaient vérifiées en un seul parcours de la base.

DEUXIEME IDEE.

Parcourir tous les objets de la base dans un ordre quelconque, et dès l'accès à un objet, vérifier toutes les contraintes le concernant.

Pour l'exemple ci-dessus, un seul accès par ITEM suffit.

L'idée, bien que bonne, est incomplète. En effet, si toutes les contraintes étaient locales(a) à chaque objet, ce sens de parcours serait le meilleur puisque leur vérification nécessiterait au maximum un accès par objet de la base. Malheureusement, certaines contraintes portent sur des objets devant être considérés non pas séparément, mais en relation avec d'autres objets de même type ou de type différent.

(a) On dira qu'une contrainte est locale à un objet si sa vérification ne nécessite pas d'autres accès qu'à l'objet considéré.

Exemples de telles contraintes :

- pas plus de x niveaux de décomposition d'ITEMs;
- pas d'ITEMs répétitifs ayant un ancêtre répétitif.

En appliquant à la lettre la seconde idée, on risque d'accéder plusieurs fois à certains objets.

La nécessité de prise en compte des contraintes non locales oblige à mieux structurer le sens de parcours du vérificateur. Cette structuration sera guidée essentiellement par les contraintes non locales à partir des conditions de structuration qu'elles posent sur le sens de parcours pour une vérification efficace. Ci-dessous nous analysons ces conditions et recherchons pour chaque contrainte une technique permettant de la vérifier efficacement.

CONTRAINTE SUR LES TYPES D'ARTICLES.

La contrainte de présence d'ITEMs dans un type d'articles oblige à emboîter le parcours des ITEMs dans celui du type d'articles pour lequel ils sont définis.

CONTRAINTES SUR LES ITEMS.

Certaines contraintes sur les ITEMs imposent un parcours dynastique (du père vers les fils) dans le graphe de décomposition des ITEMs.

De telles contraintes portent par exemple sur :

- le nombre maximal de niveaux de décomposition;
- le nombre maximal d'ancêtres répétitifs d'un ITEM répétitif;
- l'emplacement des ITEMs dans le graphe de décomposition telle que la contrainte COBOL spécifiant qu'un ITEM de répétitivité variable doit être le dernier dans le graphe de décomposition.

Ce sens de parcours est le plus efficace puisqu'il permet de vérifier presque toutes les contraintes portant sur la notion d'ITEM au moyen d'un seul accès par ITEM.

Seules les contraintes concernant la notion d'ITEM compteur de répétitivité variable posent problème. L'efficacité de leur vérification ne dépend cependant nullement du sens de parcours adopté mais uniquement de leur représentation choisie dans le schéma de l'atelier. Si une liaison compteur est représentée par une simple relation de type COUNTER entre ITEMs, la vérification de ces contraintes nécessitera au maximum un accès supplémentaire par ITEM de répétitivité variable ne pouvant pas être relié à un ITEM compteur (par exemple en COBOL et CODASYL). Par contre, si elle est représentée par une contrainte d'intégrité, il sera nécessaire pour tout ITEM de répétitivité variable ne pouvant être relié à un ITEM compteur, de parcourir en moyenne la moitié des contraintes d'intégrité qui lui sont reliées.

Etant donné le sens de parcours dynastique choisi, la vérification des contraintes sur les ITEMs pourra s'effectuer de la manière suivante :

- contrainte sur le nombre de niveaux de décomposition : nécessite la gestion d'un compteur de niveaux de décomposition;
- pas d'ITEM facultatif : contrainte locale;
- contrainte de format : contrainte locale;
- contraintes sur la notion de répétitivité :
  - "pas d'ITEM répétitif" : contrainte locale;
  - "pas d'ITEM de répétitivité variable sans ITEM compteur" : contrainte vérifiable soit par un test de la cardinalité de cet ITEM dans la relation de type COUNTER, soit par accès aux contraintes d'intégrité reliées à cet ITEM;
  - "pas d'ITEM de répétitivité variable ayant un ancêtre répétitif" : nécessite la gestion d'un compteur de niveaux de répétitivité;
  - "pas d'ITEM de répétitivité variable qui ne soit pas le dernier dans le graphe de décomposition" : il suffit de vérifier que l'ITEM précédant n'était pas de répétitivité variable;
  - "pas d'ITEM répétitif ayant un certain nombre de pères répétitifs" : vérification au moyen du compteur de niveaux de répétitivité (le même que pour tester si un ITEM de répétitivité variable a ou non un ancêtre répétitif).

CONTRAINTES SUR LES TYPES DE CHEMINS.

La contrainte d'absence de type de chemins récursifs (CODASYL) oblige pour sa vérification à accéder aux types d'articles membres des types de chemins. Cette constatation amène à l'idée que l'emboitement dans un unique parcours de la base à la fois des contraintes sur la notion de type de chemins et celles sur la notion de type d'articles pourrait être plus efficace qu'une séparation en deux parcours indépendants de la vérification de ces deux classes de contraintes. Un tel emboitement de façon à n'accéder qu'une seule fois à tout objet de ces deux types est cependant impossible. Il est dès lors nécessaire d'effectuer une étude comparative de l'efficacité des solutions envisageables. Pour cette comparaison, nous supposerons que la base de données ne contient pas de type de chemins multitypes, ces derniers n'ayant aucune influence sur le résultat final de la comparaison.

Première solution : effectuer la vérification des contraintes portant sur la notion de type de chemins au moyen d'un sens de parcours indépendant de celui qui est nécessaire à la vérification des contraintes portant sur la notion de type d'articles.

En adoptant ce sens de parcours, le nombre d'accès nécessaires à la vérification des contraintes sur les types d'articles et les types de chemins serait d'un accès par type d'articles et de trois accès par type de chemins.

Deuxième solution : emboiter le parcours des types d'articles dans celui des types de chemins.

Cette solution est moins efficace qu'elle ne paraît a priori. En effet, chaque type d'articles n'étant pas nécessairement membre d'un type de chemins, la vérification des contraintes sur les types d'articles qui n'interviennent dans aucun type de chemins nécessite de reparcourir tous les types d'articles.

Si cette solution est choisie, la vérification des contraintes sur les types d'articles et les types de chemins nécessite trois accès par type de chemins plus un accès par type d'articles (même résultat que pour la solution précédente).

Troisième solution : emboiter le parcours des types de chemins dans celui des types d'articles.

En adoptant ce sens de parcours, il est possible de réduire le nombre d'accès pour vérifier les contraintes sur les notions de type de chemins et de type d'articles à un accès par type d'articles plus deux accès par type de chemins.

Il suffit de disposer d'une table à une entrée par type de chemins et de procéder en deux étapes de la manière suivante :

```
Pour chaque type d'articles rt
     vérifier les contraintes sur les notions de type
     d'articles
     pour chaque type de chemins pt dont rt est membre
          mémoriser dans l'entrée de la table correspondant
          à pt toutes les informations sur le rôle de rt dans
          pt (nom du rôle, nom de rt, sa connectivité ...)

Pour chaque entrée dans la table
     vérifier les contraintes sur le type de chemins
     correspondant.
```

Remarque : le nom du type de chemins l'identifie non pas dans la base de données, mais parmi tous les types de chemins (autre que son inverse) ayant mêmes membres. Pour permettre d'identifier une entrée dans la table sans devoir accéder à tous les types d'articles membres lors de chaque parcours d'un type de chemins, il est indispensable d'insérer dans la table les codes identifiants internes des types de chemins (cfr. schéma de l'atelier).

Choix d'une solution. la troisième solution, bien que nécessitant la manipulation d'une table, a été choisie en raison de son efficacité.

Toutes les contraintes concernant les types de chemins pourront être vérifiées au moyen des informations dans la table une fois le parcours des types d'articles terminé.

CONTRAINTES SUR LES FICHIERS.

La vérification de la contrainte sur le nombre de fichier(s) dans le(s)quel(s) un type d'articles peut être contenu, doit être intégrée dans le parcours des types d'articles.

La contrainte "pas de fichier ne contenant aucun type d'articles" exige par contre un parcours de tous les fichiers. Ce parcours peut être considéré comme séparé du parcours des types d'articles.

CONTRAINTES SUR LES NOMS D'OBJETS.

Les contraintes sur la formation des noms d'objets se divisent en deux catégories :

- les contraintes de "format" de noms et
- les contraintes d'unicité de noms.

Les contraintes de "format" de noms.

Les contraintes de "format" sont des contraintes locales à chaque objet qui ne nécessitent aucun accès supplémentaire par rapport au parcours actuel. Ce dernier prévoit en effet l'accès à tous les objets sur lesquels une telle contrainte peut être posée.

Pour les ITEMs et types d'articles, aucun problème ne se pose quant à l'endroit dans le parcours où insérer leur vérification. Ces objets ne sont en effet parcourus qu'à un seul endroit. Pour les fichiers, il suffit d'intégrer les contraintes de format dans le parcours de tous les fichiers de la base. Pour les types de chemins, la vérification de ces contraintes pourra être effectuée à partir de la table contenant les informations sur les types de chemins dès que celle-ci est complète.

Les contraintes d'unicité de noms.

Les contraintes d'unicité de noms d'objets sont assez spécifiques à chaque SGD. Elles présentent cependant une similitude qu'il est intéressant d'exploiter.

La contrainte CODASYL concernant l'unicité des noms d'ITEMs dans la BD nécessite la gestion d'une table destinée à contenir les noms des ITEMs de la BD. Dès l'accès à un ITEM, il suffit d'aller tester si son nom est déjà ou non repris dans la table. Dans l'affirmative, il y aura violation de la contrainte; sinon ce nouveau nom sera ajouté dans la table.

La contrainte d'unicité des noms des types de chemins peut être vérifiée d'une manière similaire en utilisant soit la table des types de chemins présentée préalablement, soit une table qui ne contiendra que les noms des types de chemins. Dans le premier cas, les règles d'insertion dans le sens de parcours de la vérification de cette contrainte, sont les mêmes que celles pour la vérification des contraintes de format. Dans le second cas, ne pouvant pas déterminer, comme pour les ITEMs, un moment unique pour les opérations de test et de mise à jour de la table, il sera nécessaire d'utiliser des indicateurs qui précisent pour chaque type de chemins si sa contrainte d'unicité a déjà ou non été vérifiée.

La contrainte d'unicité des noms restant à vérifier pour COBOL est la suivante : "tout ITEM doit avoir un nom différent de celui du type d'articles dans lequel il est contenu". La simplicité de cette contrainte et l'emboitement du parcours des ITEMs dans le parcours de leur type d'articles rendent sa vérification à la fois aisée et efficace. Dans un soucis d'uniformisation de la méthode de vérification des contraintes d'unicité des noms, la contrainte COBOL sera cependant vérifiée en utilisant le même procédé que celui qui a été proposé pour CODASYL. Pour le cas COBOL, la table contiendra uniquement le nom du type d'articles pour lequel les ITEMs vérifiés sont définis.

CONTRAINTES SUR LES IDENTIFIANTS, CLES D'ACCES ET D'ORDRE.

Comme nous l'avons déjà signalé, les notions d'identifiant, de clé d'accès et de clé d'ordre sont étroitement liées. Pour cette raison il a été décidé de les représenter dans le schéma de l'atelier au moyen d'un même type d'objets appelé IKO. Ces notions doivent donc également être regroupées pour la recherche d'une vérification efficace des contraintes portant sur elles. Le TYPE de l'IKO indique précisément son rôle : identifiant et/ou clé d'accès et/ou clé d'ordre(a).

La fait de ne pas tenir compte des IKOs globaux (IKOs définis sur plusieurs types d'articles) facilite davantage la vérification.

Tous les IKOs étant définis sur un type d'articles, toutes les contraintes portant sur la notion d'IKO pourront être vérifiées en emboitant leur parcours dans celui du type d'articles pour lequel ils sont définis.

Des contraintes telles que celles sur le nombre d'IKOs par type d'articles et celle spécifiant que "si une ou plusieurs clés d'accès sont définies pour un même type d'articles, une ou moins est identifiante" exigent de plus cet emboitement.

Il se pourrait également qu'il soit plus efficace, pour les contraintes portant plus particulièrement sur le nombre et le type des composants des IKOs, d'intégrer leur vérification dans le parcours de ces

---

(a) Le lecteur intéressé trouvera dans les documents de travail une analyse détaillée des concepts d'identifiant, de clé d'accès et de clé d'ordre, ainsi qu'une étude de toutes les restrictions posées sur la notion d'IKO (CHAR-MUL,85c).

composants pour éviter de les reparcourir. L'efficacité de cet emboitement peut cependant dépendre non seulement de chaque contrainte en particulier mais également d'informations statistiques telles que le nombre moyen de composants par IKO, éventuellement en distinguant les composants ITEMs des composants types de chemins. Nous n'entrerons pas dans les détails ici.

Ci-dessous un procédé de vérification qui apparait a priori efficace est donné pour chaque contrainte sur la notion d'IKO.

Les contraintes "pas de clé d'ordre qui ne soit pas clé d'accès" et "pas d'identifiant qui ne soit pas clé d'accès" peuvent être vérifiées facilement en testant le type de l'IKO dès qu'on y accède.

Il en est presque de même pour la contrainte : "les seuls ordres possibles si la clé d'ordre n'est pas clé d'accès sont l'ordre FIFO et l'ordre aléatoire et si la clé d'ordre est clé d'accès, seul l'ordre trié par valeur croissante de la clé de tri est permis". Pour sa vérification il suffit de tester les lois d'ordre des IKOs clés d'ordre.

La contrainte "si une ou plusieurs clés d'accès est définie sur un type d'articles, une au moins est identifiante" nécessite la gestion d'un compteur de clés d'accès lors du parcours des IKOs définis sur un type d'articles et un indicateur de présence pour un type d'articles d'une clé d'accès identifiante.

Un compteur du nombre de composants ITEMs d'une clé d'accès permet de vérifier la contrainte "une clé d'accès est constitué d'un seul ITEM".

La contrainte "pas d'identifiant ayant plus d'un composant type de chemins" peut être vérifiée de la même façon.

La contrainte "pas plus d'une clé d'accès par type d'articles dans un fichier" ne peut être vérifiée efficacement qu'à l'aide d'une table reprenant les correspondances entre les IKOs de référentiel fichier et les fichiers qu'ils ont pour référentiel. Lors du parcours des IKOs définis sur un type d'articles dès qu'on rencontre un identifiant dont le référentiel est un fichier, on teste si le fichier a déjà ou non une entrée dans la table. Dans l'affirmative, il y a une structure invalide; sinon on crée une nouvelle entrée correspondante à ce fichier.

La vérification de la contrainte "pas plus d'un identifiant par type d'articles dans un fichier" peut s'effectuer de la même manière.

Pour vérifier que "toute clé d'ordre est définie dans un type de chemins 1-N", un accès est nécessaire à l'éventuel composant type de chemins de toute clé d'ordre.

Pour vérifier la contrainte "pas d'identifiant dont le référentiel n'est pas type de chemins qui ne soit pas clé d'accès", un accès à chaque composant type de chemins des identifiants non clé d'accès est nécessaire

Un test du caractère répétitif de chaque ITEM composant d'une clé d'accès est nécessaire pour vérifier la contrainte "pas de clé d'accès répétitive".

La contrainte "pas de clé d'accès ayant un ancêtre répétitif" peut être vérifiée selon trois procédés différentes :

- soit en testant pour chaque ITEM lors du parcours du graphe de décomposition des ITEMs s'il est ou non composant d'une clé d'accès. Dans l'affirmative, aucun de ses ancêtres ne pouvait être répétitif.
- soit en remontant le graphe de décomposition dès qu'on accède à un composant ITEM d'une clé d'accès tout en testant le caractère répétitif de chacun de ses ancêtres.
- soit en faisant précéder le parcours des ITEMS du parcours des IKOs définis sur un type d'articles et en sauvant lors du parcours des IKOs tous les ITEMs composants d'une clé d'accès. Ainsi, le niveau de répétitivité de chaque ITEM figurant dans la table des clés d'accès peut être testé lors du parcours des ITEMs.

La troisième solution, bien que nécessitant la manipulation d'une table, a été choisie en raison de son efficacité.

Des techniques analogues peuvent être proposées pour la vérification de la contrainte "dans un même type d'articles, deux clés d'accès ne peuvent pas commencer à la même position".

Pour vérifier la contrainte "aucun IKO de référentiel type de chemins ne peut être défini sur le type d'articles origine", il suffit d'emboiter dans le parcours des types de chemins (qui est emboité dans le parcours des types d'articles) un parcours de tous les IKOs dont le type de chemins est composant et de tester si le type d'articles membre pour lequel est défini l'IKO est origine ou cible dans le type de chemins.

La contrainte "si une ou plusieurs clés d'accès sont définies sur un type d'articles, toute clé d'ordre définie sur ce type d'articles doit être clé d'accès" exige la gestion d'un compteur de clés d'ordre pour chaque type d'articles lors d'un parcours de tous les IKOs. Si un ou plusieurs de ces IKOs sont clés d'accès, alors toutes les clés d'ordre qui ne sont pas clés d'accès sont invalides.

CONCLUSION.

Le vérificateur de conformité analysé ci-dessus est dit paramétrable en ce sens qu'il ne fait pas apparaitre explicitement la notion de SGD mais qu'il est utilisable pour un jeu de contraintes quelconque, à condition bien sûr que celui-ci soit inclus dans l'ensemble des contraintes prises en compte. Il est important de noter que la recherche des structures MAG non compatibles aux SGDs COBOL, CODASYL et SQL pourra être effectuée par ce vérificateur de conformité puisqu'il considère toutes les contraintes de ces SGDs. Pour la mise en oeuvre de l'outil logiciel, il sera nécessaire de décider d'une technique de paramétrage du sens de parcours en fonction des contraintes et d'une expression paramétrable des contraintes. Pour permettre à l'outil de vérifier la conformité à un SGD particulier, on pourra par exemple lui associer une liste de contraintes, structurée de la même façon que la liste servant de référence présentée au chapitre 10. Ce procédé pourra par exemple être mis en oeuvre au moyen de fichiers. Dans

le fichier reprenant les contraintes particulières à un SGD, une ligne vide pourra représenter une absence de contrainte. Les lignes non vides pourront être accompagnées de valeurs pour les paramètres des contraintes correspondantes. Un autre procédé de paramétrisation applicable pourrait être de construire une table à double entrée permettant d'adapter chaque contrainte paramétrable aux besoins des SGDs en particulier.

Pour aider le concepteur à choisir la technique de transformation à appliquer, il serait intéressant que chaque construction invalide repérée par le vérificateur soit accompagnée de suggestions concernant les techniques de transformation qui permettent de l'éliminer. Cette extension serait facile à réaliser à partir des résultats de l'étude effectuée dans la section "Transformations de conformité" du chapitre 7.

Une autre extension possible serait d'orienter le vérificateur de conformité non plus vers un outil d'aide séparé du transformateur mais vers un processeur de correction automatique de schémas non conformes en fonction d'un jeu de contraintes donné. Le vérificateur deviendrait alors un processeur de conformité. Une telle orientation n'a cependant pas été choisie parce que d'une part, elle ne répond pas à l'objectif premier de laisser au concepteur une liberté maximale dans le choix des transformations - et d'autre part, un tel processeur semble a priori assez complexe à concevoir, surtout si l'on désire qu'il soit paramétrable. En effet, une transformation utilisée pour éliminer une structure non conforme donnée peut, en fonction du SGD amener de nouvelles incompatibilités. Dans un tel cas, il serait nécessaire de rechercher un sens de parcours des contraintes et des objets de la base qui prévoit des retours en arrière sur des objets et éventuellement des contraintes déjà analysés.

CONCLUSION.

Arrivées au terme de ce mémoire, nous tentons une brève évaluation de ce qui a été réalisé et des difficultés rencontrées. Nous présenterons ensuite les extensions qui nous paraissent indispensables, puis nous clôturerons en établissant une comparaison de notre travail avec celui réalisé dans le cadre de l'élaboration de bases de données relationnelles.

Rappellons que l'objectif de ce mémoire consistait à analyser et développer un support méthodologique et logiciel qui assistera la production d'une solution COBOL, CODASYL et SQL au départ d'un schéma E/A en passant par une solution logique exprimée dans le **MAG**.

La figure suivante expose en bref l'organisation de ce support :

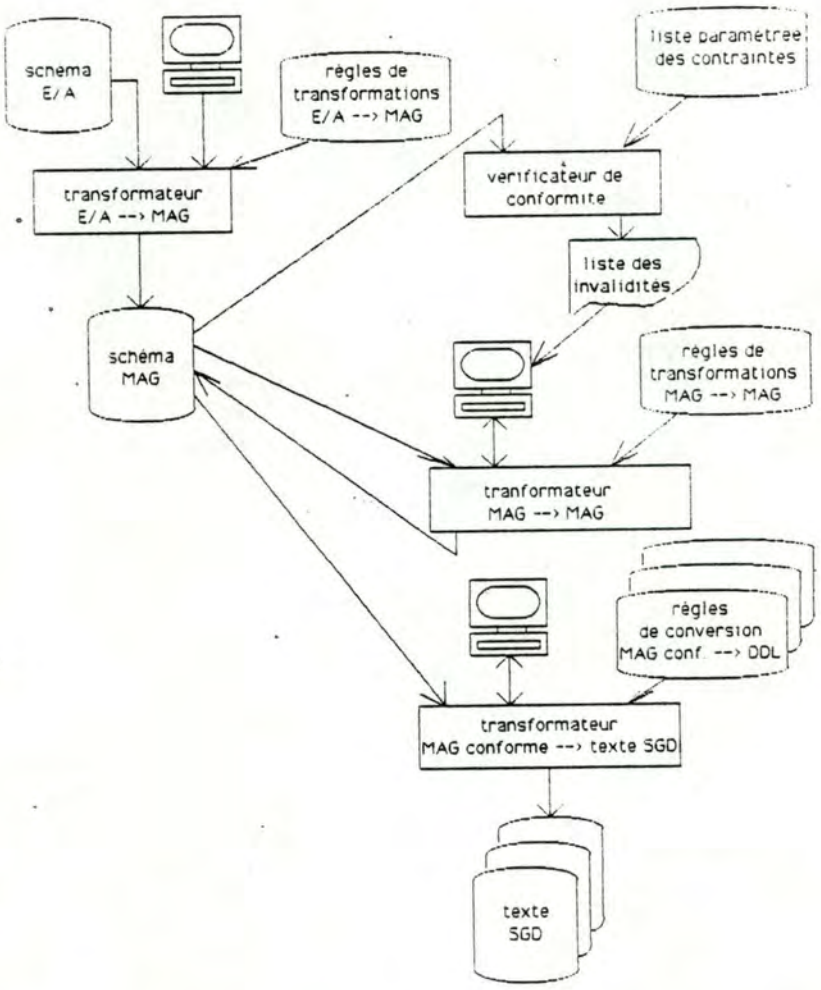


Figure : Scénario d'utilisation du transformateur de schémas.

Evaluation du travail effectué.

Dans le cadre de ce mémoire, nous avons approfondi d'une part, la recherche et l'analyse des transformations E/A vers MAG et MAG vers MAG et d'autre part, l'expression des restrictions réalistes d'un SGD par rapport au MAG. Une classification de ces restrictions et un sens de parcours efficace pour un vérificateur de conformité paramétrable ont également été conçus.

Les aspects étudiés des transformations sont essentiellement : la recherche des informations qui doivent ou qui peuvent être données par le concepteur, l'étude des préconditions d'application pour qu'elles assurent l'équivalence, l'expression de leur règles de transformation et les objectifs qui peuvent leur être assignés.

Pour les transformations E/A vers MAG la seule restriction par rapport aux constructions prises en charge par IDA est que l'on suppose que les types d'associations sont minimaux.

Les transformations à l'intérieur du MAG ont d'abord été étudiées dans une approche théorique modularisée en fonction des concepts du MAG (les notions d'accès et d'identifiant, les différentes sortes de répétitivité, les types de chemins récursifs, inverses et multirôles, ...). Cette approche a surtout permis de mettre en évidence la variété des transformations de base assurant l'équivalence sémantique et d'accès et la multiplicité des constructions MAG envisageables, en l'occurrence, les diverses combinaisons possibles des concepts d'identifiant, de clé d'accès et d'ordre.

L'ampleur de cette étude nous a forcées à élaguer les cas ne présentant pas une utilité pratique particulière pour l'étape d'analyse préparatoire à la mise en oeuvre.

Outre la variété apparamment exhaustive des transformations et des cas particuliers d'application, les problèmes rencontrés proviennent essentiellement de :

- notre manque de pratique dans le domaine des bases de données qui nous a amenées à vouloir considérer tous les cas théoriques possibles;
- la difficulté d'une part, d'établir des règles de transformation dans les termes abstraits de types d'objets et d'associations et d'autre part, de se persuader de la préservation de l'équivalence sémantique et d'accès par les transformations;
- la définition quelque peu imprécise du concept de clé d'accès et de la propriété pour un item d'être obligatoire ou facultatif;
- la combinaison des concepts d'identifiant, de clé d'accès et d'ordre par la plupart des SGDs;
- le vie active de l'atelier logiciel de conception de bases de données et d'IDA.

Quelques propositions d'extensions.

Outre la traduction en une solution exécutable par les SGDs retenus d'un schéma MAG conforme, un éventuel regroupement des transformations similaires pour alléger le répertoire offert et la prise en compte de l'aspect ergonomique lors de la mise en oeuvre des outils, nous proposons les extensions suivantes du travail réalisé :

- complèter l'analyse des valeurs par défaut des noms des nouveaux objets et choisir la solution de transformation standard (lorsque la technique offre différentes possibilités), pour permettre au concepteur de léguer ses prises de décision au support logiciel;

- examiner les possibilités d'une part, de définir un noyau de transformations élémentaires fini et minimal à partir duquel pourra être définie toute transformation utile et d'autre part, de réaliser un processeur de conformité automatique (éventuellement paramétrable);

- prendre en compte d'une part, la non linéarité de la démarche pour éviter de reparcourir toute la chaîne suite à une mise à jour du schéma conceptuel et d'autre part, la désagrégation des types d'associations non minimaux, soit au travers d'une étape de transformation qui précèderait la production d'un premier schéma MAG, soit en étendant l'ensemble des règles de transformation E/A vers MAG;

- réaliser un outil d'évaluation à la fois du processeur de transformation et du vérificateur de conformité;

- intégrer dans les règles de transformations relatives à la notion d'accès la gestion des traffics dans la base de données et établir les règles d'équivalence entre les différentes versions de schémas pour faciliter la prise en compte ultérieure des traitements;
  ...

Comparaison de l'étude avec celle effectuée par J. De CRANE et A. GONAY.

Le mémoire de J. DE CRANE et A. GONAY (DEC-GON,86) consiste en l'analyse et développement d'un outil d'aide spécifique au processus de transformation d'un schéma E/A en un texte exécutable par la classe des SGBDs relationnels. La démarche qui soustend cet outil se différencie donc de celle retenue dans le cadre de ce mémoire qui est applicable quel que soit le SGD cible.

Les types d'associations étant éliminés avant d'aborder la phase de conception logique (au moyen d'une technique qui peut être considérée comme une combinaison de celles analysées ci-dessus), l'ensemble des transformations nécessaires est fortement réduit.

Notons que ce support fait plus appel au savoir-faire et au bon sens du concepteur et ne se soucie primordialement pas de la préservation de l'équivalence sémantique par les transformations proposées. Pour les transformations offertes de spécialisation/généralisation, il semble de plus que cette équivalence ne puisse être assurée par aucun outil logiciel.

Ce dernier ne pourrait qu'aider le concepteur à préserver l'équivalence.

Lorsque ce dernier a décidé de la représentation finale des structures de données, la définition des accès est effectuée presqu'automatiquement en considérant que toute relation devra être munie d'une clé d'accès (index) qui sera l'un de ses identifiants. Les SGBDs relationnels ne permettant de définir des identifiants qu'au travers d'index, l'étape de définition des accès proposée est indispensable pour permettre la prise en charge des identifiants par le SGBD. Elle devrait cependant être suivie de la définition des accès nécessaires à une exécution efficace des programmes d'application. L'intégration des mécanismes d'accès dans un schéma de données proche de la solution conceptuelle semble intéressante pour permettre une spécification des structures d'accès indépendante des choix d'implémentation.

Signalons que J. DE CRANE et A. GONAY ont analysé et développé des primitives d'optimisation qui calculent, pour le SGD ADR, la taille prise en volume par les données et clés d'accès. En utilisant ces primitives, le concepteur peut plus facilement minimiser la place mémoire occupée par ses structures.

BIBLIOGRAPHIE.

(ANT-LIV,85), ANTONELLIS, LIVA, "DATAID-1 : a database design methodology", Information Systems, Vol. 10. No. 2, pp 181-195, 1985.

(BENCI), BENCI, BODART, BOGAERT, CABANES, "Concepts for the design of a conceptual schema", pp 181-200.

(BER-GAT,86), BERTINCHAMPS, GATELLIER, "Sstème pour la conception d'un schéma conceptuel d'informations", Institut d'Informatique, Mémoire 1985-86.

(BOD-PIGN,83), BODART, PIGNEUR, "Conception assistée des applications informatiques, 1.Etude d'opportunité et analyse conceptuelle", MASSON, 1983.

(BOU,85), BOUZEGHOUB, GARDARIN, METAIS, "SECSI : Un outil interactif de modélisation conceptuelle de bases de données", Modèles et Bases de Données, Octobre 1985, pp 13-22.

(BRAEGGER,85), BRAEGGER, DUDLER, REBSAMEN, ZEHNDER, " Gambit : An Interactive Database Design Tool for Data Structures, Integrity Constraints, and Transactions", IEEE Transaction of Software Engineering, Vol. SE-11, NO 7. Juillet 1985.

(BRES,85), BRES, " Une intégration d'un module d'analyse organique et d'un dictionnaire de données physiques entre les outils logiciels IDA DELTA", Méthodes et technologie des systèmes d'informations, Novembre 1985.

(CAD-MUL,85), CADELLI, MULLER, "Contribution to a database design workbench ADL-COBOL/GAM Compiler", Institut d'Informatique, Mémoire 1984-85.

(CERI,83), CERI, "Methodology and tools for database design", North-Holland, Publish 1983.

(CHAR-MUL,85a), CHARLOT, MULLER, "Transformations de schémas : 1. E/A --> MAG (partie 1)", Document de travail, Genève, Octobre 1985.

(CHAR-MUL,85b), CHARLOT, MULLER, "Transformations de schémas : 1. E/A --> MAG (partie 2)", Document de travail, Genève, Octobre 1985.

(CHAR-MUL,85c), CHARLOT, MULLER, "Etude des valeurs possibles des

propriétés d'IKO", Document de travail, Genève, Novembre 1985.

(CHAR-MUL,85d), CHARLOT, MULLER, "Transformations de schémas : 2. MAG --> MAG", Document de travail, Genève, Décembre 1985.

(CHAR-MUL,85e), CHARLOT, MULLER, "Transformations de schémas : schémas SAP --> SAN", Document de travail, Décembre 1985.

(CHAR-MUL,85f), CHARLOT, MULLER, "Transformations de schémas : transformations élémentaires", Document de travail, Genève, Décembre 1985.

(CHEN,76), CHEN, "The Entity-Relationship Model - toward a unified view of data", ACM TODS, 1976, Vol 1,1.

(CLARINVAL,81), CLARINVAL, "Comprendre, connaitre et maitriser le COBOL, normes ANSI COBOL 1974", Presses Universitaires de Namur, 1981.

(DATE,81), DATE, "An introduction to database systems", ADDISON WESLEY, 3rd edition, 1981.

(DATE,83), DATE, "An introduction to database systems", Vol 2, ADDISON WESLEY, 1983.

(DEC-GON,86), DE CRANE, GONAY, "Elaboration d'une base de données ADR à partir d'un schéma E/A", Institut d'Informatique, Mémoire 1985-86.

(HAINAUT,81a), HAINAUT, "Theoretical and practical tools for data base design", 1981 V.D.L.B. Conf. Proceedings, Septembre 1981, ACM/IEEE pp 216-224.

(HAINAUT,81b), HAINAUT, "Un modèle descriptif de bases de données au niveau organique : le modèle d'accès", Notes de cours, Institut d'Informatique, Janvier 1981.

(HAINAUT,83), HAINAUT, "Cadre de référence pour la conception de bases de données", Institut d'Informatique, Décembre 1983.

(HAINAUT,84), HAINAUT, "Introduction à un système de gestion de bases de données relationnel SQL/DS d'IBM", Institut d'Informatique, Janvier 1984.

(HAINAUT,85), HAINAUT, "Introduction aux systèmes de gestion de bases de

données CODASYL 71", Institut d'Informatique, Octobre 1985.

(HAINAUT,86a), HAINAUT, "Conception assistée des applications informatiques, 2. Conception de la base de données", MASSON, 1986.

(HAINAUT,86b), HAINAUT, " Projet Atelier Bases de Données - IDA II : Schémas de la Base de Spécifications", Institut d'Informatique, Janvier, 1986.

(OLLE,78), OLLE, "The Codasyl Approach to Data Base Management", Wiley, 1978.

(REINER), REINER, BRODIE, BROWN, FRIEDELL, KRAMLICH, LEHMAN, ROSENTHAL, "The Database Design and Evaluation Workbench (DDEW) Project at CCA", Computer Corporation of America, Four Cambridge Center.

(SMITH,77), SMITH, SMITH, "Database abstraction : aggregation and generalization", ACM TODS, 2, 2,Juin 1977, pp 105-133.

(TEOREY,85), TEOREY, YANG, FRY, "Relational Database Design using the ER Model - A Pratical Methodology", University of Michigan, Ann Arbor, MI 48109-1109, Octobre 1985.

ANNEE ACADEMIQUE 1985-1986

FACULTÉS
UNIVERSITAIRES
N. D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION A L'ATELIER LOGICIEL DE CONCEPTION DE BASES DE DONNEES : ETUDE DE TRANSFORMATIONS DE SCHEMAS

ANNEXES

PROMOTEUR : J-L. HAINAUT

Mémoire présenté par
Charlot C. et Müller I.
en vue de l'obtention du titre
de Licencié et Maître en
Informatique

RUE GRANDGNAGE, 21, B - 5000 NAMUR (BELGIUM)

ANNEXE 1

CONTRAINTES DE CONFORMITE

1 LISTE DES CONTRAINTES PAR SGD.

1.1 SQL

Les RECORD-TYPEs.

- pas de record-type sans items

Les ITEMs

- pas d'item décomposable
- pas d'item répétitif
- pas d'item facultatif
- le format : - pas de booléen
  - pas de chaines de caractères de plus de 32768 caractères

Les PATH-TYPEs.

- pas de path-type

Les noms d'objets : ITEM, RECORD-TYPE, INDEX.

- pas de minuscule, pas de point, ...
- unicité des noms :

  RECORD-TYPE : le nom du record-type l'identifie parmi l'ensemble des record-types de la BD

  ITEM : le nom de l'item l'identifie parmi l'ensemble des items du même record-type

Les identifiants, clés d'accès et clés d'ordre.

- pas d'identifiant qui ne soit pas clé d'accès
- pas de clé d'ordre qui ne soit pas clé d'accès

  Remarque : contrairement à la philosophie des SGBD purement relationnels, SQL n'exige pas que tout record-type aie un identifiant.

1.2 COBOL

Les RECORD-TYPEs.

- pas de record-type sans items

Les ITEMs.

- pas d'item facultatif
- pas plus de 49 niveaux de décomposition
- format : - pas plus de 18 positions numériques
  - le type booléen n'existe pas explicitement en COBOL, mais il sera cependant possible, lors de la conversion du schéma dans le DDL COBOL de traduire les items booléens par des conditions au moyen de la clause 88.
- répétitivité : - pas plus de trois niveaux de répétivité
  - la borne inférieure de la répétitivité ne peut pas être nulle, c'est-à-dire pas d'items facultatif
  - pas de répétitivité variable sans item compteur

    Remarque : apparamment, il n'existe aucune utilité COBOL à ce que l'item compteur puisse ne pas se trouver dans le même record-type que l'item compté. Mais, ne pouvant pas encore actuellement peser le pour et le contre de la présence d'une telle contrainte au niveau MAG sur la conception des familles de transformation, nous ne l'imposerons pas tout de suite au niveau MAG mais la spécifierons au niveau des SGDs qui la posent (dans le cas présent, CODASYL).

  - pas d'item de répétitivité variable ayant un ancêtre répétitif et par conséquence, l'item compteur ne peut pas être répétitif
  - pas d'item de répétitivité variable qui ne soit pas le dernier dans le graphe de décomposition des items du record-type
  - pas d'item compteur de type non numérique

Remarque : cette contrainte disparaît dans le cas où elle est déja posée au niveau du MAG.

Les PATH-TYPEs.

- pas de path-type

Les FICHIERS.

- pas de record-type qui ne soit pas contenu dans un et un seul fichier

- pas de fichier ne contenant aucun record-type

Les noms d'objets : PICHIER, ITEM, RECORD-TYPE.

- aucun nom d'objet ne peut :
  - être un mot réservé
  - être un nom-système
  - être formé d'autres caractères que :
    - les chiffres ou
    - les lettres ou
    - le trait d'union
  - ne contenir aucune lettre
  - avoir un trait d'union en première ou dernière position

- unicité des noms :

Les règles d'attribution des noms données à la page 87 du CLARINVAL sont les suivantes :

A. soit deux sommets distincts X et Y du graphe de structure; l'ensemble complet des noms déclarés sur le chemin allant de la racine au sommet X inclusivement ne peut être égal à aucun sous-ensemble des noms déclarés sur le chemin allant de la racine au sommet Y inclusivement; en particulier,

B. tous les noms déclarés sur un même chemin du graphe de structure doivent être différents;

C. tous les sommets immédiatement subordonnés à un même sommet doivent se voir attribuer des noms différents.

Ces règles, n'étant pas tout à fait correctes, ne peuvent pas être vérifiées telles quelles. En effet, la règle A n'a de sens que si les deux sommets X et Y ne sont pas

ancêtres l'un de l'autre et par conséquent, la règle B a été incorrectement considérée comme cas particulier de la règle A.

Ces règles seront dès lors reformulées de façon, à la fois à séparer les règles d'unicité de noms concernant les fichiers, les record-types et les items et, à minimiser le coût de leur vérification (par exemple, pour deux items non ancêtres l'un de l'autre, il suffira de remonter jusqu'à l'ancêtre commun et non pas jusqu'à la racine qui, selon la règle A, devait être le fichier ou la base de données).

FICHIER : le nom du fichier l'identifie parmi l'ensemble des fichiers de la base (de type fichier ou non)

RECORD-TYPE : le nom du record-type l'identifie parmi l'ensemble des record-type contenus dans le même fichier.

ITEM :

Contrainte 1.

tout couple d'items contenus dans un même record-type doit être tel que :

si ces deux items sont ancêtres l'un de l'autre ils doivent avoir des noms différents

sinon, l'ensemble complet des noms déclarés sur le chemin allant de l'ancêtre commun vers X inclusivement ne peut pas être un sous-ensemble de l'ensemble complet des noms déclarés sur le chemin allant de l'ancêtre commun vers Y inclusivement.

Contrainte 2.

tout item doit avoir un nom différent de celui du record-type dans lequel il est contenu.

Les identifiants, clés d'accès et clés d'ordre.

Remarque:
La notion d'ordre n'est pas représentée explicitement en COBOL mais est implicitée par le type d'organisation du fichier puisque l'utilisateur a pu expliciter les ordres désirés dans les termes du MAG, il est nécessaire d'en fixer les contraintes d'ordre qui devront être respectées dans l'atelier.

- si une clé d'accès est définie sur un record-type, toute clé d'ordre définie sur ce record-type doit être clé d'accès

- les seuls ordres qui auront un sens COBOL sont :

  si la clé d'ordre n'est pas une clé d'accès :

  - l'ordre FIFO (fichiers séquentiels) et
  - l'ordre désordre (fichiers relatifs);

  si la clé d'ordre est une clé d'accès :

  uniquement l'ordre trié par valeur croissante de la clé de tri

- pas d'identifiant qui ne soit pas clé d'accès

- si une ou plusieurs clés d'accès sont définies sur un record-type, une au moins doit être identifiante

- pas de clé d'accès composée de plusieurs items

- pas de clé d'accès ayant un ancêtre ou étant lui-même répétitif

- si un fichier contient plusieurs record-types, toute clé d'accès décrite dans un de ces record-types doit exister dans tous les autres record-types du fichier et y occuper le même emplacement par rapport à la première position du record-type

- si un item est clé d'accès, aucun de ses fils de numéro de séquence égal à 1 ne peut être clé d'accès, c'est-à-dire dans un même record-type, deux clés différentes ne peuvent pas commencer à la même position

1.3 CODASYL

Les ITEMs.

- pas d'item facultatif (sauf répétitivité 0 exprimée sous forme de répétivité variable. Le compteur qui devra obligatoirement accompagner l'item prendra la valeur 0 en cas d'absence de valeurs.)

- pas plus de 99 niveaux de décomposition

- répétitivité : - pas de répétitivité variable sans item compteur

  - pas d'item compteur qui ne soit pas contenus dans le même record-type que les l'items comptés

  - pas d'item compteur décomposable

    Remarque : cette contrainte disparaît dans le cas où elle est déja posée au niveau du MAG.

  - pas d'item compteur de type non numérique

    Remarque : cette contrainte disparaît dans le cas où elle est déjà posée au niveau du MAG.

  - pas d'items compteur facultatif, répétitif : question à poser au niveau MAG

- format d'un item : tous les formats d'items valides au niveau du MAG sontrepris en CODASYL.

Les PATH-TYPEs.

- pas de path-type :

  - sans accès

    Remarque : pas de contrainte d'absence de path-types sans inverse. En effet, ceux-ci pourront être représentés au moyen de la notion de "DYNAMIC SETS".

  - 1-1, N-N ou N-1 sans inverse, les connectivités étant considérées du

record-type ORIGIN vers le(s) record-type(s) TARGET.

- multiorigines

- dans lequel le record-type ORIGIN est aussi TARGET (pas de cycle élémentaires formés d'un seul path-type et donc pas non plus de path-type récursif)

- ayant le record-type SYSTEM comme TARGET

Remarque :

le record-type SYSTEM, existant implicitement dans tout schéma CODASYL, pourra être défini explicitement au niveau de l'atelier. Cependant, certaines contraintes devront être respectées par ce record-type spécial :
- il ne devra contenir aucun item
- il ne pourra pas être contenu dans un fichier utilisateur
- il ne pourra être TARGET d'aucun path-type
- aucun IKO ne pourra être défini pour lui

et éventuellement d'autres.

Remarque concernant les cycles non éléméntaires.

Si l'on obtient un cycle par composition de plusieurs path-types (c'est-à-dire que, en parcourant ces path-types dans le sens ORIGIN --> TARGET, il est possible de retrouver le point de départ), et si tous les TARGET de tous ces path-types sont déclarés comme membres obligatoires, il sera nécessaire, lors de la conversion DDL, de permettre la violation d'au moins une de ces contraintes d'existence.

Les FICHIERs.

- pas de record-type (autre que le SYSTEM) qi ne soit contenu dans aucun fichier.

Les noms d'objets : ITEM, RECORD-TYPE, FICHIER.

- unicité des noms :

pas d'objets ayant le même nom qu'un autre objet de même type dans la BD

Les identifiants, clés d'accès et clés d'ordre.

Clés d'accès.

pas plus d'une clé d'accès par record-type dans un fichier

Remarque : le procédé de vérification de cette contrainte dépendra de la prise de décision de pouvoir ou non définir qui au niveau du MAG un IKO pour un sous-ensemble des fichiers contenant le record-type sur lequel porte cet IKO.

Identifiants.

pas plus d'un identifiant par record-type dans un fichier

Remarque : cfr clés d'accès.

Clés d'ordre.

- un path-type ne peut pas être le support de plusieurs clés d'ordre (sauf dans le cas d'une clé d'ordre globale aux record-type TARGET du même path-type puisqu'une telle clé d'ordre est représentée au moyen de plusieurs IKO, de l'ordre de un par record-type)

- pas de clés d'ordre dont le référentiel n'est pas path-type

Combinaisons identifiants - clés d'accès - clés d'ordre

- pas d'identifiant dont le référentiel n'est pas path-type qui ne soit pas clé d'accès

- pas d'IKO ayant plus d'un composant path-type

- aucun IKO de référentiel path-type ne peut être défini sur le record-type ORIGIN

  Remarque : cette contrainte est tout à fait normale puisque
  - le path-type 1-N identifie toujours l'ORIGIN et
  - l'accés de la TARGET vers l'ORIGIN est implicite si l'inverse existe
  - il est insensé de définir un ordre dans le cas où l'ensemble des record-types à ordonner est toujours un singleton.

- pas de clé d'accès globale dont le référentiel est path-type qui ne soit pas également clé d'ordre.

2 LISTE PARAMETREE DES CONTRAINTES

La liste paramétrée suivante étudie un regroupement des contraintes des trois SGDs. Elle est structurée de façon à regrouper par concept les restrictions qui sont posées par l'un ou l'autre des SGDs retenus en allant du plus restrictif vers le moins restrictif.

Elle servira d'intermédiaire pour la recherche des buts qui peuvent être visés par l'utilisateur lors de ses demandes de transformations. De plus, la contruction d'une telle liste est indispensable pour la mise en oeuvre d'un vérificateur de conformité paramétrée.

Un des objectifs fixé dans cette étude est de couvrir un champs maximum de contraintes tant que son extension par rapport aux contraintes SQL, COBOL et CODASYL n'amène pas trop de complications au niveau de la structure ou au niveau du paramétrage.

Par exemple, une sous-contrainte de type "dont la borne inférieure est égale à" sera remplacée par une sous-contrainte de type "dont la borne inférieure est inférieure ou égale à".

RECORD-TYPE.

- pas de record-type sans items (COBOL,SQL)

ITEM.

- pas d'item

  - décomposable dont le niveau de décomposition est strictement supérieur à une valeur déterminée. Cette valeur vaut :
    - pour COBOL : 49
    - pour CODASYL : 99
    - pour SQL : 1.

  - répétitif

    - aucun (SQL)

    - variable
      - aucun
      - sans item compteur (COBOL, CODASYL)
      - ayant un ancêtre répétitif (COBOL)
      - qui ne soit pas le dernier dans le graphe de décomposition des items du record-type (COBOL)

- dont le niveau de répétitivité est strictement supérieur à une valeur déterminée.
  Cette valeur vaut 3 pour COBOL.

- dont la borne inférieure du facteur de répétitivité est inférieure ou égale à une valeur déterminée.
  Cette valeur vaut 0 pour COBOL et SQL.

  Remarque : cette sous-contrainte inclut la contrainte d'absence d'items facultatifs.

- de type compteur :

  - aucun

  - qui ne soit pas contenu dans le même record-type que l'item compté (CODASYL)

  - de type non numérique (COBOL, CODASYL)

  - décomposable (CODASYL, COBOL)

  - facultatif

  - répétitif

    Remarque : ces quatre dernières contraintes tombent dans le cas où elles sont déjà posées au niveau du MAG.

- de format :

  - booléen (COBOL, SQL)

  - de plus d'un nombre déterminé de positions numériques.
    Ce nombre vaut :
    - pour COBOL : 18
    - pour CODASYL : ?
    - pour SQL : ?

  - chaine de caractères d'une longueur plus grande qu'une valeur déterminée.
    Cette valeur vaut :
    - pour COBOL : ?
    - pour CODASYL : ?
    - pour SQL : 32768

PATH-TYPE.

- pas de path-type

  - aucun (COBOL, SQL)

  - de connectivité :

- 1-1 (CODASYL)
- N-N (CODASYL)
- N-1 de l'origine vers la cible, n'ayant pas d'inverse (CODASYL)

- sans accès (CODASYL)

- multiorigines (CODASYL)

- où le record-type ORIGIN est aussi TARGET (CODASYL)

- ayant le record-type SYSTEM comme TARGET (CODASYL)

FICHIER.

- pas de fichier ne contenant aucun record-type (COBOL)

- pas de record-type qui ne soit contenu dans aucun fichier (COBOL,CODASYL)

- pas de record-type contenu dans plusieurs fichiers (COBOL)

LES NOMS D'OBJETS.

Les types d'objets possibles dépendront du SGD cible.

- les noms d'objets ne peuvent appartenir à un ensemble déterminé.
  Cet ensemble sera formé

  - pour COBOL : des mots réservés et mots systèmes.

    Restriction : un item pourra avoir le nom réservé "FILLER".
  - pour CODASYL : ?
  - pour SQL : ?

- les noms d'objets ne peuvent contenir aucun des symboles suivants :

  - pour COBOL : ?
  - pour CODASYL : ?
  - pour SQL : ?

- les noms d'objets ne peuvent pas commencer par des symboles suivants

  - pour COBOL : le trait d'union, ?
  - pour CODASYL : ?
  - pour SQL : ?

- les noms d'objets ne peuvent pas se terminer par des symboles suivants

  - pour COBOL : les traits d'union, ?
  - pour CODASYL : ?
  - pour SQL : ?

- les noms d'objets doivent contenir au moins un des symbles suivants :

- pour COBOL : les lettres
- pour CODASYL : ?
- pour SQL : ?

- contraintes d'unicité des noms d'objets.

  - le nom du fichier l'identifie parmi

    - l'ensemble des fichiers de la BD (COBOL, CODASYL)

  - le nom du record-type l'identifie parmi

    - l'ensemble des record-types du même fichier (COBOL)
    - l'ensemble des record-types de la BD (SQL, CODASYL)

  - le nom de l'item l'identifie parmi

    - l'ensemble des items du même record-type (SQL)
    - l'ensemble des items de la BD (CODASYL)

  - si ces deux items sont ancêtres l'un de l'autre, ils doivent avoir des noms différents;

    sinon, l'ensemble complet des noms déclarés sur le chemin allant de l'ancêtre commun vers X inclusivement ne peut pas être un sous-ensemble de l'ensemble complet des noms déclarés sur le chemin allant de l'ancêtre commun vers Y inclusivement (COBOL)

- tout item doit avoir un nom différent de celui du record-type dans lequel il est contenu (COBOL).

IKO.

Les contraintes sur les IKO étant très différentes d'un SGD à l'autre, il apparaît inutile de les paramétrer. Par conséquent, la liste des contraintes sur les IKOs sera la réunion des contraintes des trois SGDs. Il n'y a d'ailleurs qu'une seule contrainte dont l'expression est la même ou similaire (dans le cas présent, il s'agit d'une contrainte commune aux SGDs COBOL et SQL) : "pas d'identifiants qui ne soient pas clés d'accès".

ANNEXE 2

TRANSFORMATIONS CONCRETES

1 Les ITEMs.

1.1 TRANSFORMATIONS LIEES A LA NOTION DE REPETITIVITE.

1.1.1 Elimination de la répétitivité par transformation d'un ITEM répétitif en un RECORD-TYPE.

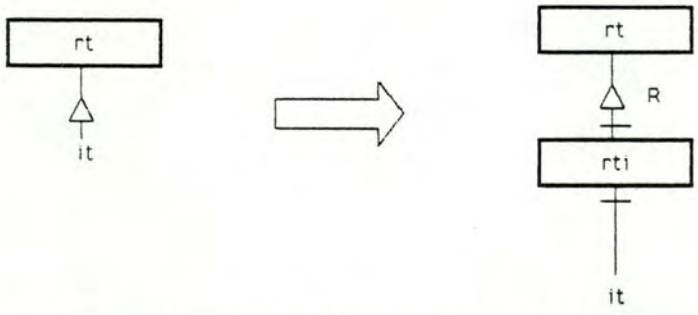


Figure 1.1: Elimination de la répétitivité par création d'un record-type intermédiaire.

Mapping du point de vue théorique.

Introduction d'un RECORD-TYPE intermédiaire rti dans un type d'associations CONTAINS entre un RECORD-TYPE rt et un ITEM principal it et choix des solutions qui évitent que it soit répétitif dans rti.

Opérations.

1. Création du RECORD-TYPE rti.
2. Transfert de it de rt vers rti.

Remarques.

1. it aura dans un premier temps les mêmes valeurs de propriétés dans rti que dans rt.
2. Tous les descendants de it seront automatiquement transférés dans rti.
3. Si it est de répétitivité variable relié à un ITEM

compteur, cet ITEM compteur ne sera pas transféré dans rti mais sera traité plus loin.

3. Création du PATH-TYPE R entre rt et rti.

Etude des ROLE dans R : donnés par l'utilisateur.

Etude des connectivités minimales dans R :

la connectivité minimale de rt dans R = 1
si it était obligatoire dans rt
la connectivité minimale de rt dans R = 0
sinon

la connectivité minimale de rti dans R = 1

Etude des connectivités maximales dans R :

la connectivité maximale de rt dans R = Many

la connectivité maximale de rti dans R
dépend à la fois
- de la propriété d'identifiant ou non de it de rt et
- de la propriété d'identifiant ou non de it de rti.

si it était identifiant de rt :
connectivité maximale de rti dans R = 1 et
it sera identifiant de rti;
si it était non identifiant de rt :
soit connectivité maximale de rti dans R = Many et
it sera identifiant de rti
soit connectivité maximale de rti dans R = 1 et
it sera identifiant de rti dans le PATH-TYPE R

4. Etude des nouvelles valeurs des propriétés de it dans rti.

Uniquement les propriétés concernant le caractère obligatoire, le caractère répétitif et le caractère identifiant pourront être modifiés.
it deviendra obligatoire et non répétitif dans rti.
Pour la modification du caractère identifiant : cfr l'étude des connectivités maximales dans R.

5. Etude des nouvelles contraintes d'intégrité qui devront éventuellement être ajoutées en fonction du type de la répétitivité de it dans rt en parallèle avec la traduction de l'éventuel compteur de répétitivité.

Si it était de répétivité fixe égale à X, la CI suivante doit être créée :
"tout record rt doit être relié par R à exactement X records rti".

Si it était de répétitivité variable relié à un ITEM compteur,
si ce compteur ne comptait pas d'autres ITEMs,
il faut le supprimer;
sinon, la CI suivante doit être créée :
"tout record rt doit être relié par R à un nombre de record rti égal à la valeur de l'ITEM compteur".

6. Etude des accès.

Si l'utilisateur exige l'équivalence du point de vue des accès :

1. Etude des accès dans R.

   Puisque l'accès du RECORD-TYPE vers l'ITEM existe toujours, rti devra toujours être accessible à partir de rt dans R.
   rt ne sera accessible à partir de rti que si it était clé d'accès dans rt.

2. Etude de it comme clé d'accès dans rti.

   it sera clé d'accès dans rti si it était clé d'accès dans rt.

3. Traduction de it comme composant (non unique) d'une clé d'accès définie sur rt.

Les préconditions assurent que de tels identifiants ne peuvent exister.

7. Traduction de it comme composant (non unique) d'un identifiant défini sur rt.

Les préconditions assurent que de tels identifiants ne peuvent exister.

Préconditions.

- il n'existe aucun RECORD-TYPE dans le schéma de même nom que rti;
- il n'existe aucun PATH-TYPE dans le schéma de même nom que R;
- it existe et est un ITEM principal répétitif;
- si it est composant d'une clé d'accès ou d'un identifiant, il en est l'unique composant;
  (En effet, de tels clés d'accès ou identifiants sont dans un premier temps trop compliqués à traduire (cfr. le niveau théorique.))
- les éventuels fils de it ne sont pas reliés à un compteur et ne sont composants ni d'identifiants, ni de clés d'accès.

Remarque.

La transformation inverse de reconstruction d'un ITEM répétitif par transformation d'un RECORD-TYPE en un ITEM ne sera pas offerte à ce niveau car ce dernier type de transformations pourrait être demandé même si l'ITEM qui avait été transformé en un RECORD-TYPE n'était pas répétitif.

1.1.2 Création d'un ITEM compteur.

Mapping du point de vue théorique.

Création d'un ITEM technique suivie d'une liaison compteur entre cet ITEM technique et un ITEM de répétitivité variable.

Opérations.

1. Création de l'ITEM compteur :

   - insertion de l'ITEM it counter dans le RECORD-TYPE qui contient it à l'endroit désiré par l'utilisateur;

   - étude des valeurs des propriétés de l'ITEM compteur :
     non décomposable et de type numérique,
     non répétitif et
     obligatoire.

2. Création de la liaison compteur.

Préconditions.

- l'ITEM compteur n'existe pas;
- it existe et est un ITEM de répétitivité variable pas encore lié à un compteur.

1.1.3 Suppression d'un ITEM compteur.

Mapping du point de vue théorique.

Suppression d'une liaision compteur suivie d'une suppression d'un ITEM technique.

Opérations.

1. Suppression de la liaison COUNTER entre it et it-counter.
2. Suppression de it-counter.

Préconditions.

- l'ITEM it-counter existe et
- it-counter ne joue le rôle de compteur que pour it.

Remarque.

* Des transformations du type 'report de la répétitivité d'un ITEM sur ses fils' (et inversément, 'report de répétitivités identiques d'ITEMs frères sur leur père') ne sont pas offertes ici car elles n'assurent pas d'équivalence sémantique. En effet, supposons la situation suivante où it est répétitif et a deux fils A et B. Le report de la répétitivité de it sur chacun de ses fils entraine la perte de l'information de couplage entre A et B.

* On pourrait également vouloir offrir à l'utilisateur la possibilité de transformer un ITEM non répétitif ou de répétitivité fixe par un ITEM de répétitivité variable accompagné d'une contrainte d'intégrité, mais le niveau d'utilité de ces transformations oblige à les proposer comme extension plutôt que de les analyser de suite.

1.2 TRANSFORMATIONS LIEES AU CARACTERE FACULTATIF.

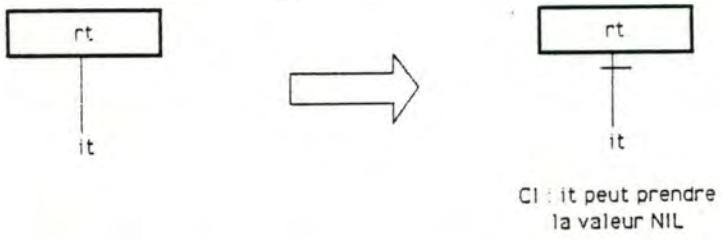


Figure 1.2: Transformations liées au caractère facultatif.

1.2.1 Transformation d'un ITEM facultatif en un ITEM obligatoire accompagné d'une contrainte d'intégrité exprimant la possiblité d'absence de valeurs.

Remarque préliminaire.

Cette transformation pourrait ne pas être analysée si l'on supposait qu'une contrainte SGBD d'absence d'ITEMs facultatifs sera prise en compte au moment de la conversion de schémas compatibles vers un texte DDL. Par exemple, le processeur COBOL ignorera dans le schéma la notion de facultatif/obligatoire.
Cependant, lors de cette conversion est-il permis de partir de schémas compatibles si ceux-ci contiennent encore des "invalidités" ? Et dans quelle mesure peut-on partir d'invalidités si le SGBD cible ignore les concepts qui lui sont liés ?

Pour éviter de s'attarder sur des questions sans réponse et dans un souci de complétude, l'ensemble des transformations offertes à l'utilisateur permettra de lever toutes les contraintes SGBD.

Mapping du point de vue théorique.

Remplacement d'une contrainte exprimée dans le schéma par une contrainte d'intégrité qui sera gérée par programme d'application.

Opérations.

1. Mise à 1 de la répétitivité minimale de it.

2. Création de la contrainte d'intégrité exprimant que it peut prendre la valeur NIL.

Précondition.

- it existe et est un ITEM facultatif.

1.2.2 Remplacement d'une contrainte d'intégrité exprimant la possibilité d'absence de valeurs par une répétitivité minimale nulle exprimée dans le schéma.

Mapping du point de vue théorique.

Remplacement d'une contrainte d'intégrité qui pourrait être exprimée dans le schéma par une contrainte dans le schéma.

Opérations.

1. Mise à 0 de la répétitivité minimale de it.
2. Suppression de la contrainte d'intégrité exprimant la possibilité d'absence de valeurs pour it.

Préconditions.

- il existe une contrainte d'intégrité exprimant la possiblité d'absence de valeurs pour it;
- l'ITEM it existe.

Remarque.

La transformation d'un ITEM en un RECORD-TYPE permet également d'éliminer un ITEM facultatif. Mais cette transformation étant offerte à l'utilisateur sans la nécessité de connaitre le but des demandes de cette transformation, celle-ci ne sera ni analysée ni offerte à ce niveau.

1.3 MODIFICATION DU FORMAT D'UN ITEM.

1.3.1 Modification de la longeur d'un ITEM.

Mapping du point de vue théorique.

Changement de la valeur d'une propriété d'un objet dont l'effet n'entraine aucune modification sémantique.

Opération.

Modification de la longueur de l'ITEM it concerné.

Précondition.

- it existe et est un ITEM non décomposable.

1.3.2 Modification du type d'un ITEM.

cfr. modification de la longueur d'un ITEM.

1.4 AJOUT D'UN ITEM TECHNIQUE (pour éviter l'absence d'ITEMs dans un RECORD-TYPE).

Mapping du point de vue théorique.

Création d'un ITEM technique

Opérations.

1. Création de l'ITEM technique it de longeur, type, répétitivité minimale, répétitivité maximale, ... donnés par l'utilisateur.
2. Rattachement de it à son père.

Préconditions.

- it n'existe pas;
- it est non décomposable.

Remarque.

Pour une raison de simplicité de cette transformation et d'autant plus que celle-ci sera demandée dans un but technique, l'ITEM créé sera un ITEM
- non décomposable,
- principal et
- non relié à un ITEM compteur puisque dans la plupart des cas, ceci correspondra aux désirs de l'utilisateur. L'utilisateur n'admettant pas cette prise de décision pourra toujours, après cette transformation, en appliquer d'autres pour que son but complet soit atteint.

1.5 SUPPRESSION D'UN ITEM TECHNIQUE.

Mapping du point de vue théorique.

Suppression d'un ITEM technique (ou d'un objet technique).

Opérations.

1. Suppression des éventuelles contraintes d'intégrité reliées à it;
2. Suppression de l'ITEM technique.

Précondition.

- it existe et est un ITEM technique (c'est-à-dire porteur d'aucune sémantique et par conséquent ne peut avoir été dupliqué) et ne peut être composant d'une clé d'accès si l'utilisateur désire l'équivalence du point de vue des accès.

Remarque.

Parallèlement à la création d'un ITEM technique, l'ITEM it sera supposé
- non décomposable,
- principal et
- non relié à un ITEM compteur.

1.6 MODIFICATION DU GRAPHE DE DECOMPOSITION D'ITEMS D'UN RECORD-TYPE.

1.6.1 Applatissement d'ITEMs ou suppression d'un niveau de décomposition d'ITEMs.

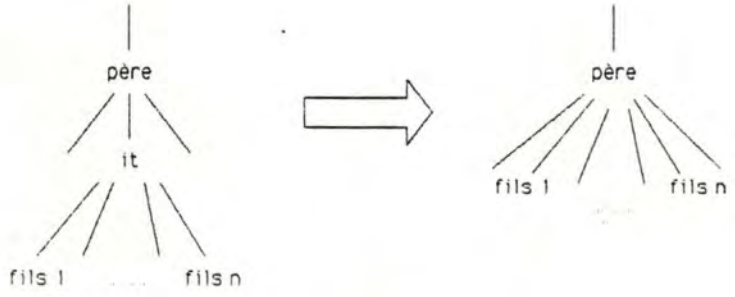


Figure 1.3: Suppression d'un niveau de décomposition d'ITEMs.

Mapping du point de vue théorique.

Applatissement d'ITEMs.

Opérations.

1. Traitement des identifiants/clés d'accès ayant comme composant l'ITEM it à applatir :
   remplacement du composant it par un composant correspondant à chaque fils de it (l'ordre des composants n'a aucune importance pour les identifiants, les clés d'accès)

2. Les fils de it deviennent fils du père de it (avec mise à jour de la séquence des ITEMs dans le père de it, c'est-à-dire avec insertion des fils de it à l'endroit où se trouvait it).

3. Suppression de it.

Précondition.

- it existe et est un ITEM décomposable et non répétitif.

1.6.2 Rassemblement d'ITEMs ou création d'un niveau de décomposition supplémentaire.

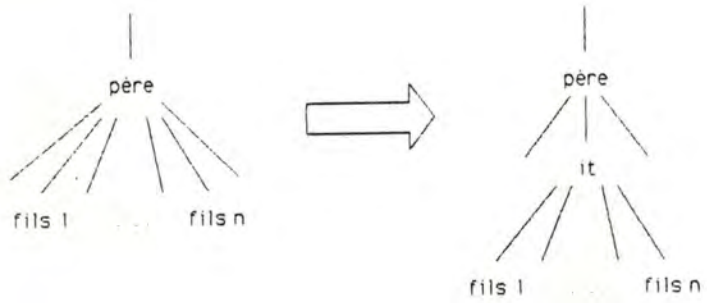


Figure 1.4: Création d'un niveau de décomposition supplémentaire.

Mapping du point de vue théorique.

Rassemblement d'ITEMs.

Opérations.

Soient - père-item l'ITEM ou le RECORD-TYPE père des ITEMs à rassembler,
- it l'ITEM à créer représentant le rassemblement et
- liste-it la liste des ITEMs à rassembler en it.
(L'ordre des ITEMs dans cette liste reflètera l'ordonnancement dans it.)

1. Création de it :
it sera un ITEM - non répétitif,
- obligatoire et
- décomposable.
(aucune information n'a besoin d'être demandée à l'utilisateur sur les valeurs des propriétés de it.

2. Insertion de it dans le graphe de décomposition.
- l'ITEM/RECORD-TYPE père-item sera le père de it et
- it deviendra le père des ITEMs se trouvant dans la liste liste-it.

Puisque aucune restriction n'est posée sur l'emplacement des ITEMs frères à rassembler, l'emplacement de l'ITEM it n'est pas une information déductible du contenu de liste-it. Par conséquent, et pour plus de facilité, l'ITEM it deviendra le dernier fils de père-item. L'utilisateur n'acceptant pas cette prise de décision pourra, après la transformation , déplacer it grâce aux transformations de modification de la séquence d'ITEMs.
L'ordonnancement des ITEMs (autre que it) dans père-item sera effectué de façon à conserver l'ancien séquencement, c'est-à-dire que, si deux ITEMs iti et itj, fils de

père-item, ne sont pas repris comme fils de it et que iti précédait itj avant la transformation alors iti précédera itj après la transformation (et ce, pour tout iti, itj avec i différent de j).

3. Mise-à-jour de la séquence des ITEMs fils de it selon l'ordre des ITEMs dans liste-it.

Pécondition.

- tous les ITEMs àrassembler existent et sont frères l'un de l'autre.

Remarque.

Des transformations de types suivants pourraient également être offertes à l'utilisateur :

- montée d'un ITEM d'un niveau de décomposition c'est-à-dire rendre un ITEM fils de son grand-père;

- descente d'un ITEM d'un niveau de décomposition c'est-à-dire rendre un ITEM fils d'un de ses frères.

Celles-ci ne seront cependant pas offertes à l'utilisateur en raison des préconditions trop restrictives dans leur application et de l'absence d'une réelle utilité pratique.

1.7 MODIFICATION DE LA SEQUENCE D'ITEMs.

1.7.1 Inversion de l'emplacement de deux ITEMs frères.

Mapping du point de vue théorique.

Inversion de deux objets dans un ensemble ordonné.

Opérations.

Inversion du numéro de séquence de deux ITEMs it1 et it2.

Précondition.

- it1 et it2 existent et sont des ITEMs frères.

1.7.2 Déplacement d'un ITEM à un endroit déterminé dans sa séquence.

Mapping du point de vue théorique.

Une ou plusieurs inversions de deux objets dans un ensemble ordonné.

Opérations.

Soit it le jème fils de it-père et devient le ième fils par cette transformation.

Si déplacement vers la gauche :
augmentation de 1 du numéro de séquence des ITEMs dont le numéro est compris entre j et i (i compris).

Si déplacement vers la droite :
diminution de 1 du numéro de séquence des ITEMs dont le numéro est compris entre i et j (i compris).

Précondition.

- le père de l'ITEM à dupliquer à au moins i fils.

1.8 TRANSFORMATION D'UN ITEM EN UN RECORD-TYPE.

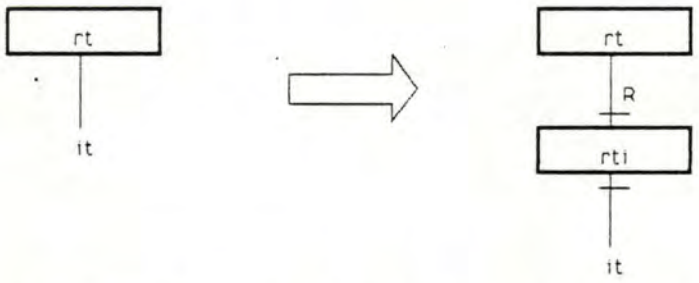


Figure 1.5: Transformation d'un item en un record-type.

Mapping du point de vue théorique.

Introduction d'un RECORD-TYPE intermédiaire rti dans un type d'associaitions CONTAINS entre un RECORD-TYPE et un ITEM principal it.

Opérations.

Remarque : les opérations sont globalement les mêmes que dans le cas de l'élimination de la répétitivité par transformation d'un ITEM répétitif en un RECORD-TYPE.
Cependant, il faut apporter certains changements dûs au fait que :
- l'ITEM it n'est pas nécessairement répétitif
et - dans le cas où it est répétitif, le but de cette transformation n'est pas nécessairement ici d'éliminer la répétitivité.

Par conséquent, nous n'examinerons que les opérations qui doivent être adaptées en fonction de ces considérations.

1. L'étude des connectivités maximales dans R et l'étude des nouvelles valeurs des propriétés de it dans rti sont rassemblées en l'étude de la traduction des propriétés de it dans rt.

   Le caractère identifiant ou non de it dans rt sera traduit à la fois dans la connectivité maximale de rti dans R et dans le caractère identifiant ou non de it dans rti.

   Le caractère répétitif ou non de it dans rt sera traduit à la fois dans la connectivité maximale de rt dans R et dans le caractère répétitif ou non de it dans rti.

   Traduction du caractère identifiant ou non de it dans rt :

   si it était identifiant dans rt,
   alors la connectivité maximale de rti dans R = 1 et

it est identifiant de rti;

si it était non identifiant dans rt,
alors soit la connectivité maximale de rti dans R = Many
et it est identifiant de rti;
alors soit la connectivité maximale de rti dans R = 1
et it est non identifiant de rti;

Traduction du caractère répétitif ou non de it dans rt :

si it était non répétitif dans rt,
alors la connectivité maximale de rt dans R = 1 et
it est non répétitif dans rti;

si it était répétitif dans rt,
alors soit la connectivité maximale de rt dans R = 1
et it garde sa répétitivité;
(sa répétitivité minimale
sera égale à 1.)
alors soit la connectivité maximale de rt dans R = Many
et it devient répétitif dans rti;

2. Etude des nouvelles contraintes d'intégrité qui devront éventuellement être ajoutées.
Celle-ci ne sera effectuée que dans le cas où it était répétitif dans rt et ne l'est plus dans rti.
Elle sera effectuée de la même façon que pour l'élimination de la répétitivité par transformation d'un ITEM répétitif en un RECORD-TYPE.

Remarque : dans le cas où l'ITEM it est de répétitivité variable dans rt et relié à un compteur, mais que it devient répétitif dans rti, l'ITEM compteur devra être transféré de rt vers rti.

3. Etude des accès.

l'étude des accès dans R et l'étude de it comme clé d'accès de rti : cfr. élimination de la répétitivité par transformation d'un ITEM en un RECORD-TYPE.

Traduction de it comme composant (non unique) d'une clé d'accès définie sur rt.
Dans un premier temps, les préconditions assurent que de telles clés d'accès ne peuvent exister.

4. Traduction de it comme composant (non unique) d'un identifiant défini sur rt.
Dans un premier temps, les préconditions assurent que de tels identifiants ne peuvent exister.

Préconditions.
cfr. élimination d'un ITEM répétitif,
mais it n'est pas nécessairement répétitif.

1.9 TRANSFORMATION D'UN RECORD-TYPE EN UN ITEM.

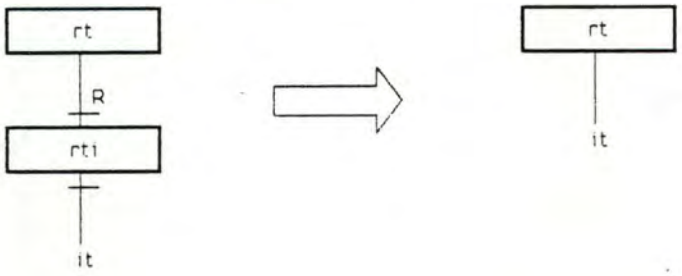


Figure 1.6: Transformation d'un RECORD-TYPE en un ITEM.

Mapping du point de vue théorique.

Suppression d'un RECORD-TYPE intermédiaire introduit dans un type d'associations CONTAINS entre un RECORD-TYPE et un ITEM it.

Opérations.

1. Transfert de it de rti vers rt.

2. Etude des propriétés de it dans rt.

   1. le caractère répétitif ou non de it dans rt :

      si la connectivité maximale de rt dans R = 1,
      alors it aura la même répétitivité dans rt que dans rti.

            N.B. si it était relié à un ITEM compteur dans rti,
                 ce compteur devra être transféré dans rt.

      si la connectivité maximale de rt dans R = Many,
      alors il faut examiner les éventuelles contraintes
            d'intégrité sur la connectivité maximale de rt dans
            R. Si une telle contrainte d'intégrité n'existe
            pas, it sera de répétitivité illimitée non relié à
            un ITEM compteur.

   2. le caractère identifiant ou non de it dans rt :

      it sera identifiant de rt
       si et seulement si - it est identifiant de rti et
                          - la connectivité maximale de rti dans
                            R = 1.

   3. le caractère facultatif ou non de it dans rt :

      it sera facultatif dans de rt

si et seulement si la connectivité minimale de rt dans R = 1.

4. it comme clé d'accès dans rt :

it sera clé d'accès dans rt
si et seulement si it est clé d'accès dans rti.

3. Suppression des éventuels IKOs définis sur rti.

Remarque : tous les IKOs identifiants - clés d'accès possibles définis sur rti auront été traduits lors de l'étude des propriétés de it dans rt.

4. Suppression d'éventuelles contraintes sur la connectivité maximale de rt dans R.

5. Suppression du PATH-TYPE R.

6. Suppression du CECORD-TYPE rti.

Préconditions.

- rti peut être considéré comme un RECORD-TYPE intermédiaire entre un RECORD-TYPE et un ITEM, c'est-à-dire que :
  - rti n'est membre que d'un seul PATH-TYPE (non récursif et non à extrémité(s) multi-membres);
  - rti ne contient au maximum qu'un seul ITEM principal accompagné éventuellement de son ITEM compteur;
  - les valeurs des propriétés de it dans rti et les connectivités maximales dans R sont telles qu'elles ne peuvent parvenir que de la traduction des propriétés de it dans rt lors de la transformation de l'ITEM it en le RECORD-TYPE rti.
    Autrement dit :
    - it ne peut pas être répétitif dans rti si la connectivité maximale de rt dans R est égale à Many et
    - it ne peut pas être non identifiant (sans le composant PATH-TYPE R) sur rti si la connectivité maximale de rti dans R est égale à Many;
- it est obligatoire dans rti;
- la connectivité maximale de rt dans R est égale à 1.

1.10 TRANSFERT D'UN ITEM D'UN RECORD-TYPE VERS CHACUN DES RECORD-TYPES JOUANT UN ROLE DANS SON IDENTIFIANT.

1.11 ET INVERSEMENT.

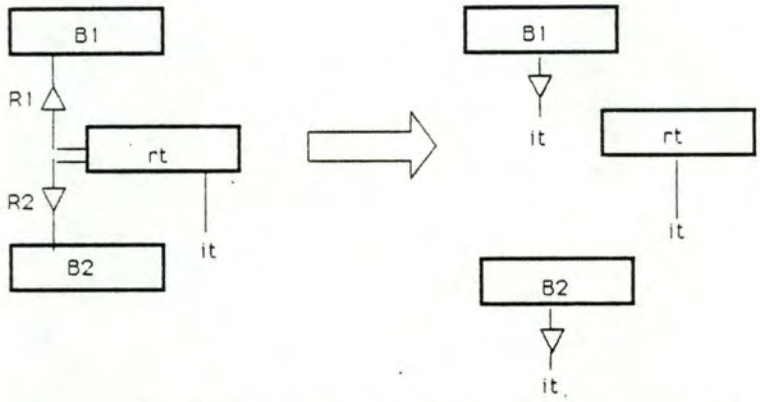


Figure 1.7: Tranfert d'items entre record-type.

Mapping du point de vue théorique.

Rotation d'un type d'associations CONTAINS entre un RECORD-TYPE rt et un ITEM principal it en utilisant un identifiant servant de base à la rotation qui ne soit composé que de PATH-TYPEs (et la transformation inverse).

Intérêt de cette transformation.

Cette transformation n'est pas indispensable mais peut être très utile pour accroitre les performances au niveau des accès. Etant donné la complexité de cette transformation, elle ne sera pas analysée dans ce rapport mais sera proposée comme extension.

2 Les PATH-TYPEs.

2.1 ELIMINATION D'UN PATH-TYPE PAR DUPLICATION D'ITEMs.

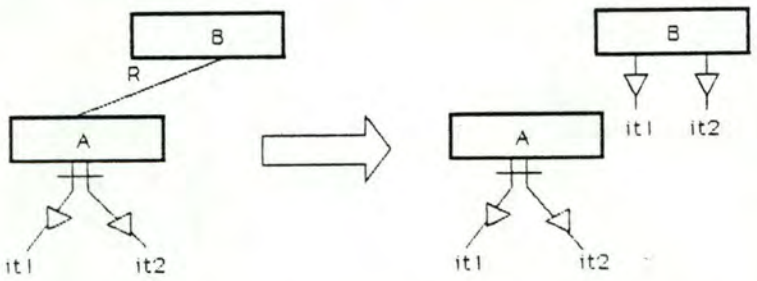


Figure 2.1: Elimination d'un path-type par duplication d'items.

Mapping du point de vue théorique.

Rotation d'un PATH-TYPE en utilisant un identifiant qui sert de base à la rotation n'ayant que des composants ITEMs - éventuellement précédé de la création d'un identifiant technique.

Remarques préliminaires.

1. R sera supposé non récursif.
   En effet, cette même transformation sera proposée comme l'une des solutions d'élimination de la récursivité.

2. R sera également supposé non multitype.

3. Nous supposons connu le sens de transfert des ITEMs à dupliquer.
   Soit A le RECORD-TYPE contenant les ITEMs à dupliquer et B le RECORD-TYPE dans lequel ils seront "copiés".

4. Nous supposons également connu l'ensemble des ITEMs it1,...itn à "copier" de A dans B.

Opérations.

1. Création d'un éventuel identifiant technique défini sur A formé de l'ensemble des ITEMs à "copier" dans B.

Remarque.
Les préconditions assurent que l'ensemble des ITEMs iti forment un identifiant défini sur A dans le cas où tous ces ITEMs existent déjà dans A. Par conséquent, la création d'un identifiant technique ne sera effectuée que si au moins un des ITEMs iti à copier n'existe pas encore dans A. Cette précondition a pour but d'assurer l'équivalence sémantique.

2. Pour chaque ITEM iti : le copier dans B.

Remarque.

Les préconditions assurent qu'aucun des ITEMs iti n'est répétitif dans A et par conséquent, le problème du transfert des éventuels compteurs reliés aux iti ne se posent pas. Mais se pose toujours le problème du transfert des éventuels compteurs reliés aux fils de iti pour les iti décomposables.

3. Etude des propriétés de chaque ITEM iti dans B.

Le caractère obligatoire de iti dans B :

iti est obligatoire dans B si et seulement si
la connectivité minimale de B dans R = 1,
iti est facultatif sinon.

Remarque : iti n'a pas besoin d'être obligatoire dans A pour être obligatoire dans B.
De toute façon, les préconditions assurent que tout A relié par R à B a obligatoirement une valeur pour l'ITEM iti.

Le caractère répétitif de iti dans B :

si la connecitivté maximale de B dans R = 1,
alors iti sera non répétitif dans B;
sinon iti sera de répétitivité variable illimitée dans B.

Remarque : l'utilisateur pourra par la suite éliminer la répétitivité illimitée en donnant des bornes techniques.

Le caractère identifiant de iti dans B :

iti sera identifiant dans B uniquement s'il est le seul ITEM créé (c'est une condition nécessaire mais pas suffisante). Dans ce cas, l'étude du caractère identifiant de iti dans B est traitée dans le point suivant.

4. Création d'un identifiant défini sur B ayant l'ensemble des

ITEMs iti comme composants.

Un tel identifiant ne sera créé que si la connectivité maximale de A dans R = 1.

5. Etude de l'ensemble des iti comme clé d'accès dans B.

Une telle clé d'accès ne sera créée que si B était accessible à partir de A via R.

Remarque : les préconditions d'équivalence du point de vue des accès assurent que dans ce cas, une clé d'accès ayant comme unique composant l'ensemble des iti est définie sur A.

6. Traitement des clés d'accès/identifiants ayant R comme composant (non unique).

Les préconditions assurent que de telles clés d'accès/identifiants définis sur A n'existent pas. De tels identifiants/clés d'accès définis sur B seront traduits en remplaçant ce composant par l'ensemble des ITEMs copiés dans B.

7. Etude des contraintes d'intégrité supplémentaires.

1. Pour chaque iti : création de la CI suivante :
iti(:B) IN iti(:A)

Remarque : le signe 'IN' est remplacé par le signe d'égalité dans le cas où la connectivité minimale de A dans R est égale à 1.

2. Création de la Ci spécifiant que
l'ensemble des B ayant une valeur pour it1 =
l'ensemble des B ayant une valeur pour it2 =
l'ensemble des B ayant une valeur pour ... =
l'ensemble des B ayant une valeur pour itn
dans le cas où la connectivité minimale de B dans R n'était pas égale à 1. (Puisque dans ce cas, tous les iti seront facultatifs dans B.)

8. Suppression du PATH-TYPE R ainsi que de toutes ses liaisons avec d'autres objets (RECORD-TYPE, IKO, ..).

Préconditions.

- R est non récursif et n'a pas plus de deux RECORD-TYPEs membres;

- si tous les ITEMs iti de A à copier dans B existent déjà dans A, alors il existe un identifiant défini sur A ayant l'ensemble complet de ces ITEMs comme uniques composants;
- aucun des iti n'est répétitif;
- aucun des descendants dans iti n'est relié a un ITEM compteur;
- tout A relié par R à B a obligatoirement une valeur pour chaque iti;
  Remarque : cette contrainte est automatiquement vérifiée si tous les iti sont obligatoires dans A.
- le PATH-TYPE R ne peut être composant d'un identifiant/clé d'accès défini sur A;
- il existe une clé d'accès définie sur A ayant l'ensemble complet des ITEMs iti comme uniques composants;
- si plusieurs ITEMs doivent être copiés dans B, la connectivité maximale de B dans R doit être égale à 1.

2.2 REMPLACEMENT D'UN PATH-TYPE PAR LIAISON DE L'UNE DE SES EXTREMITES A L'IDENTIFIANT DE L'AUTRE EXTREMITE.

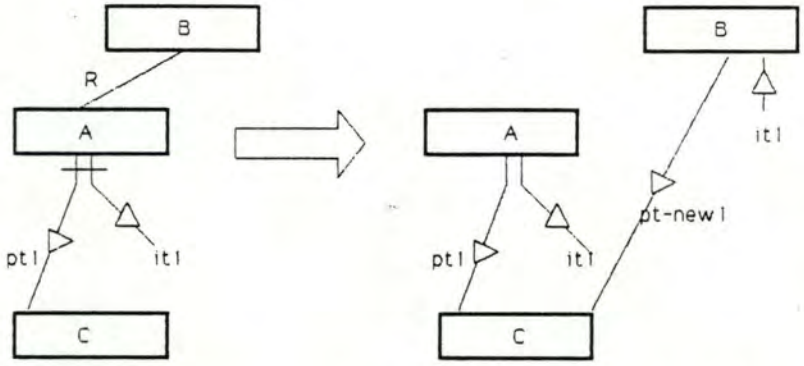


Figure 2.2: Remplacement d'un path-type par liaison de d'une de ses extrémités à l'identifiant de l'autre extrémité.

Mapping du point de vue théorique.

Rotation d'un PATH-TYPE en utilisant un identifiant qui sert de base à la rotation ayant au moins un composant PATH-TYPE (sinon nous nous trouvons dans les cas de l'élimination d'un PATH-TYPE par duplication d'ITEMs).

Remarques préliminaires.

1. Comme pour l'élimination d'un PATH-TYPE par duplication d'ITEMs, le PATH-TYPE R à transformer sera supposé non récursif et non multitype.

2. Nous supposons connue l'extrémité de R sur laquelle est définie l'identifiant : soit A.
   B sera l'autre extrémité de R.

3. Est également supposé connu l'identifiant de A auquel B sera relié afin de remplacer.

4. Cette transformation est une transformation non indispensable mais peut être très utile du point de vue performance au niveau des accès.

Opérations.

1. Liaison de B à chaque composant de l'identifiant de A

   Pour chaque composant ITEM iti : le copier dans B.
   Pour chaque composant PATH-TYPE pti :

créer un PATH-TYPE pt-newi ayant comme membres B et le RECORD-TYPE autre que A, membre de pti (soit rti ce RECORD-TYPE).

2. Etude des propriétés dans B de chaque ITEM iti composant de l'identifiant.

Cfr l'étude des propriétés de chaque iti dans B dans le cas de l'élimination de PATH-TYPEs par duplication d'ITEMs.
Il faut cependant noter que l'ITEM iti ne sera jamais identifiant de B puisque l'identifiant doit avoir au moins un composant PATH-TYPE.

3. Etude des propriétés des PATH-TYPEs pti-newi.

Connectivités minimales dans pt-newi.

La connectivité minimale de B dans pt-newi =
la connectivité minimale de B dans R.

La connectivité minimale de rti dans pt-newi = 1
si et seulement si la connectivité minimale de rti dans pti est égale à 1 (c'est-à-dire que tout RECORD rti est relié à au moins un RECORD A) et l'ensemble des RECORD A reliés par R à un RECORD B est égal à (et non seulement inclu dans) l'ensemble des RECORD A reliés par pti à un RECORD rti.

Remarque.

La contrainte que l'ensemble des A reliés par R à B soit égal à l'ensemble des A reliés par pti à rti est automatiquement vérifiée si la connectivité minimale de A dans R est égale à 1. Ceci découle du fait que les préconditions assurent que l'ensemble des A reliés par R à B est inclu dans l'ensemble des A reliés par pti à rti.

Connectivités maximales dans pt-newi.

La connectivité maximale de B dans pt-newi =
la connectivité maximale de B dans R.

La connectivité maximale de rti dans pt-newi =
maximum (la connectivité maximale de rti dans pti,
la connectivité maximale de A dans R).

4. Création d'un identifiant défini sur B ayant l'ensemble complet des iti et pti comme uniques composants.

Cfr élimination d'un PATH-TYPE par duplication d'ITEMs mais remplacer le terme iti par iti-pti.

5. Etude de l'ensemble des iti-pti comme clé d'accès dans B.

   Cfr élimination d'un PATH-TYPE par duplication d'ITEMs mais remplacer le terme iti par iti-pti.

6. Etude de l'ensemble des iti-pti comme clé d'accès dans A.

   Cfr élimination d'un PATH-TYPE par duplication d'ITEMs mais remplacer le terme iti par iti-pti.

7. Traitement des clés d'accès/identifiants ayant R comme composant (non unique).

   Cfr élimination d'un PATH-TYPE par duplication d'ITEMs mais le composant PATH-TYPE R sera remplacé non seulement par les ITEMs iti copiés dans B mais y seront ajoutés des composants correspondants aux PATH-TYPEs pt-newi.

8. Etude des contraintes d'intégrité supplémentaires.

   1. Pour chaque composant de l'identifiant défini sur A

      si ce composant est un ITEM (soit iti)
      alors création de la contrainte d'intégrité
      iti (: B) IN iti (: A)
      sinon (ce composant est un PATH-TYPE : soit pti)
      création de la contrainte d'intégrité
      rti (pt-newi : B) IN rti (pti : A).

   Remarques.

   1. Comme dans le cas de l'élimination de PATH-TYPEs par duplication d'ITEMs, le signe IN est remplacé par le signe d'égalité dans le cas où la connectivité minimale de A dans R est égale à 1.

   2. Les contraintes d'intégrité sur les PATH-TYPEs pt-newi n'ont pas à être créées si la connectivité minimale de rti dans pt-newi est égale à 1 puisqu'elles seront automatiquement vérifiées dans ce cas.

   2. Création de la contrainte d'intégrité spécifiant que l'ensemble des B ayant une valeur pour l'ITEM iti ou contraint d'exister dans pt-newi doit avoir une valeur pour tous les autres ITEMs iti et être contraint d'exister dans tous les PATH-TYPEs pt-newi.

9. Suppression du PATH-TYPE R ainsi que toutes ses liaisons avec

d'autres objets (RECORD-TYPE, IKO, contriantes d'intégrité).

Préconditions.

- le PATH-TYPE R est non récursif et non multitype;
- tous les ITEMs iti existent dans le RECORD-TYPE A;
- tous les PATH-TYPEs pti ont le RECORD-TYPE A pour membre;
- aucun des iti n'est répétitif;
- la connectivité maximale de A est égale à 1 dans tous les PATH-TYPEs pti;
- tout A relié par R à B a une valeur pour chaque ITEM iti;
  Remarque : cette contrainte est automatiquement vérifiée si tous les iti sont obligatoires dans A.
- tout A relié par R à B est contraint d'exister dans chaque PATH-TYPE pti;
  Remarque : cfr contrainte 6.
- R ne peut être composant d'un identifiant/clé d'accès défini sur A;
  Pour la justification, veuillez consulter l'étude théorique.
- l'identifiant de A auquel B sera relié est clé d'accès;
  Pour la justification, veuillez consulter l'étude théorique.
- si cet identifiant a plus d'un composant, la connectivité maximale de B dans R est égale à 1;
- le(s) composant(s) PATH-TYPE(s) de cet identifiant ne peut (peuvent) être ni récursifs ni multitypes.
  Pour la justification, veuillez consulter l'étude théorique.

2.3 REMPLACEMENT DE CONNECTIVITES RESTRICTIVES EXPRIMEES DANS LE SCHEMA PAR DES CONTRAINTES D'INTEGRITE.

Mapping du point de vue théorique.

Remplacement d'une contrainte exprimée dans le schéma par une contrainte d'intégrité qui sera traitée par programmes d'application.

2.3.1 Remplacement d'une connectivité maximale connue dans le schéma par une contrainte d'intégrité.

Opérations.

1. Création de la contrainte d'intégrité sur la connectivité maximale d'un membre donné A dans un PATH-TYPE donné R.
2. La connectivité maximale de ce membre devient infinie.

Préconditions.

- le PATH-TYPE R existe;
- A identifie un membre du PATH-TYPE R;
- la connectivité maximale de A dans R n'est pas infinie.

2.3.2 Remplacement d'une connectivité minimale autre que 0 dans le schéma par une contrainte d'intégrité.

Opérations.

1. Création de la contrainte d'intégrité sur la connectivité minimale d'un membre donné A dans un PATH-TYPE donné R.
2. La connectivité minimale de ce membre devient nulle.

Préconditions.

- le PATH-TYPE R existe;
- A identifie un membre du PATH-TYPE R;
- la connectivité minimale de A dans R n'est pas nulle.

2.3.3 Remplacement d'une contrainte d'intégrité sur une connectivité maximale par une connectivité plus restrictive dans le schéma.

Opérations.

1. Expression dans le schéma de la contrainte d'intégrité sur la connectivité maximale d'un membre donné A dans un PATH-TYPE donné R.

2. Suppression de la contrainte d'intégrité.

Préconditions.

- le PATH-TYPE R existe;
- A identifie un membre du PATH-TYPE R;
- il existe une contrainte d'intégrité sur la connectivité maximale de A dans R.

2.3.4 Remplacement d'une contrainte d'intégrité sur une connectivité minimale par une connectivité plus restrictive dans le schéma.

Opérations.

1. Expression dans le schéma de la contrainte d'intégrité sur la connectivité minimale d'un membre donné A dans un PATH-TYPE donné R.

2. Suppression de la contrainte d'intégrité.

Préconditions.

- le PATH-TYPE R existe.
- A identifie un membre du PATH-TYPE R.
- il existe une contrainte d'intégrité sur la connectivité minimale de A dans R.

2.4 TRANSFORMATION D'UN PATH-TYPE EN UN RECORD-TYPE.

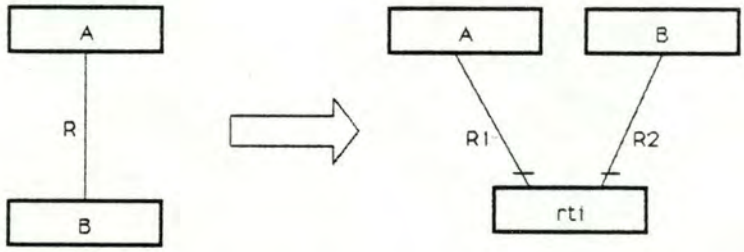


Figure 2.3: Transformation d'un path-type en un record-type.

Mapping du point de vue théorique.

Introduction d'un RECORD-TYPE intermédiaire rti dans un PATH-TYPE R.

Remarques.

1. R sera supposé non récursif.
   En effet, cette même transformation sera proposée comme l'une des solutions d'élimination de la récursivité.

2. R sera également supposé non multitype.

3. Soient A et B les deux RECORD-TYPEs membres de R.

Opérations.

1. Création du RECORD-TYPE rti.

2. Création du PATH-TYPE R1 entre rti et A.

3. Création du PATH-TYPE R2 entre rti et B.

4. Etude des connectivités minimales dans R1 et R2.

   La connectivité minimale de A dans R1 =
   la connectivité minimale de A dans R.

   La connectivité minimale de B dans R2 =
   la connectivité minimale de B dans R.

   La connectivité minimale de rti dans R1 = 1 et
   la connectivité minimale de rti dans R2 = 1.

5. Etude des connectivités maximales dans R1 et R2.

Remarque : La connectivité maximale de A (resp. B) dans R sera traduite à la fois dans la connectivité maximale de A (resp. B) dans R1 (resp. R2) et dans la connectivité maximale de rti dans R2 (resp. R1).

Connectivités maximales de A dans R1 et de rti dans R2 :

soit la connectivité maximale de A dans R1 =
la connectivité maximale de A dans R et
la connectivité maximale de rti dans R2 = 1
soit l'inverse.

Connectivités maximales de B dans R2 et de rti dans R1 :

soit la connectivité maximale de B dans R2 =
la connectivité maximale de B dans R et
la connectivité maximale de rti dans R1 = 1
soit l'inverse.

6. Création d'un identifiant pour rti ayant R1 et R2 comme uniques composants.

Cet identifiant ne sera créé que si la connectivité maximale de A dans R1 est différent de 1 et que la connectivité maximale de B dans R2 est différent de 1.

7. Etude des accès dans R1 et R2.

S'il existe un accès de A vers B dans R
alors il existe un accès de A vers rti dans R1 et
il existe un accès de rti vers B dans R2.

S'il existe un accès de B vers A dans R
alors il existe un accès de B vers rti dans R2 et
il existe un accès de rti vers A dans R1.

8. Traduction des identifiants/clés d'accès ayant R comme composant.

Les préconditions assurent que de tels identifiants/clés d'accès n'existent pas.

9. Suppression du PATH-TYPE R et de toutes les liaisons avec d'autres objets (RECORD-TYPE, IKO, contraintes d'intégrité).

Préconditions.

- le PATH-TYPE R existe;
- le RECORD-TYPE rti n'existe pas;
- les PATH-TYPE R1 et R2 sont différents et n'existent pas;
- R est ni récursif, ni multitype;
- il n'existe aucun identifiant/clé d'accès ayant R comme composant (justification : cfr. niveau théorique).

2.5 TRANSFORMATION D'UN RECORD-TYPE EN UN PATH-TYPE.

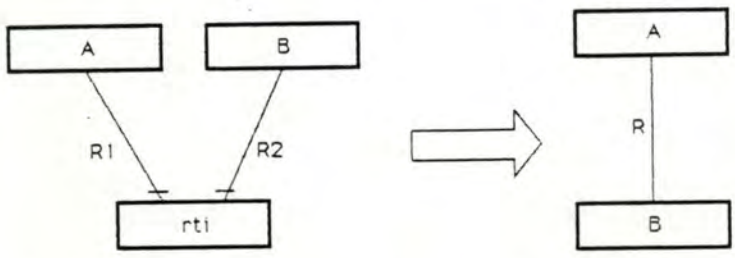


Figure 2.4: Transformation d'un record-type en un path-type.

Mapping du point de vue théorique.

Suppression d'un RECORD-TYPE intermédiaire rti introduit dans un PATH-TYPE R.

Remarques.

1. Les PATH-TYPE R1 et R2 peuvent être connus à partir de rti puisqu'eux uniquement portent sur rti.

2. Nous supposons que R était non récursif. (R serait récursif si rti n'était pas le seul membre commun entre R1 et R2.)

3. Soit A le RECORD-TYPE membre de R1 (autre que rti) et
   B le RECORD-TYPE membre de R2 (autre que rti).
   (A différent de B selon la deuxième remarque.)

4. Nous supposons que R était également non multitype.
   Par conséquent, R1 et R2 seront non multitype.

Opérations.

1. Création du PATH-TYPE R entre A et B.

2. Etude des connectivités minimales dans R.

   La connectivité minimale de A dans R =
   la connectivité minimale de A dans R1.

   La connectivité minimale de B dans R =
   la connectivité minimale de B dans R2.

3. Etude des connectivités maximales dans R.

   La connectivité maximale de A dans R =

maximum(la connectivité maximale de A dans R1,
la connectivité maximale de rti dans R2).

La connectivité maximale de B dans R =
maximum(la connectivité maximale de B dans R2,
la connectivité maximale de rti dans R1).

4. Suppression de l'éventuel identifiant défini sur rti ayant R1 et R2 comme composants.

5. Etude des accès dans R.

S'il existe un accès de A vers rti dans R1
alors il existe un accès de A vers B dans R.

S'il existe un accès de B vers rti dans R2
alors il existe un accès de B vers A dans R.

6. Suppression de R1, R2 et rti.

Préconditions.

- rti peut être considéré comme PATH-TYPE, c'est-à-dire :
  - rti ne contient aucun ITEM;
  - rti n'est membre que de deux PATH-TYPEs (soient R1 et R2);
  - la connectivité maximale de rti dans R1 (resp. R2) et la connectivité maximale du membre autre que rti dans R2 (resp. R1) ne peuvent être simultanément différent de 1;
  - rti est constraint d'exister dans R1 et R2;
  - un identifiant définie sur rti devra
    - avoir R1 et R2 comme uniques composants;
    - n'exister que si la connectivité maximale de A dans R1 et la connectivité maximale de B dans R2 sont différents de 1;
  - s'il existe un accès de A vers rti dans R1 (resp. de B vers rti dans R2), alors il existe un accès de rti vers B dans R2 (resp. de rti vers A dans R1) et réciproquement.
- rti n'est pas à considérer comme un PATH-TYPE récursif ou multitype;
- pas d'identifiants/clé d'accès définis sur A ayant R1 comme composant (pour la justification : cfr. niveau théorique;)
- pas d'identifiants/clé d'accès définis sur B ayant R2 comme composant (pour la justification : cfr. niveau théorique;)
- R n'existe pas.

2.6 ELIMINATION DE LA REURSIVITE.

Remarques.

1. Nous supposons que les PATH-TYPEs récursifs à transformer sont non multitypes.

2. Dans le cas de PATH-TYPEs récursifs, un membre est identifié par son rôle et non par le RECORD-TYPE membre.

ETUDE DES DIFFERENTS TYPES DE TRANSFORMATIONS OFFERTS A L'UTILISATEUR POUR ELIMINER LA RECURSIVITE.

2.6.1 Transformation d'un PATH-TYPE récursif en un RECORD-TYPE.

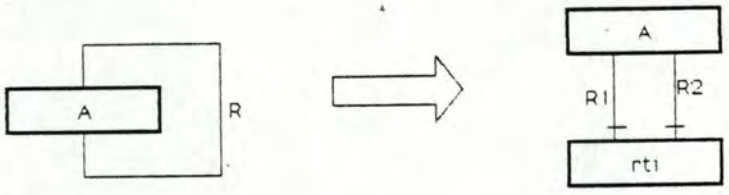


Figure 2.5: Transformation d'un PATH-TYPE récursif en un RECORD-TYPE.

Mapping du point de vue théorique.

Introduction d'un RECORD-TYPE intermédiaire dans un PATH-TYPE récursif.

Opérations.

Cfr transformation d'un PATH-TYPE R en un RECORD-TYPE dans le cas où R est non récursif tout en notant que les membres A et B de R seront identifiés non pas par le nom du RECORD-TYPE membre mais par leur ROLE.

Préconditions.

Cfr transformation d'un PATH-TYPE R en un RECORD-TYPE mais R est récursif.

2.6.2 Elimination d'un PATH-TYPE récursif par duplication d'ITEMs.

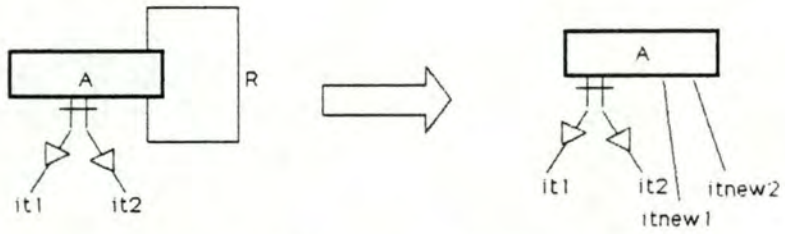


Figure 2.6: Elimination d'un PATH-TYPE récursif par duplication d'ITEMs.

Mapping du point de vue théorique.

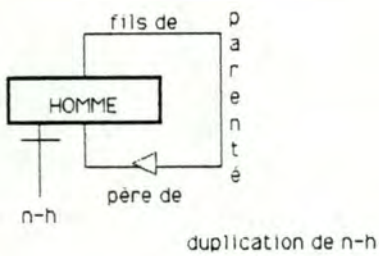
Rotation d'un PATH-TYPE récursif en utilisant un identifiant qui sert de base à la rotation n'ayant comme composants que des ITEMs.

Remarques.

1. Les opérations à effectuer sont similaires à celles dans le cas de l'élimination d'un PATH-TYPE non récursif mais dans un soucis de compréhension par l'utilisateur, ces opérations sont reformulées pour tenir compte du fait que le RECORD-TYPE A est égal au RECORD-TYPE B et que par conséquent, le sens de transfert des ITEMs ne peut pas être connu de la même façon. Il faut cependant également noter la différence que les nouveaux ITEMs devront avoir des noms différents des ITEMs dont ils sont la "copie".

2. Nous supposons que nous connaissions le ROLE dans le PATH-TYPE récursif R qui sera représenté par les nouveaux ITEMs. Ceci oblige à définir ce que signifie "les nouveaux ITEMs représenteront un ROLE déterminé d'un PATH-TYPE à éliminer".

   Considérons le PATH-TYPE récursif suivant :



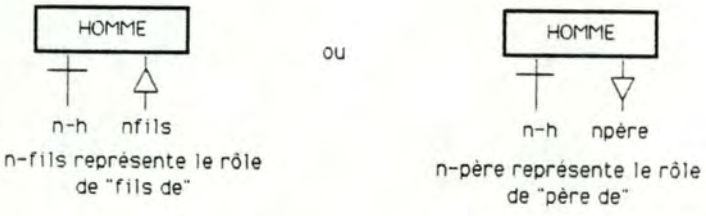


Figure 2.7: Exemple 1 de path-type récursif.

Soient R1, le ROLE de R représenté par les nouveaux ITEMs et
R2, le ROLE de R non représenté par les nouveaux ITEMs (peut être connu à partir de R1).

La connaissance de R1 est équivalente à la connaissance du sens de transfert dans le cas de PATH-TYPEs non récursifs et servira entre autres lors de l'étude des propriétés des nouveaux ITEMs.

3. Nous supposons également connus :

   - l'ensemble des ITEMs iti à "recopier" dans le RECORD-TYPE A sur lequel porte le PATH-TYPE récursif R et
   - les noms des ITEMs "copiés" new-iti et
   - le couplage des iti aux new-iti.

Opérations.

1. Création d'un éventuel identifiant technique défini sur A formé de l'ensemble des ITEMs iti.

   Remarque : cfr la remarque faite dans le cas de l'élimination d'un PATH-TYPE non récursif par duplication d'ITEMs.

2. Pour chaque ITEM iti, le recopier dans le RECORD-TYPE A mais en forçant son nom à la valeur donnée au new-iti correspondant.

   Remarques : cfr les remarques faites dans le cas de l'élimination d'un PATH-TYPE non récursif par

duplication d'ITEMs.

3. Etude des propriétés des new-iti.

Caractère obligatoire ou non de new-iti.

new-iti sera obligatoire si la connectivité minimale du roole R2 est égale à 1;
il sera facultatif sinon.

Remarque : les new-iti seront soit tous obligatoires, soit tous facultatifs.

Caractère répétitif ou non de new-iti.

Si la connectivité maximale de R2 est égale à 1 :
tous les new-iti seront non répétitifs;
ils seront tous de répétitivité variable illimitée sinon.

Caractère identifiant ou non de new-iti.

Comme dans le cas de l'élimination de PATH-TYPEs non récursifs, new-iti sera identifiant s'il est le seul ITEM créé (condition nécessaire mais non suffisante). La création de l'identifiant formé de new-iti sera dès lors prise en charge par l'opération suivante.

4. Création d'un identifiant défini sur A ayant l'ensemble des ITEMs new-iti comme uniques composants.

Un tel identifiant ne sera créé que si la connectivité maximale de R1 dans R est égale à 1.

5. Etude de l'ensemble des new-iti comme clé d'accès dans A.

Une telle clé d'accès ne sera créée que si R2 était accessible à partir de R1 dans R.

Remarque : cfr la remarque faite dans le cas de l'élimination d'un PATH-TYPE non récursif par duplication d'ITEMs.

6. Traitement des clés d'accès/identifiants ayant les ROLEs R1 ou R2 comme composants.

Les préconditions assurent qu'aucune clé d'accès/identifiant n'aura le ROLE R1 comme composant.
Les clés d'accès/identifiants ayant le ROLE R2 comme composant seront traduits en remplaçant ce composant par l'ensemble des ITEMs new-iti.

7. Etude des contraintes d'intégrité supplémentaires.

1. Pour chaque ITEM copié, création de la contrainte d'intégrité suivante : l'ensemble des valeurs prises par l'ITEM new-iti doit être inclu dans l'ensemble des valeurs prises par l'ITEM iti dont il est la copie.

   Remarque : le signe d'inclusion est remplacé par le signe d'égalité dans le cas où la connectivité minimale du ROLE R2 est égale à 1.

2. Création de la contrainte d'intégrité spécifiant que tous les RECORDs A ayant une valeur pour l'un des ITEMs new-iti ont également une valeur pour les autres ITEMs new-iti. Cette contrainte d'intégrité ne sera créée que si la connectivité minimale du ROLE R2 n'était pas égale à 1 (puisque dans ce cas, tous les new-iti seront facultatifs).

8. Suppression du PATH-TYPE R ainsi que toutes ses liaisons avec d'autres objets (RECORD-TYPE, IKO, contraintes d'intégrité).

Préconditions.

1. Le PATH-TYPE R n'est pas multitype.

2. Si tous les ITEMs iti existent dans le RECORD-TYPE A sur lequel porte R, il existe un identifiant ayant l'ensemble complet de ces ITEMs comme uniques composants.

3. Aucun des ITEMs new-iti n'existe dans A.

4. Aucun des iti n'est répétitif.

5. Aucun des descendants des iti n'est relié à un compteur.

6. Il existe une contrainte d'intégrité spécifiant que tout RECORD A jouant le ROLE R1 a une valeur pour chaque ITEM iti.

7. Le ROLE R1 ne peut être composant d'un identifiant/clé d'accès.

   Pour la justification de cette précondition, veuillez consulter l'étude théorique.

8. Si le nombre d'ITEMs new-iti à créer est plus grand que un, la connectivité minimale du ROLE R2 doit être égale à un.

2.6.3 Remarques.

1. L'élimination d'un PATH-TYPE récursif par "liaison de l'une de ses extrémités à l'identifiant de l'autre extrémité" comme dans le cas de PATH-TYPEs non récursifs, ne sera pas analysée. Il existe de toutes façons suffisamment de moyens d'éliminer la récursivité et par conséquent, l'utilité de ce type de transformation se situe essentiellement au niveau des accès.

2. Les types d'associations suivants pourraient également être offerts à l'utilisateur comme solutions d'élimination de la récursivité :

1. transformation du PATH-TYPE récursif en un RECORD-TYPE suivie d'une duplication d'ITEMs pour éliminer un des PATH-TYPEs né de la transformation précédente.
En prenant comme exemple le PATH-TYPE récursif PARENTE, ce type de transfomations permettrait à l'utilisateur l'obtenir directement une des solutions suivantes :

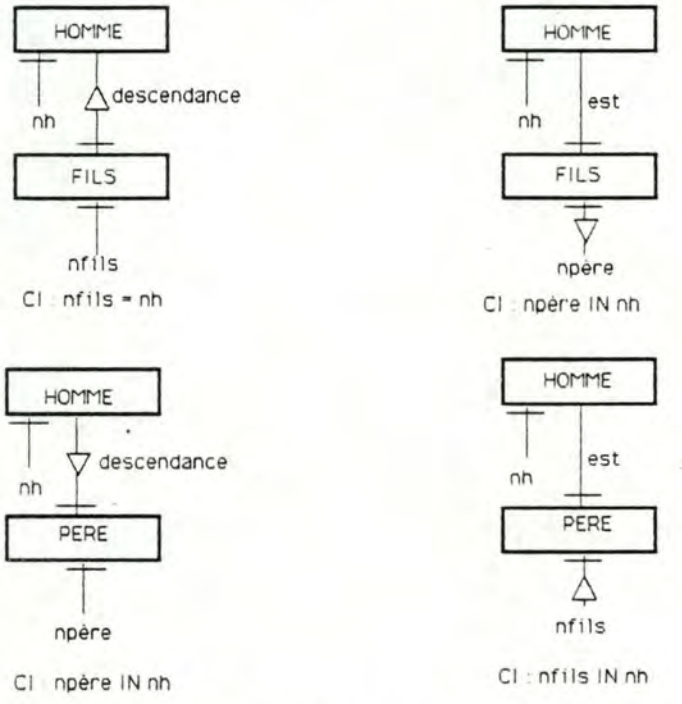


Figure 2.8: Exemple 2 de path-type récursif.

2. transformation du PATH-TYPE récursif en un RECORD-TYPE suivie d'une duplication d'ITEMs pour éliminer les deux PATH-TYPEs créés lors de la transformation précédente.
En prenant comme exemple le PATH-TYPE récursif PARENTE, ce type de transformation permettrait à l'utilisateur d'obtenir directement l'une des solutions suivantes :

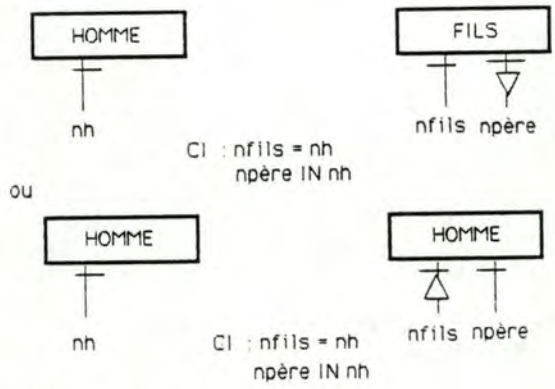


Figure 2.9: Exemple 3 de path-type récursif.

N.B.
Dans un premier temps d'analyse, ces solutions d'élimination de PATH-TYPEs récursifs ne seront pas offertes de façon directe à l'utilisateur puisque celui-ci peut les obtenir au moyen d'autres transformations qui elles seront offertes.
De plus, certaines de ces solutions peuvent introduire de la redondance. Par exemple dans la solution (1) le RECORD-TYPE HOMME est devenu redondant.
En effet, la contrainte d'existence sur le ROLE fils du PATH-TYPE initial PARENTE implique que tout HOMME est un FILS.
Par conséquent, l'utilisateur pourrait préférer la solution où le RECORD-TYPE HOMME a disparu du schéma.
Cette dernière solution peut être obtenu de la même façon que la solution précédente, mais l'élimination du PATH-TYPE "est" né de la transformation du PATH-TYPE PARENTE en le RECORD-TYPE FILS par duplication d'ITEMs sera remplacée par une fusion des RECORD-TYPEs HOMME et FILS. Cette dernière transformation peut être demandée par l'utilisateur puisque HOMME et FILS peuvent être considérés comme équivalents du point de vue sémantique.

3. Transfomations de reconstitution du PATH-TYPE récursif.

De telles transformations ont essentiellement un but de correction, non dans un désir de conformité à un SGBD (il existe heureusement aucun SGBD exigeant que certains types de PATH-TYPEs sont récursifs), mais en raison d'un backtracking dans les décisions de l'utilisateur.
C'est pourquoi ces transformations seront dès lors considérées comme secondaires et ne seront pas analysées.

2.7 ECLATEMENT DE PATH-TYPES MULTITYPES.

Remarques.

1. Les solutions possibles d'éclatement de PATH-TYPE multitypes peuvent différer à la fois en fonction du but visé dans la demande de cette transformation (élimination ou simplement transformation des PATH-TYPEs multitypes) et en fonction du nombre d'extrémités multitypes dans le PATH-TYPE à transformer.
   De plus, dans les cas où les deux extrémités sont multitypes, l'utilisateur peut vouloir éliminer le multitype soit dans les deux extrémités soit dans uniquement l'une des extrémités.
   Ces constatations permettent de faire ressortir la difficulté dans le choix d'une orientation de l'utilisation vers la solution désirée.

2. Tout identifiant/clé d'accès ayant comme composant le PATH-TYPE multitype sera ignoré (cfr. limite du transformateur : ignorance de la notion d'identifiant/clé d'accès et d'ordre globaux).

2.7.1 UNE SEULE DES EXTREMITES EST MULTITYPES.

Conventions de dénommination.

R : le PATH-TYPE multitype à éclater
A : l'extrémité multitype de R
B = l'extrémité non multitype de R (le RECORD-TYPE de l'extrémité non multitype de R)
A1, A2, ... An : les RECORD-TYPEs membres de R faisant partie de A

2.7.1.1 ELIMINATION D'UNE EXTREMITE MULTITYPE.

La seule solution possible analysée et offerte est la suivante : éclatement du PATH-TYPE multitype en autant de PATH-TYPEs qu'il y a de membres dans l'extrémité multimembre.

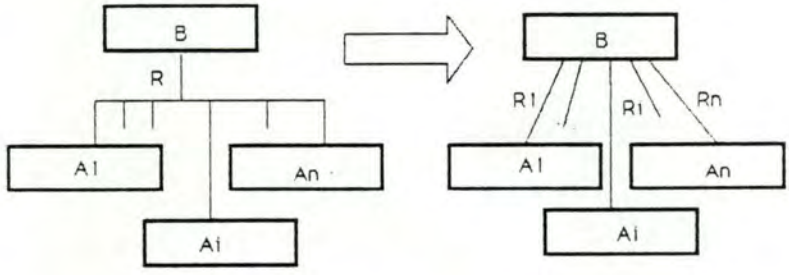


Figure 2.10: Elimination d'une extremité multitype.

Mapping du point de vue théorique.

Eclatement(s) en deux d'une extrémité d'un PATH-TYPE multitype.

Opérations.

1. Pour chaque membre Ai de l'extrémité multitype de R :

   - crétion d'un PATH-TYPE Ri ayant Ai et B pour membres;

   - étude des propriétés du PATH-TYPE Ri :

     1. Rooles dans Ri :
        les rooles de Ai et B dans Ri seront respectivement les mêmes que les rooles de Ai et B dans R.

     2. Connectivités minimales dans Ri :
        les connectivités minimales de Ai et B dans Ri seront respectivement les mêmes que les connectivités minimales de Ai et B dans R.
        Les préconditions assurent que la connectivité minimale de B dans R est égale à None.

     3. Connectivités maximales dans Ri :
        cfr. les rooles dans Ri.

     4. Etude des accès dans Ri :
        il existera un accès de Ai vers B (resp. de B vers Ai) dans Ri si et seulement si il existe un accès de Ai vers B (resp. de B vers A) dans R.

2. Suppression du PATH-TYPE R et de toutes ses liaisons avec d'autres objets (RECORD-TYPE, PATH-TYPE, contraintes d'intégrité ...)

Préconditions.

- le PATH-TYPE R existe et a une et une seule extrémité multitype;
- aucune des PATH-TYPEs Ri qui remplace R n'existe;
- le membre B de l'extrémité non multitype de R a une connectivité minimale égale à None.
  Cfr. remarques d'introduction à l'éclatement de PATH-TYPE multitypes ou cfr. l'étude des familles théoriques.

2.7.1.2 TRANSFORMATION DU MULTITYPE.

La solutions qui sera proposée à l'utilisateur est l'éclatement du PATH-TYPE multitype en un nombre quelconque de PATH-TYPEs, pourvu que ce nombre ne soit pas strictement supérieur au nombre de membres de l'extrémité multitype.

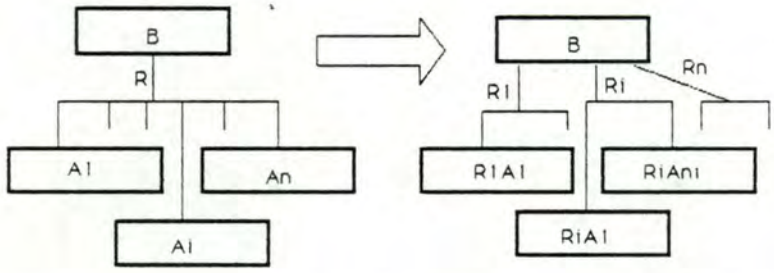


Figure 2.11: Transformation du multitype.

Mapping du point de vue théorique.

Eclatement(s) en deux d'une extrémité d'un PATH-TYPE multitype.

Opérations.

1. Création des PATH-TYPEs Ri qui remplacent le PATH-TYPE R.

   Les extrémités de Ri seront constituées de la manière suivante :
   - une des extrémités aura B comme unique membre et
   - l'autre extrémité sera composée d'un sous-ensemble RiAi des membres de l'extrémité A de R.

2. Etude des propriétés des PATH-TYPEs Ri.

   Pour chaque membre de Ri son roole, sa connectivité minimale et maximale, de même que son roole d'accès seront les mêmes que le membre correspondant dans R.

3. Suppression du PATH-TYPE R et de toutes ses liaisons avec d'autres objets (RECORD-TYPE, PATH-TYPE, contraintes d'intégrité ...)

Préconditions.

Cfr. les proconditions dans le cas de l'élimination du multitype. Cependant, certaines préconditions devront être ajoutées. Cet ajout est dû au fait que le PATH-TYPE R peut être éclaté en un nombre quelconque de PATH-TYPEs.
Dans un but de simplifier l'expression de ces préconditions, nous supposons que Ri-A représente le sous-ensemble des membres de l'extrémité multitype de R qui deviendront par la transformation membre du nouveau PATH-TYPE Ri.

Ces préconditions sont les suivantes :

- le nombre de PATH-TYPEs Ri qui replaceront le PATH-TYPE R doit être inférieur ou égal au nombre de RECORD-TYPE membres de l'extrémité multitype de R;
- $\bigcup_{i=1}^{n} Ri\text{-}A$ = l'ensemble des A1, A2, ... An
- Ri-A inter Rj-A = l'ensemble vide pour tout i et pour tout j de 1 à n (si i différent de j).

2.7.2 LES DEUX EXTREMITES SONT MULTITYPE.

2.7.2.1 ELIMINATION DU MULTITYPE DANS LES DEUX EXTREMITES.

La transformation d'éclatement du PATH-TYPE multitype en un nombre de PATH-TYPEs égal au produit du nombre de membres dans l'une des extrémités et du nombre de membres dans l'autre extrémité ne sera pas analysée.
Si l'utilisateur désire cependant éliminer le multitypage et que pour certains PATH-TYPEs, les deux extrémités sont multitypes, il devra effectuer tout d'abord une transformation d'élimination du multitypage dans l'une des extrémités dans le cas où les deux extrémités sont multitypes et ensuite, éliminer le multitype des PATH-TYPEs résultant (qui auront une et une seule extrémité multitype).

2.7.2.2 ELIMINATION DU MULTITYPE DANS L'UNE DES EXTREMITES.

Conventions de dénommination.

R : le PATH-TYPE multitype à éclater
A et B les extrémités de R
A1, A2, ... An : les RECORD-TYPEs membres de R faisant partie de A
B1, B2, ... Bm : les RECORD-TYPEs membres de R faisant partie de B

Nous supposons que A est l'extrémité multitype à éclater.

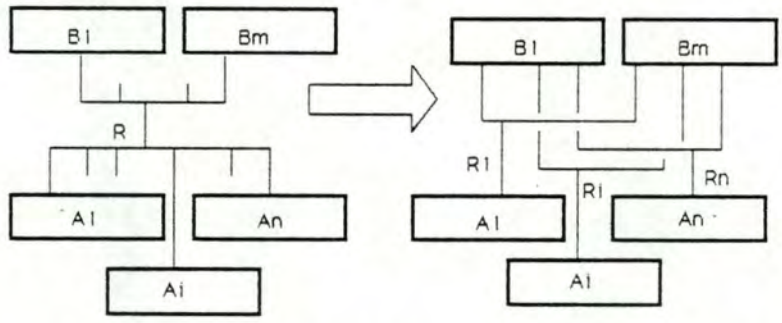


Figure 2.12: Elimination du multitype dans l'une des extrémites.

Dans le cas où un PATH-TYPE R a ses deux extrémités multitypes et que l'utilisateur désire éliminer le multitype uniquement dans une des deux extrémités, les deux types de transformations suivants lui sont proposées :

2.7.2.2.1 ECLATEMENT D'UNE EXTREMITE MULTITYPE EN AUTANT DE PATH-TYPES QU'IL Y A DE MEMBRES DANS CETTE EXTREMITE.

Mapping du point de vue théorique.

Eclatement(s) en deux d'une extrémité d'un PATH-TYPE multitype.

Opérations.

Cfr. l'élimination d'une extrémité multitype dans le cas où une seule des extrémités est multitype, mais tenir compte du fait que l'extrémité B est composée de plusieurs membres et que par conséquent, dans les nouveaux PATH-TYPEs Ri, le membre B sera remplacé par l'ensemble des membres B1, B2, ... Bn faisant partie de B.

Préconditions.

- le PATH-TYPE R existe et a ses deux extrémités multitypes,
- aucun des PATH-TYPEs Ri qui remplaceront le PATH-TYPE R n'existe,
- tous les RECORD-TYPEs B1, B2, ... Bn, membres de R faisont partie de l'extrémité non à éclater, ont une connectivité minimale égale à None.
  cfr. remarques d'introduction à l'éclatement des PATH-TYPEs multitypes ou cfr. l'étude des familles théoriques.

2.7.2.2.2 TRANSFORMATION D'UN PATH-TYPE MULTITYPE EN UN RECORD-TYPE.

Remarques.

1. La transformation d'un PATH-TYPE multitype R en un RECORD-TYPE exige d'imposer :
   - soit que tous les RECORD-TYPEs membres faisant partie d'une même extrémité ont même connectivité maximale. Dans ce cas, toutes les solutions de traduction des connectivités maximales possibles dans le cas de PATH-TYPEs non multitypes peuvent être proposées ici puisqu'il est permis de parler de la connectivité maximale d'une extrémité;
   - soit que, tout RECORD-TYPE membre de R aura dans le nouveau PATH-TYPE le reliant au RECORD-TYPE traduisant R, une connectivité maximale égale à celle qu'il avait dans R.

   Dans un premier temps d'analyse, ces deux contraintes seront imposées.
   Cependant, celles-ci n'auront pas à être reprises au niveau des préconditions puisque :
   - le modèle MAG lui-même impose déjà la première contrainte et
   - la deuxième contrainte sera imposée par le déroulement de la transformation elle-même étant donné que le choix de la solution de traduction des connectivités maximales prendra en compte cette contrainte.

2. Cette transformation n'est pas proposée dans le cas où une seule des extrémités est multitype puisque dans ce cas, l'origine de la transformation ne serait pas une inconformité SGBD, mais il s'agirait d'une correction à effectuer au niveau conceptuel.

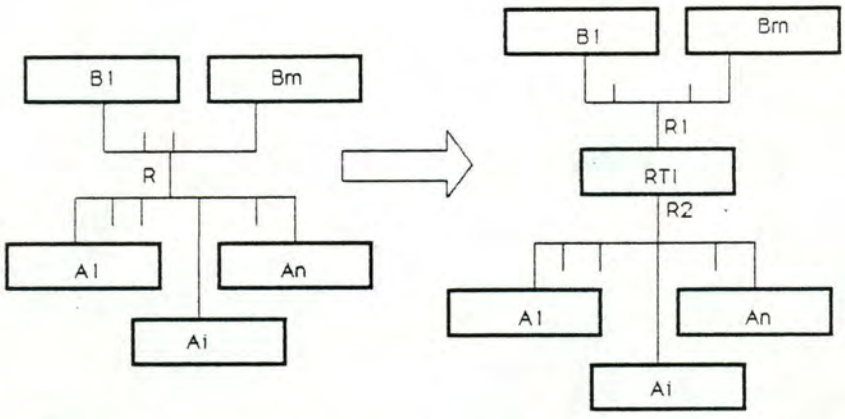


Figure 2.13: Transformation d'un path-type multitype en un record-type.

Mapping du point de vue théorique.

Introduction d'un RECORD-TYPE intermédiaire dans un PATH-TYPE.

Opérations.

1. Création du RECORD-TYPE rti.

2. Création du PATH-TYPE R1 reliant rti à l'extrémité A de R.

3. Création du PATH-TYPE R1 reliant rti à l'extrémité B de R.

4. Etude des connectivités minimales/maximales dans R1 et R2.

   rti aura des connectivités minimales et maximales égales à One dans R1 et R2.
   Les connectivités minimales et maximales des membres de R1 et R2, autre que rti, seront les mêmes que celles qu'il avait dans R.

5. Etude des accès dans R1 et R2.

   S'il existe un accès de l'extrémité A (resp. B) vers l'extrémité B (resp. A) dans R, alors il existeront un accès de l'extrémité A (resp. B) vers rti dans R1 (resp. R2) et un accès de rti vers l'extrémité B (resp. A) dans R2 (resp. R1).

6. Suppression du PATH-TYPE R.

Préconditions.

- le RECORD-TYPE rti n'exite pas;
- le PATH-TYPE R existe et a ses deux extrémités multitypes;
- les PATH-TYPEs R1 et R2 n'existent pas.

2.8 CREATION D'UN PATH-TYPE DANS LEQUEL LE RECORD-TYPE SYSTEM EST MEMBRE.

Mapping du point de vue théorique.

Création d'un PATH-TYPE dans lequel le RECORD-TYPE SYSTEM est membre.

Opérations.

1. Création du PATH-TYPE R entre le RECORD-TYPE rt donné et le RECORD-TYPE SYSTEM.

Remarques :

1. Le RECORD-TYPE SYSTEM est supposé exister et avoir été créé dès que l'utilisateur a spécifié qu'il désirait une conformité CODASYL.
2. R sera un PATH-TYPE 1-N du RECORD-TYPE SYSTEM vers le RECORD-TYPE rt avec contrainte d'existence sur la cible et il sera accessible dans les deux sens.

Préconditions.

- les RECORD-TYPE SYSTEMs et rt existent;
- le PATH-TYPE R n'existe pas;
- il n'existe pas de PATH-TYPE entre les RECORD-TYPE SYSTEM et rt;
- rt n'est pas le RECORD-TYPE SYSTEM.

2.9 SUPPRESSION D'UN PATH-TYPE AYANT LE RECORD-TYPE SYSTEM COMME MEMBRE.

Mapping du point de vue théorique.

Suppression d'un PATH-TYPE ayant le RECORD-TYPE SYSTEM comme membre.

Opérations.

1. Suppression du PATH-TYPE R donné et de toutes ses liaisons avec d'autres objets (RECORD-TYPE, contraintes d'intégrité, IKO, ...).

Préconditions.

- Le PATH-TYPE R a le PATH-TYPE SYSTEM comme membre;
- il n'existe aucune clé d'accès/identifiant ayant R comme composant.

3 Les RECORD-TYPEs.

3.1 ECLATEMENT D'UN RECORD-TYPE EN DEUX RECORD-TYPEs DE MEME SEMANTIQUE.

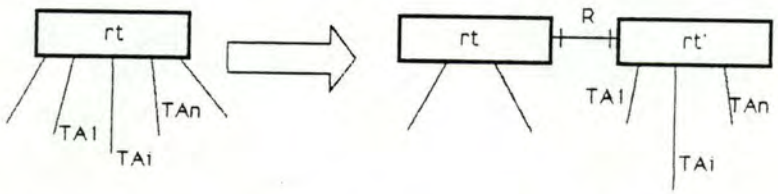


Figure 3.1: Eclatement d'un RECORD-TYPE en deux.

Soit TA l'ensemble des types d'associations TA1, ... TAn à transférer de rt vers rt'

Mapping du point de vue théorique.

Eclatement d'un record-type en deux record-types de même sémantique.

Opérations.

1. Création du record-type rt'.

2. Création du path-type R entre rt et rt' tel que :

   les rôles = les rôles donnés par l'utilisateur,
   les connectivités minimales de rt et de rt' dans R = One,
   les connectivités maximales de rt et de rt' dans R = One.

3. Pour chaque TAi :

   si TAi est un ITEM : transfert de TAi de rt vers rt',
   si TAi est un path-type : suppression du membre rt de TAi et
   création du membre rt' de TAi.

   Remarque : aucune propriété de TAi n'est modifiée lors du transfert.

4. Etude des accès : si l'utilisateur désire l'équivalence du point de vue des accès, R sera bidirectionnel.

5. Etude des IKOs : si un des TAi à transférer est composant d'un IKO, alors l'IKO concerné sera retiré du record-type rt et relié au record-type rt'
   (les préconditions assurent que les autres TAi sont également composants).

Préconditions.

- le record-type rt existe dans le schéma;
- le path-type R n'existe pas dans le schéma;
- TA n'est pas vide;
- pour chaque TAi :
  - soit TAi est un item dans rt,
  - soit TAi est un path-type ayant rt comme membre;
  et TAi est différent de R.
- pour chaque TAi :
  si TAi est un composant d'un identifiant et/ou d'une clé d'ordre défini sur rt (ou d'une clé d'acce si l'utilisateur désire l'équivalence d'accès), alors tous les autres composants sont également transférés de rt vers rt';
- il existe au moins un type d'associations portant sur rt non contenu dans TA (sauf R).

3.2 TRANSFERT DE TYPES D'ASSOCIATIONS DEFINIS SUR UN RECORD-TYPE VERS SA DUPLICATION SEMANTIQUE.

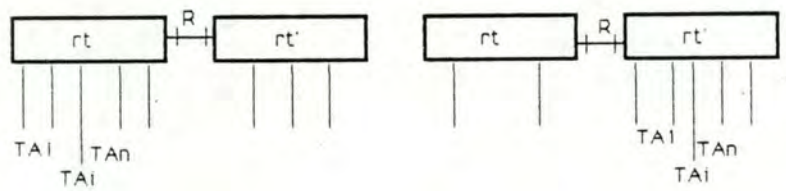


Figure 3.2: Transfert de types d'associations entre RECORD-TYPEs.

Soit TA l'ensemble des types d'associations TA1, ... TAn à transférer de rt vers rt',
rt le record-type origine du tranfert et
rt' le record-type sur lequel seront définis les types d'associations après le transfert.

Mapping du point de vue théorique.

Transfert de types d'associations définis sur un record-type vers sa duplication sémantique.

Opérations.

1. Pour chaque TAi :

   si TAi est un ITEM : transfert de TAi de rt vers rt',
   si TAi est un path-type : suppression du membre rt de TAi et
   création du membre rt' de TAi.

   Remarque : aucune propriété de TAi n'est modifiée lors du transfert.

2. Etude des IKOs : si un des TAi à transférer est composant d'un IKO, alors l'IKO concerné sera retiré du record-type rt et relié au record-type rt'
   (les préconditions assurent que les autres TAi sont également composants).

3. Si TA est égal à l'ensemble des types d'associations portant sur rt, (sauf R),
   alors : - suppression du path-type R entre rt et rt' et
   - suppression du record-type rt.

Préconditions.

- le record-type rt existe dans le schéma;
- le record-type rt' existe dans le schéma;
- il existe un path-type R entre rt et rt' tel que :
  la connectivité minimale de rt et de rt' dans R = One et
  la connectivité maximale de rt et de rt' dans R = One;
- TA n'est pas vide;
- pour chaque TAi :
  - soit TAi est un item dans rt,
  - soit TAi est un path-type ayant rt comme membre;
  et TAi est différent de R.
- pour chaque TAi :
  si TAi est un composant d'un identifiant et/ou d'une clé d'ordre défini sur rt (ou d'une clé d'acce si l'utilisateur désire l'équivalence d'accès), alors tous les autres composants sont également transférés de rt vers rt'.

4 Les FICHIERs.

4.1 CREATION D'UN FICHIER.

Mapping du point de vue théorique.

Création d'un fichier.

Opération.

Création du fichier fi.

Précondition.

- le fichier à créer n'existe pas.

4.2 SUPPRESSION D'UN FICHIER.

Mapping du point de vue théorique.

Suppression d'un fichier.

Opération.

Suppression du fichier fi.

Précondition.

- le fichier à supprimer existe.

4.3 INSERTION D'UN RECORD-TYPE DANS UN FICHIER.

Mapping du point de vue théorique.

Insertion d'un record dans un fichier.

Opération.

Insertion du record-type rt dans le fichier fi.

Précondition.

- le record-type rt existe;
- le fichier fi existe;
- rt n'est pas contenu dans fi.

4.4 RETRAIT D'UN RECORD-TYPE D'UN FICHIER.

Mapping du point de vue théorique.

Retrait d'un record d'un fichier.

Opération.

Retrait du record-type rt du fichier fi.

Précondition.

- le record-type rt existe;
- le fichier fi existe;
- rt est contenu dans fi.

5 Les IKOs.

5.1 CREATION D'UN IDENTIFIANT TECHNIQUE.

Mapping du point de vue théorique.

Création d'un identifiant technique.

Opérations.

S'il n'existe aucun IKO iko ayant l'ensemble compl, ... compn comme uniques composants
alors :

1. Création d'un IKO iko identifiant.
2. Rattachement de iko au record-type rt.
3. Rattachement des composants compl, ... compn à iko.
4. Si le référentiel est 'fichier' :
   création d'une contrainte d'intégrité spécifiant quel fichier est le référentiel pour iko.

sinon :

1. Modification du type de iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il n'existe aucun IKO iko identifiant ayant l'ensemble complet compl, ... compn comme composants (uniques ou non) de référentiel donné;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.2 SUPPRESSION D'UN IDENTIFIANT TECHNIQUE.

Mapping du point de vue théorique.

Suppression d'un identifiant technique.

Opérations.

Si le type de l'IKO iko = 'I' (c'est-à-dire si iko n'est pas également clé d'accès et/ou clé d'ordre)
alors :

1. Suppression de toutes les liaisons entre iko et les composants compi.
2. Suppression de iko.
3. Suppression des éventuelles contraintes d'intégrité à iko.

sinon :

1. Modification du type de iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il n'existe aucun IKO iko identifiant ayant l'ensemble complet compl, ... compn comme uniques composants de référentiel donné;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.3 CREATION D'UNE CLE D'ACCES ET/OU D'ORDRE.

Mapping du point de vue théorique.

Création d'une clé d'accès d'ordre.

Opérations.

S'il n'existe aucun IKO iko ayant l'ensemble compl, ... compn comme uniques composants
alors :

1. Création d'un IKO iko clé d'accès et/ou d'ordre.
2. Rattachement de iko au record-type rt.
3. Rattachement des composants compl, ... compn à iko.
4. Si le référentiel est 'fichier' :

   création d'une contrainte d'intégrité spécifiant quel fichier est le référentiel pour iko.

sinon :

1. Modification du type de iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il n'existe aucun IKO iko clé c'accès et/ou d'ordre ayant l'ensemble complet compl, ... compn comme composants, de référentiel donné, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre.
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.4 SUPPRESSION D'UNE CLE D'ACCES ET/OU D'ORDRE.

Mapping du point de vue théorique.

Suppression d'une clé d'accès et/ou d'ordre.

Opérations.

Si le type de l'IKO iko = 'C' et/ou 'O'
alors :

1. Suppression de toutes les liaisons entre iko et les composants compi.
2. Suppression de iko.
3. Suppression des éventuelles contraintes d'intégrité reliées à iko.

sinon :

1. Modification du type de iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il n'existe aucun IKO iko clé d'accès et/ou d'ordre ayant l'ensemble complet compl, ... compn comme composants, de référentiel donné, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre.
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.5 AJOUT D'UN COMPOSANT A UN IDENTIFIANT - CLE D'ACCES - CLE D'ORDRE.

Mapping du point de vue théorique.

Ajout d'un composant item/path-type à un identifiant technique, une clé d'accès et/ou d'ordre.

Opérations.

Ajout du composant comp à l'IKO iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il existe un IKO iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- comp est un item dans rt ou un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j) et comp est différent de compi.

5.6 SUPRESSION D'UN COMPOSANT D'UN IDENTIFIANT - CLE D'ACCES - CLE D'ORDRE.

Mapping du point de vue théorique.

Suppression d'un composant item/path-type à un identifiant technique, une clé d'accès et/ou d'ordre.

Opérations.

Suppression du composant comp à l'IKO iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il existe un IKO iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- il existe un i tel que compi = comp;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j) er
  comp est différent de compi.

5.7 MODIFICATION DU REFERENTIEL D'UN IDENTIFIANT - CLE D'ACCES - CLE D'ORDRE.

5.7.1 Insertion d'un identifiant - clé d'accès - clé d'ordre dans un path-type.

Mapping du point de vue théorique.

Ajout d'un composant path-type à un identifiant technique, une clé d'accès et/ou d'ordre.

Opérations.

Ajout du composant comp à l'IKO iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il existe un IKO iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko : compi est un item dans rt
- comp est un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.7.2 Retrait d'un identifiant - clé d'accès - clé d'ordre d'un path-type.

Mapping du point de vue théorique.

Suppression d'un composant path-type à un identifiant technique, une clé d'accès ou d'ordre.

Opérations.

Suppression du composant comp à l'IKO iko.

Préconditions.

- le record-type rt existe dans le schéma;
- il existe un IKO iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- il existe un i tel que compi = comp;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.7.3 Définition d'un fichier comme référentiel d'un IKO.

Mapping du point de vue théorique.

Création d'une contrainte d'intégrité spécifiant qu'un fichier est le référentiel d'un IKO.

Opérations.

1. Création d'une contrainte d'intégrité spécifiant qu'un fichier est le référentiel de iko.

2. Relier la contrainte d'intégrité créée à iko et au fichier fi.

Préconditions.

- le record-type rt existe dans le schéma;
- le fichier fi existe;
- rt est contenu dans fi;
- il existe un IKO iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel BD, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko : compi est un item dans rt

- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.7.4 Suppression de la définition d'un fichier comme référentiel d'un IKO.

Mapping du point de vue théorique.

Suppression d'une contrainte d'intégrité spécifiant qu'un fichier est le référentiel d'un IKO.

Opérations.

1. Suppression de la liaison entre iko et la contrainte d'intégrité spécifiant que le référentiel de iko est le fichier fi.

2. Suppression de la contrainte d'intégrité spécifiant que le fichier fi est le référentiel de iko.

Préconditions.

- le record-type rt existe dans le schéma;
- le fichier fi existe;
- rt est contenu dans fi;
- il existe un IKO iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel fi, et, dans le cas d'une clé d'ordre, ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko : compi est un item dans rt
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.8 MODIFICATION DE LA LOI D'ORDRE D'UNE CLE D'ORDRE.

Mapping du point de vue théorique.

Changement de la valeur d'une propriété d'un objet dont l'effet n'entraine aucune modification sémantique.

Opérations.

Modification de la loi d'ordre de iko.

Préconditions.

- le record-type rt existe dans le schéma;
- rt est contenu dans fi;
- il existe un IKO clé d'ordre iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné et ayant les mêmes lois d'ordre et numéros de séquence que celles donnés;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.9 MODIFICATION DE LA SEQUENCE DES COMPOSANTS D'UNE CLE D'ORDRE.

5.9.1 Inversion de l'emplacement de deux composants.

Mapping du point de vue théorique.

Inversion de deux objets dans un ensemble ordonné.

Opérations.

Inversion du numéro de séquence de deux composants compl et compk.

Préconditions.

- le record-type rt existe dans le schéma;
- rt est contenu dans fi;
- il existe un IKO clé d'ordre iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné et ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

5.9.2 Déplacement d'un composant à un endroit déterminé dans sa séquence.

Mapping du point de vue théorique.

Une ou plusierus inversions de deux objets dans un ensemble ordonné.

Opérations.

Soit comp le jème composant de iko et devient le ième composant par cette transformation.

Si déplacement vers la gauche :
augmentation de 1 du numéro de séquence des composants dont le numéro est compris entre j et i (i compris).

Si déplacement vers la droite :
diminution de 1 du numéro de séquence des composants dont le numéro est compris entre i et j (j compris).

Préconditions.

- le record-type rt existe dans le schéma;
- rt est contenu dans fi;
- il existe un IKO clé d'ordre iko défini sur rt ayant l'ensemble complet compl, ... compn comme uniques composants, de référentiel donné et ayant les mêmes lois d'ordre et numéros de séquence que ceux donnés
- iko a au moins i composants;
- pour chaque composant compi d'iko :
  - soit compi est un item dans rt
  - soit compi est un path-type ayant rt comme membre;
- pour tout i, pour tout j : compi est différent de compj (si i est différent de j).

ANNEXE 3

ALGORITHME D'UN SENS DE PARCOURS DU VERIFICATEUR DE CONFORMITE

Remarque : (1) = COBOL
(2) = CODASYL
(3) = SQL

pour chaque RECORD-TYPE rt de la base de données

pour chaque IKO iko défini sur rt

"pas de clé d'ordre qui ne soit pas clé d'accès" (3)
"pas d'identifiant qui ne soit pas clé d'accès" (1 3)
"les seuls ordres possibles si la clé d'accès n'est pas clé d'accès sont l'ordre FIFO et l'ordre aléatoire et, si la clé d'ordre est une clé d'accès, seul l'ordre trié par valeur croissante de la clé de tri est permis." (1)
"si une ou plusieurs clés d'accès sont définies sur un record-type, une au moins est identifiante" (1)
"pas d'identifiant ayant plusieurs composants path-type" (2)
"toute clé d'ordre est définie dans un path-type 1-N" (2)
"si un identifiant n'est pas clé d'accès, alors il est défini dans un type de chemins 1-N" (2)
"une clé d'accès est constitué d'un seul item" (1)
"pas plus d'un identifiant par record-type dans un fichier" (2)
"pas plus d'une clé d'accès par record-type dans un fichier" (2)
"dans un même record-type deux clés d'accès ne peuvent pas commencer à la même position" (1)
"pas de clé d'accès répétitive" (1 2)
"pas de clé d'accès ayant un ancêtre répétitif" (1)

pour chaque ITEM it dans rt

(a) test sur le niveau de décomposition de it (1 2 3)
test sur le caractère facultatif de it (1 3)
test sur la répétitivité de it
- "pas d'item répétitif" (3)
- "pas d'item répétitif ayant un certain nombre d'ancêtre répétitif" (1)
- "pas d'item de répétitivité variable ayant un ancêtre répétitif" (1)
- "pas d'item de répétitivité variable qui ne soit pas le dernier dans le graphe de décomposition" (1)
- "pas d'item de répétitivité variable sans item compteur" (1)

si it est élémentaire :
alors test sur son format (1 2 3)
sinon pour chaque item fils de it ----> (a)

test sur l'unicité du nom de it (1 2 3)
test sur la formatation du nom de it (1 2 3)

```
      test sur le nombre d'items dans rt                (1)


      pour chaque PATH-TYPE pt dont rt est membre       (2)

           si pt n'a pas encore une entrée dans la table des path-types
            alors mémoriser le nom et le code interne dans la table des
            path-types
           mémoriser le nom du rôle de rt dans pt,
                  les connectivités minimales et maximales de rt dans pt
                  le degré d'accessibilité de rt dans pt


           test d'unicité des noms des path-types

      test "un path-type ne peut être le support de plusieurs clés d'ordre" (2)
      test "aucun IKO de référentiel path-type ne peut être
      défini sur le record-type origine"                (2)



      pour chaque FICHIER dans lequel rt est contenu
           ...
      test sur le nombre de fichier(s) dans le(s)quel(s) rt peut être contenu (1)

pour chaque entrée dans la table des path-types         (2)

     vérifier toutes les contraintes portant sur la notion de path-type
     test sur la formatation du nom du path-type


pour chaque FICHIER fi de la base de données            (1)

     pour chaque RECORD-TYPE rt contenu dans fi
          ...

     test sur le nombre de record-types dans fi
```