MASTER EN SCIENCES INFORMATIQUES

Étude des structures sémantiques communes aux systèmes IDS et IMS. Indépendance des programmes par rapport à ces systèmes

Bastin, Félix

Award date:
1975

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

Année académique 1974-1975

ÉTUDE DES STRUCTURES SÉMANTIQUES COMMUNES AUX SYSTÈMES IDS ET IMS. INDÉPENDANCE DES PROGRAMMES PAR RAPPORT A CES SYSTÈMES

Félix BASTIN

MEMOIRE PRESENTE EN VUE
DE L'OBTENTION DU GRADE
DE LICENCIE ET MAITRE EN
INFORMATIQUE

Nous adressons notre vive reconnaissance à Monsieur le professeur H. Leroy qui a bien voulu accepter d'être notre directeur de mémoire, à Messieurs J.L. Hainaut et B. Lecharlier dont les précieux conseils nous ont permis de mener à bien ce mémoire.

Au terme de ce travail, qu'il nous soit permis de remercier Monsieur Bodart, directeur, et les professeurs de la Faculté d'Informatique de l'Université Notre-Dame de la Paix de Namur, pour la formation qu'ils nous ont permis d'acquérir ; ainsi que Monsieur Lemaire, directeur de l'informatique de la société UCB, pour la compréhension qu'il nous a témoignée durant nos études.

TABLE DES MATIERES

* * * * *

1. INTRODUCTION

1.1. Présentation du sujet de l'étude

A l'heure actuelle, l'utilisateur envisageant l'installation d'une base de données doit savoir qu'il peut se trouver facilement prisonnier d'un système d'implémentation de base de données, non seulement du fait des problèmes de réécriture des programmes d'application que poserait le passage d'un système d'implémentation à un autre, mais également, du fait des différences logiques de ces systèmes en ce qui concerne l'expression des structures sémantiques d'information.
L'étude proposée devrait mener à la définition de structures sémantiques communes aux systèmes IDS et IMS, ainsi qu'à la définition d'un langage de manipulation commun et susceptible de réaliser l'indépendance des programmes par rapport au système d'implémentation.

1.2. Présentation de la démarche suivie pour atteindre les buts proposés.

Nous avons suivi la démarche utilisée pour résoudre nombre de problèmes de l'informatique et qui va du réel perçu par l'utilisateur vers l'implémentation avec retour vers cet utilisateur pour lui présenter le modèle implémenté et les contraintes de cette implémentation.

Sur base de cette démarche, nous présenterons sous la forme d'un modèle que nous avons appelé modèle relationnel

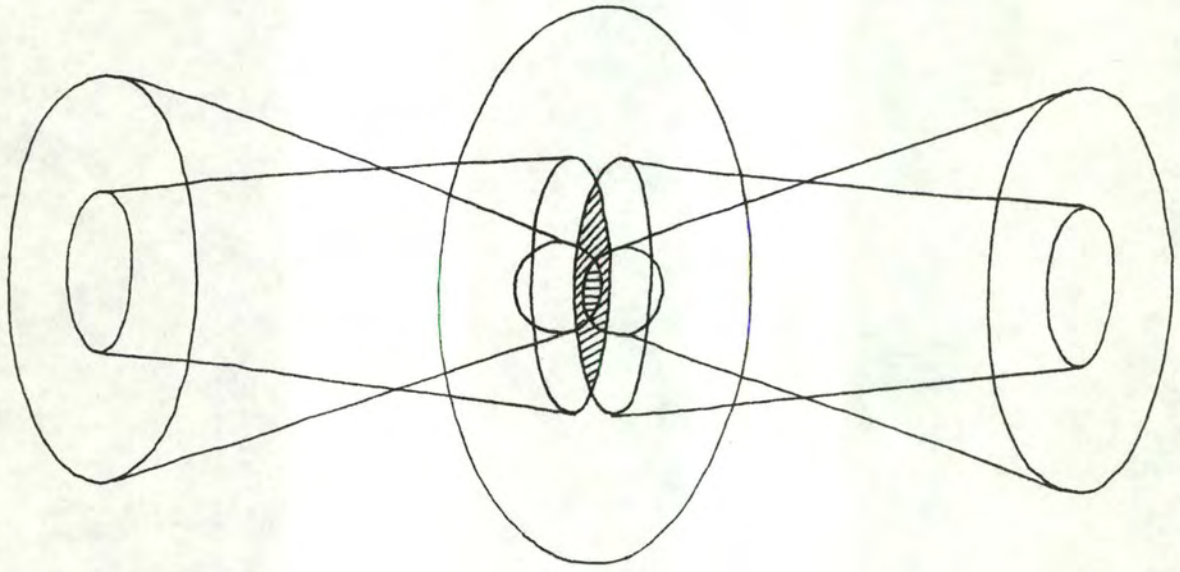
la perception qu'un utilisateur a de son environnement base de données. Celui-ci percevant et manipulant un ensemble d'unités d'information structuré par des relations d'association, le modèle relationnel, qui sera un sous-ensemble du modèle fonctionnel développé à l'institut d'informatique des facultés universitaires de Namur, donnera une définition et une typologie de ces éléments.

Pour permettre l'implémentation des structures dégagées par le modèle relationnel, nous présenterons un modèle appelé modèle d'accès général qui s'emboitera au modèle relationnel et qui fournira des outils de conception relatif à cette implémentation.

La troisième partie de l'étude représentera l'implémentation d'un sous-ensemble du modèle d'accès général par le système IDS et l'implémentation d'un sous-ensemble du modèle d'accès général par le système IMS. Dans cette troisième partie, nous aurions pu uniquement considérer la projection des deux systèmes IDS et IMS sur le modèle d'accès général, mais l'ensemble commun (intersection) de ces deux projections risquait d'être insuffisante pour permettre le développement d'un système de gestion de base de données encore utilisable.
Pour l'agrandir, nous avons préféré réaliser l'implémentation de deux sous-ensembles du modèle d'accès général.
L'ensemble intersection de ces deux sous-ensembles implémentés que nous appellerons modèle d'accès réduit sera présenté dans la partie suivante de l'étude, associé à un langage de manipulation de données.
Cette association représentera un système de gestion de base de données implémentable par les systèmes IDS et IMS.

Le modèle d'accès réduit contiendra les structures sémantiques communes à ces systèmes et le langage de manipulation assurera l'indépendance des programmes par rapport au système d'implémentation.

Graphiquement nous représenterons cette demarche par :



Système IDS

Système IMS

Modèle d'accès général

La partie hachurée de lignes horizontales représente l'ensemble intersection des projections des deux systèmes sur le modèle d'accès.
La partie hachurée de lignes obliques représente l'ensemble intersection des sous-ensembles implémentés l'un par le système IDS, l'autre par le système IMS.

Avant de conclure, nous examinerons rapidement l'aide que peut apporter l'étude entreprise à la résolution du problème de la conversion d'une base de données implémentée par un système à la même base de données implémentée par l'autre système.

* * * * *

2. PRESENTATION D'UN MODELE RELATIONNEL

2.1. Introduction

Rappelons que, comme annoncé au chapître précédent, le modèle relationnel que nous présentons, est un sous-ensemble d'un modèle relationnel plus général appelé "modèle fonctionnel" développé à l'institut d'informatique des facultés universitaires "Notre Dame de la Paix" à Namur.

Le modèle relationnel doit permettre une représentation structurée du réel perçu par l'utilisateur ; il comprendra donc la définition des unités d'information, des relations d'association entre celles-ci et une typologie de ces relations suivant leur contenu sémantique.

Bien que les unités d'information et les relations d'association soient interdépendantes, nous présentons une définition des unités d'information indépendante des relations d'association en ne laissant apparaître cette interdépendance que plus tard dans l'étude, au niveau de la présentation du modèle d'accès.

2.2. Les unités d'information

2.2.1. Une unité d'information est un ensemble de données élémentaires qui définissent une même entité, un même individu.

Le terme "entité", qui couvre la signification des mots - chose - concept - être - évènement..., représente sémantiquement l'élément fondamental de la transmission de

l'information entre utilisateurs.

Exemple : - L'entité "Tiers" représente un être qui sera client ou fournisseur
- Les entités "Produit" et "Magasin" représentent des objets
- L'entité "Centre de frais" représente un concept
- L'entité "Mouvement de stock" représente un évènement.

Chaque entité se caractérise par un ensemble de propriétés ou attributs dont la réalisation est l'ensemble des données élémentaires de l'unité d'information.

Une réalisation d'une entité est la valorisation des propriétés la caractérisant et représente une unité d'information.
Remarquons que nous pouvons avoir plusieurs réalisations d'une entité ayant les mêmes valeurs de propriétés, c'est-à-dire que la valorisation des propriétés d'une entité n'est pas nécessairement identifiante pour les réalisations de cette entité.

Exemple : Considérons l'entité "Client". Un exemple de valorisation sera :

Entité Client

Propriétés	Valorisation
- Numéro de Tiers	N 725063815
- Nom	HENRION
- Premier prénom	GUY
- Numéro Compte bancaire	210-29580-41
- Code rating	1
- Plafond de crédit	99999999
- Condition de paiement	30 jours fin de mois

L'ensemble des propriétés d'une entité est fonction du problème posé.
A un même objet peuvent être associées plusieurs entités suivant le point de vue que l'utilisateur a de cet objet.

Considérons deux utilisateurs, un gestionnaire des stocks et un vendeur, un même produit ne représente pas, pour chacun d'eux, un même ensemble de propriétés.

La perception que le vendeur a de l'entité "Produit" pourra être l'ensemble des propriétés suivantes :

- Code produit
- Nom du produit
- Prix de vente en gros
- Prix de vente au détail
- Délai de livraison
- Pourcentage commission

Alors que l'ensemble des propriétés de l'entité "Produit" serait pour le gestionnaire :

- Code produit
- Nom du produit
- Délai de fabrication
- Quantité en stock
- Quantité réservée
- Lot économique de commande
- Point de commande

2.2.2. Par entité, nous pouvons avoir trois types de propriétés :

- Le premier type est celui des propriétés identifiantes. Une propriété est dite identifiante pour une entité si ses

valeurs sont telles qu'elles établissent une correspondance biunivoque entre chaque valeur et chaque réalisation de l'entité c'est-à-dire pour chaque valeur de cette propriété, nous aurons au plus une réalisation de l'entité.

Exemple : Si nous reprenons l'exemple précédent, parmi les propriétés de l'entité "Produit", le code produit est une propriété identifiante.

- Le deuxième type est celui des propriétés d'ordre.
Une propriété est dite propriété d'ordre si la relation qui existe entre ses valeurs et les réalisations de l'entité permet de définir sur l'ensemble de ces réalisations une structure d'ordre.

Exemple : Pour l'entité "Produit", le nom du produit peut être considéré comme une propriété d'ordre.

Une propriété identifiante est une propriété d'ordre, mais la réciproque est fausse sauf si l'ordre établi est un ordre strict.

- Le troisième type est celui qui regroupe toutes les autres propriétés, celles qui n'ont pas été définies comme étant identifiantes ou d'ordre.

Exemple : Les propriétés suivantes de l'entité "Produit" sont des propriétés du troisième type :
- Délai de fabrication
- Quantité en stock
- Quantité réservée
- .....

Les deux premiers types de propriétés possèdent un contenu sémantique au niveau du modèle relationnel, les propriétés du troisième type n'auront pour le modèle aucun contenu sémantique et ne prendront de signification qu'au niveau de l'utilisateur.

2.2.3. Entités ou propriétés

Il n'existe pas de règles strictes permettant de déterminer si une donnée élémentaire doit être considérée comme une réalisation d'entité ou comme une valorisation de propriété d'entité, c'est-à-dire que si un ensemble de propriétés définit une entité, la distinction entre entités et propriétés est difficile à réaliser.

Nous donnerons plus loin, au niveau de la définition du modèle d'accès, la méthode utilisée par le service informatique de la société "Union Chimique Belge", pour constituer l'ensemble des entités à partir de l'ensemble des propriétés.

2.2.4. Synthèse

Une unité d'information est définie par le triplet :

- Entité
- Propriétés (attributs ou caractéristiques) de cette entité
- Valeurs prises par les propriétés.

2.3. Association sémantique entre les unités d'information

2.3.1. Introduction

Une entité correspondant à une classe homogène d'information est associée à un sous-ensemble de l'ensemble des unités d'information formant la base des données.

Si pour une entité notée A, nous écrivons $\mathcal{C}_A$ le sous-ensemble associé, ce sous-ensemble est par définition l'ensemble de toutes les réalisations de l'entité A.

Une association entre deux entités A et B est une relation qui permet de lier sémantiquement les réalisations de ces deux entités, c'est donc une relation définie sur les ensembles $\mathcal{C}_A$ et $\mathcal{C}_B$.

Mais, dans une base de données, ces ensembles de réalisation étant essentiellement dynamique, il est difficile de donner une définition mathématique des relations et d'en réaliser une typologie, les ensembles considérés en mathématique étant statiques.

L'étude des relations s'éffectuera donc en deux phases. Dans la première phase qui sera la phase descriptive, nous considérerons une base de données idéales possédant des réalisations de toutes les entités. Cette base de données est statique et nous étudierons les relations que nous pouvons définir entre les réalisations des entités. Dans la deuxième phase, nous verrons l'incidence de la typologie, établie dans la phase un, sur la dynamique de la base de données.

2.3.2. Définition des relations d'association

Rappelons la définition mathématique d'une relation, Soient les ensembles X_1, X_2, ..., X_n non nécéssairement distincts, R est une relation sur ces n ensembles si elle est un ensemble de n-tuples dont chacun est de la forme $(x_1, x_2,..,x_n)$ avec $x_1 \in X_1$, $x_2 \in X_2$,...,$x_n \in X_n$

R est donc un sous-ensemble du produit cartérien $X_1 \times X_2 \times .. \times X_n$ et est dite de degré n.
Une relation de degré 2 est appelée une relation binaire, dans la suite, nous ne considérerons plus que des relations binaires.
Tout sous-ensemble G de $X_1 \times X_2 \times \ldots \times X_n$ est une relation à savoir la relation $(x_1, x_2,\ldots,x_n) \in G$.

Nous dirons qu'il existe une relation d'association entre deux entités A et B pouvant ne pas être distinctes, s'il existe une relation R définie sur les ensembles $\mathcal{C}_A$ et $\mathcal{C}_B$.
Nous écrirons

$$R(x,y) \in \mathcal{C}_A \times \mathcal{C}_B \text{ avec } \begin{cases} x \in \mathcal{C}_A \\ y \in \mathcal{C}_B \end{cases}$$

A la proposition "l'entité A est associée à l'entité B" correspondra l'ensemble Yx des éléments $y \in \mathcal{C}_B$ tels que R (x,y) soit vraie pour $x \in \mathcal{C}_A$
et à la proposition "l'entité B est associée à l'entité A" correspondra l'ensemble Xy des éléments $x \in \mathcal{C}_A$ tels que R (x,y) soit vraie pour $y \in \mathcal{C}_B$.
L'ensemble Yx est l'image de la réalisation x de l'entité A dans l'ensemble des réalisations de l'entité B par la relation R.
L'ensemble Xy est l'image réciproque de la réalisation y de l'entité B dans l'ensemble des réalisations de l'entité A par la relation R.

L'association sémantique représentée par la relation R sera caractérisée par les valeurs maximum et minimum du cardinal des ensembles Xy et Yx pour tout $x \in \mathcal{C}_A$ et tout $y \in \mathcal{C}_B$.
Nous obtenons deux typologies des relations d'association entre entités suivant que l'on considère la valeur maximum ou la valeur minimum du cardinal de Xy et du cardinal Yx.
Remarquons que, dans le contexte de la base de données idéale , les ensembles considérés sont des ensembles non vides.

2.3.3. Typologie suivant le cardinal maximum

C'est une typologie suivant le nombre maximum de réalisations de chaque entité intervenant dans la relation d'association.

2.3.3.1. Relation "One to One" ou (1 à 1)

Nous dirons qu'il existe une relation d'association "One to One" entre deux entités A et B si
- le cardinal maximum de Yx = 1
- le cardinal maximum de Xy = 1

C'est-à-dire si à une réalisation de l'entité A correspond une et une seule réalisation de l'entité B et réciproquement.

Exemple : Considérons les entités "Clients" et "Fournisseurs"
Un client peut être fournisseur
Un fournisseur peut être client
Dans une application Compte Client, nous pouvons établir une relation "One to One" entre ces deux entités.

2.3.3.2. Relation "One to Many" ou (1 à n)

Nous dirons qu'il existe une relation d'association "One to Many" entre deux entités A et B si
- Le cardinal maximum de Yx $\in [o, \infty]$
- Le cardinal maximum de Xy est 1

C'est-à-dire si à une réalisation a de l'entité A correspond n (compris entre o et ∞) réalisations b_i de l'entité B et si à une réalisation b_i pour i $\in [1,n]$ correspond une et une seule réalisation a de l'entité A.

Exemple : Considérons les deux entités "Société" et "Usine" la relation d'association que nous pouvons établir entre elles est une relation "One to Many".
A une société peut correspondre une ou plusieurs usines, à une usine ne peut correspondre qu'une et une seule société.

2.3.3.3. Relation "Many to One" ou (n à 1)

Cette relation peut être considérée comme inverse de la relation "One to Many".

2.3.3.4. Relation "Many to Many" ou (n à n)

Nous disons qu'il existe une relation d'association "Many to Many" entre les entités A et B si
- le cardinal maximum de Yx $\in [o,\infty]$
- le cardinal maximum de Xy $\in [o,\infty]$

C'est-à-dire si à une réalisation de A correspond n réalisations de B et si réciproquement à une réalisation de B correspondent n réalisations de A.

Exemple : L'association entre l'entité " Fournisseur" et l'entité "Article" est une relation "Many to Many". Un fournisseur peut livrer plusieurs articles. Un article peut être livré par plusieurs fournisseurs.

2.3.4. Typologie suivant le cardinal minimum

Rappelons que Yx est l'ensemble image de la réalisation $x \in \mathcal{C}_A$ de l'entité A dans l'ensemble $\mathcal{C}_B$ des réalisations de l'entité B pour la relation R et que Xy est l'ensemble image de la réalisation $y \in \mathcal{C}_B$ de l'entité B dans l'ensemble $\mathcal{C}_A$ des réalisations de l'entité A par la relation inverse R^{-1}.

Rappelons aussi que l'expression cardinal minimum signifie :
- le minimum des cardinaux des ensembles Yx pour $x \in \mathcal{C}_A$
- le minimum des cardinaux des ensembles Xy pour $y \in \mathcal{C}_B$

La typologie suivant le cardinal minimum est une typologie suivant les propriétés d'existence d'une des deux entités en relation par rapport à l'existence ou la non-existence de l'autre.
Le couple formé par le cardinal minimum des ensembles Yx et le cardinal minimum des ensembles Xy définit la force de la relation d'association existant entre les deux entités A et B.

2.3.4.1. Relation à membre obligatoire

Nous dirons que l'entité B est membre obligatoire de la relation d'association R existant entre les entités A et B si

- Le cardinal minimum de Yx = 0
- Le cardinal minimum de Xy = 1

C'est-à-dire si la réalisation d'une entité déclarée membre obligatoire de la relation d'association, n'a de sens que si elle est associée à au moins une réalisation de l'autre entité.

Exemples :-Considérons les deux entités "Commande" et "Poste de la commande", l'entité "Poste de la commande" peut être déclarée membre obligatoire de la relation unissant les deux entités.
Une réalisation de "Poste de la commande" n'a de signification qu'associée à une réalisation de l'entité "Commande".

-Soient les deux entités "Société" , "Usine" et la relation d'association "appartenance" définie entre ces deux entités ; cette relation est une relation à membre obligatoire, en effet :

- A une société appartient zéro, une ou plusieurs usines
- Une usine n'appartient qu'à une société et n'a de signification que reliée à une société.

2.3.4.2. Relation faible

Nous dirons que la relation d'association existante

entre les entités A et B est une relation faible si :
- Le cardinal minimum de Yx est supérieur ou égal à 0
- Le cardinal minimum de Xy est supérieur ou égal à 0
C'est-à-dire si l'existence des réalisations a_i de l'entité A est indépendante de l'existence des réalisations b_j de l'entité B et réciproquement.

Exemples :-Dans un environnement compte client, la relation d'association établie entre les deux entités "Clients" et "Fournisseurs" est une relation faible, en effet :
Un client peut exister sans être un fournisseur
Un fournisseur peut exister sans être un client
Si un client qui est fournisseur est supprimé en tant que client, il peut toujours exister en tant que fournisseur et inversément.

-Considérons l'entité "Article" et la relation d'association "Articles de remplacement" établie entre l'entité et elle-même. Cette relation est une relation faible, en effet:
Les articles peuvent exister sans qu'ils ne soient reliés à d'autres articles par la relation "Article de remplacement".

2.3.5. Les deux typologies que nous venons de voir seront réunies en une seule pour établir la typologie des relations d'association du modèle relationnel, et nous aurons :

- Relation One to One faible
  Exemple : La relation d'association entre les entités "Clients" et "Fournisseurs" déjà vue.

- Relation One to Many faible
  Exemple : La relation "Article de remplacement" précedemment définie, si nous considérons qu'un article ne peut être remplacé que par un seul autre article.

- Relation Many to Many faible
  Exemple : La relation d'association définie entre les entités "Fournisseur" et "Article".

- Relation One to One à membre obligatoire

- Relation One to Many à membre obligatoire
  Exemples :-La relation d'association définie entre les entités "Commandes" et "Poste de la commande"

  -La relation d'association définie entre les entités "Société" et "Usine".

- Relation Many to Many à membre obligatoire
  Exemple : Considérons la relation d'association définie entre les entités "Dépôt" et "Article", cette relation est une relation du type Many to Many à membre obligatoire, en effet :
  Un article peut être stocké dans plusieurs dépôts, et un dépôt contient plusieurs articles ; mais un article devra toujours être stocké dans au moins un dépôt donc l'entité "Article" est membre obligatoire de la relation.

2.3.6. Dynamique des relations d'association

Comme nous l'avons exprimé dans l'introduction de

ce paragraphe, toute la présentation des relations d'association a été réalisée à postériori sur une base de données statiques.
Mais une base de données étant essentiellement dynamique, les caractéristiques des relations d'association seront données à priori pour toutes les relations devant exister dans la base des données. Ces caractéristiques interviennent alors comme contraintes pour maintenir la cohérence de la base lors de son évolution.
A tout instant, (point de vue statique), les relations seront soit des ensembles vides, soit du type initialement défini, c'est-à-dire que lors d'une évolution normale de la base des données, aucune relation ne peut violer ses caractéristiques.

2.3.7. Ensemble ordonné des réalisations dans une relation d'association.

Nous avons vu que l'utilisateur peut définir dans l'ensemble des propriétés d'une entité des propriétés d'ordre. Il peut, de ce fait, vouloir réaliser une structure d'ordre sur l'ensemble Yx de toutes les réalisations b_i de l'entité B en relation d'association avec une réalisation a de l'entité A, sur base d'une ou de plusieurs propriétés d'ordre de cette entité B.
Dans ce cas nous dirons que nous associons une relation avec des propriétés d'ordre et nous appellerons cette relation une relation d'ordre.
A la définition d'une relation d'ordre, il faudra toujours préciser sur quelles propriétés cette relation s'appuie pour établir la structure d'ordre.
La signification des ensembles ordonnés de réalisations dans des relations d'association apparaîtra plus clairement au

niveau du modèle d'accès.

Exemple : Considérons les entités "Client" et "Facture" et la relation d'association reliant ces deux entités, l'utilisateur peut décider que toutes les factures d'un client soient classées suivant leur date d'échéance ou suivant leur numéro de facture. Nous associerons alors à la relation la propriété date d'échéance ou numéro de facture et l'ensemble image d'un client sur l'ensemble des factures par la relation d'association sera ordonné suivant les valeurs de la propriété associée à cette relation.

2.3.8. Association de propriétés à une relation

L'utilisateur peut vouloir donner des propriétés à une relation d'association entre deux entités A et B.
Par exemple, considérons les entités "fournisseur" et "article" ; l'entité "fournisseur" reprend les propriétés se rapportant uniquement à un fournisseur, par exemple :
- nom du fournisseur
- adresse
- condition de paiement
- .....

L'entité "Article" reprend les propriétés spécifiques d'un article :
- code article
- libellé article
- .....

Nous devons associer à la relation établie entre ces deux entités les propriétés suivantes :

- Prix d'achat
- Numéro de commande
- Délai de livraison
- Pourcentage de ristourne
- .....

Le modèle relationnel présenté permet l'affectation de propriétés en nombre quelconque à une relation d'association. Remarquons que cela est très important pour les relations d'association du type "Many to Many" car dans ce cas les propriétés associées à la relation ne peuvent pas être placées dans une des deux entités.

Dans les autres cas, ces propriétés peuvent être placées dans l'ensemble des propriétés d'une des deux entités.

Le choix de l'entité à laquelle nous donnerons ces propriétés sera fonction du type de la relation.

Pour les relations "One to One", nous choisirons indifféremment une des deux entités et pour les relations "One to Many" ou "Many to One", nous choisirons l'entité dont plusieurs réalisations peuvent être reliées à une réalisation de l'autre entité. Ce choix est fait pour que chaque réalisation de l'entité choisie corresponde à au plus une occurence de la relation.

2.3.9. La présentation des relations d'associations que nous venons de réaliser peut être généralisée.

Cette généralisation, qui est la forme reprise par le modèle fonctionnel dont notre modèle relationnel est un

sous-ensembles, n'est pas nécessaire pour la suite de l'étude, nous n'en réaliserons qu'un très rapide exposé :
Nous avons vu que nous pouvions définir une relation suivant le nombre maximum de réalisations de chaque entité au sein de la relation par un couple (j,l)
avec : - j = cardinal maximum des ensembles Yx
- l = cardinal maximum des ensembles Xy

Nous avons vu aussi que nous pouvions définir une relation suivant la force de la relation pour un couple (i,k)
avec : - i = cardinal minimum des ensembles Yx
- k = cardinal minimum des ensembles Xy

Une relation sera complètement définie par le quadruple (i,j,k,l) qui représentera toutes les caractéristiques de la relation.
Les caractéristiques de notre modèle relationnel s'exprimeraient alors pour :

- Relation One to One faible : (0,1 ; 0,1)
- Relation One to Many faible : (0,∞ ; 0,1)
- Relation Many to Many faible : (0,∞ ; 0,∞)
- Relation One to One à membre obligatoire: (0,1 ; 1,1)
- Relation One to Many à membre obligatoire : (0,∞ ; 1,1)
- Relation Many to Many à membre obligatoire : (0,∞ ; 1,∞)

2.4. Représentation graphique du modèle relationnel

Pour représenter les structures de données permises par le modèle relationnel, nous utiliserons le formalisme de Bachman dans lequel nous introduirons une légère modification.

La technique de représentation graphique se base sur trois symboles fondamentaux :
- le rectangle
- le cercle
- l'arc (non orienté)

Chaque rectangle, dans un graphe de structure de données, représente l'ensemble de toutes les réalisations d'une entité, nous l'appellerons du nom de cette entité et nous écrirons ce nom à l'intérieur du rectangle.

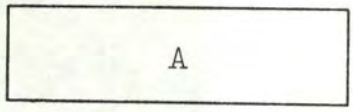
Chaque cercle représente l'ensemble de toutes les valorisations des propriétés attachées à une relation.

Chaque arc représente une relation entre les réalisations de deux entités, nous l'appellerons du nom de cette relation et nous écrirons ce nom le long de l'arc.

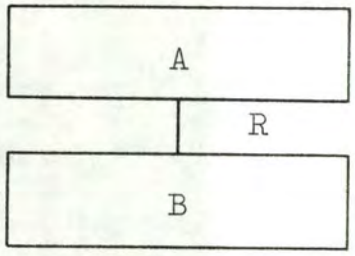
L'association logique de rectangles, de cercles, et d'arcs est la seule signification de ces symboles, c'est-à-dire que la manière dont ils sont combinés est seule significative, leur forme, dimension, placement est totalement dépourvu de contenu sémantique.

Les restrictions sur les possibilités de combinaison des symboles sont :
- Tout arc doit posséder un rectangle à chacune de ses extrêmités.
- Tout cercle ne peut être relié au'à un et un seul arc.

Nous pouvons donc avoir comme structures de base :

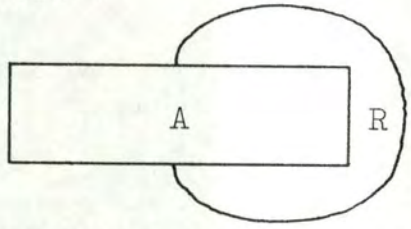


L'entité A n'intervient dans aucune relation



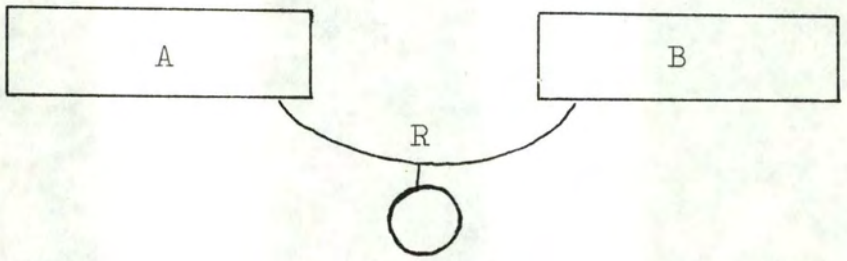
Les entités A et B sont associées par la relation R.

Les entités A et B peuvent ne pas être distinctes, nous avons alors :



Remarquons que dans ce cas, la relation d'association R ne peut pas être de type quelconque, en effet :
supposons que la relation R soit une relation à membre obligatoire ; à la création de la base de donnée, l'ensemble des réalisations de l'entité A est un ensemble vide, la création de la première réalisation de cette entité demande à être associée à une autre réalisation de cette entité de par la définition d'une relation à membre obligatoire, ce qui est contradictoire. Une telle relation d'association ne peut donc être qu'une relation faible.
Ce cas prouve que la dynamique de la base de données introduit certaines restrictions dans les possibilités de structuration du réel par le modèle relationnel.

Association de propriétés à une relation R



Tout graphe de structure de base de données est une combinaison de ces structures de base.

* * * * *

3. MODELE D'ACCES ASSOCIE AU MODELE RELATIONNEL

Si l'utilisateur, au travers des applications, perçoit des entités et des relations d'association entre celles-ci et si le modèle relationnel répond à ses besoins de structuration du réel, l'analyste et l'administrateur de la base de données voient les applications comme des algorithmes d'accès aux données.
Le modèle relationnel ne répondant pas à leurs besoins, il faut lui superposer un autre modèle qui prendra en considération les problèmes que posent les accès aux données.
Ce modèle d'accès doit s'emboîter au modèle relationnel et permettre à l'analyste et à l'administrateur de la base de concevoir l'implémentation des structures dégagées de ce dernier.
Le modèle d'accès permet donc la définition des entités et des relations d'accès entre ces entités.

Dans ce chapitre nous allons définir rapidement un modèle d'accès général se superposant au modèle relationnel pour, dans le chapitre suivant, en présenter les parties implémentables pour les systèmes IDS et IMS qui sont des modèles d'accès. Sur base de ces deux chapitres, nous définirons de façon plus détaillée, un modèle d'accès restreint au sous-ensemble du modèle d'accès implémentable par les deux systèmes à la fois, et nous définirons sur ce dernier un langage de manipulation des données.

3.1. Les entités

Rappelons qu'une entité correspond à un ensemble (classe homogène) d'information, que chaque élément de cet ensemble est une réalisation de l'entité et que le nom de l'entité référencie aussi cet ensemble.

3.1.1. Une entité sera définie par son nom, l'ensemble de ses propriétés et par l'ensemble des relations auxquelles elle participe.

Lors de la présentation du modèle relationnel nous n'avons pas définit les entités en y associant l'ensemble des relations et ce pour deux raisons :

- D'un point de vue logique, nous ne pouvions pas définir les entités en y associant les relations et les relations en y associant les entités.

- Dans le modèle relationnel, nous ne définissions que des relations sémantiques et nous ne nous posions pas le problème de l'accès à telle ou telle réalisation d'une entité.
  Le point de vue adopté dans le modèle d'accès étant différent nous devons associer à une entité, l'ensemble des relations auxquelles elle participe,comme nous allons le montrer par un exemple. En plus, nous lèverons l'ambiguité pouvant exister au niveau de la définition des relation d'accès en la déduisant de la définition des relations sémantiques.

Exemple :

Considérons les entité "Facture" et "Poste TVA" ayant respectivement comme propriétés :

- Numéro de facture
- Code TVA
- Base TVA
- Montant TVA

Nous voyons qu'une réalisation de l'entité "Poste TVA" ne peut être déterminée uniquement par la valorisation des propriétés de l'entité mais qu'il faut en plus y associer l'occurrence de la relation la reliant à l'entité "Facture" le problème étant d'accéder au poste TVA x de la facture y.

3.1.2. Les propriétés

Le modèle d'accès permet de décrire les propriétés identifiantes, les propriétés d'ordre et l'ensemble des autres propriétés. Ces dernières sont non significatives pour le modèle et n'y interviennent que comme sous-ensemble de l'ensemble des propriétés de l'entité ; l'utilisateur pourra les valoriser et les utiliser suivant les possibilités du langage employé pour écrire les programmes d'application.

Leurs valorisations n'interviendront jamais dans les procédures de contrôle et d'accès du modèle.

Nous allons sur l'exemple précédant préciser les notions de propriétés identifiantes et de propriété d'ordre.

Dans une société, la numérotation des factures doit être unique, c'est-à-dire que les valeurs de la propriété "Numéro de facture" déterminent biunivoquement les réalisations de l'entité "Facture" indépendamment des relations dans lesquelles cette entité peut intervenir. Cette propriété est donc une propriété identifiante de l'entité "Facture".

Par contre, la propriété "Code TVA" n'est pas telle que ses valeurs établissent une correspondance biunivoque entre chaque valeur et chaque réalisation de l'entité, même si cette correspondance existe pour tous les sous-ensembles $\mathcal{C}_A$ de réalisations en relation avec chaque réalisation a de l'entité "Facture".

Cette propriété est une propriété d'ordre dont l'ordre induit est strict sur les sous-ensembles $\mathcal{C}_A$ mais non sur l'ensemble de toutes les réalisations de l'entité, c'est-à-dire sur l'ensemble $\bigcup_a \mathcal{C}_A$
Dans ce cas, pour déterminer une réalisation de l'entité "Poste TVA", il faut donner une réalisation de l'entité "Facture" et une valeur à la propriété "Code TVA".
Nous appellerons cette propriété, une propriété localement identifiante.
Une propriété d'ordre dont l'ordre induit sur les sous-ensembles $\mathcal{C}_A$ n'est pas stricte est appelée une propriété d'ordre simple.

3.2. Les relations d'accès

3.2.1. Définition

Nous avons vu que toute relation d'association entre deux entités peut être représentée par une relation R et son

inverse R^{-1}. Au niveau du modèle d'accès, nous considérons ces deux relations comme indépendantes car si dans le modèle relationnel, l'inverse de toute relation existe toujours, ici, en fonction des applications et des algorithmes d'accès nécessaires, la relation inverse ne sera pas toujours implémentée. Nous décomposons donc la relation sémantique en une ou deux relations d'accès.

- L'entité A est associée à l'entité B, alors il existe une relation d'accès R définie sur les ensembles $\mathcal{C}_A$ et $\mathcal{C}_B$ des réalisations de l'entité A et de l'entité B telle que si (a,b) est une occurence de R, alors il existe un moyen d'accéder à la réalisation b de l'entité B à partir de la réalisation a de l'entité A.

- L'entité B est associée à l'entité A, alors il existe une relation d'accès R^{-1} définie sur les ensembles $\mathcal{C}_B$ et $\mathcal{C}_A$ des réalisations telle que si (b,a) est une occurence de R^{-1}, alors il existe un moyen d'accéder à la réalisation a de l'entité A à partir de la réalisation b de l'entité B.

R^{-1} est dite relation d'accès inverse de R car par définition, à tout instant, nous avons
si R (a) est l'image de $a \in \mathcal{C}_A$ dans $\mathcal{C}_B$ par R
si R^{-1} (b) est l'image de $b \in \mathcal{C}_B$ dans $\mathcal{C}_A$ par R^{-1}

$$b \in R(a) \Leftrightarrow a \in R^{-1}(b)$$

Par une abréviation du langage, nous dirons qu'une relation d'accès est définie sur les entités A et B et que A est l'entité source, B étant l'entité but de la réalisation.

3.2.2. Caractéristiques des relations d'accès

A tout instant, une relation d'accès R (A,B) sera une relation du type :
- One to One
- One to Many ou Many to One
- Many to Many

et du type :
- à Membre obligatoire
- faible

Le premier ensemble de caractéristiques donne les possibilités d'accès de la relation.
Le deuxième ensemble donne la dynamique des relations d'accès et des entités de la base de donnée.
Remarquons que si une relation d'accès R est une relation du type One to Many, la relation inverse R^{-1} est une relation du type Many to One.

3.2.2.1. Les possibilités d'accès

- Une relation "One to One" R (A,B) exprime qu'à tout instant :
  - R donne accès, à partir de toute réalisation a de A, à zéro ou une réalisation de B.
  - R permet d'accéder à toute réalisation b de B, à partir d'une et d'une seule réalisation a de A.

- Une relation "One to Many" R(A,B) exprime qu'à tout instant :
  - R donne accès, à partir de toute réalisation a de A, à n réalisations de B avec $n \in [0, \infty]$
  - R permet d'accéder à toute réalisation b de B, à partir de zéro ou d'une réalisation a de A.

- Une relation "Many to Many" R (A,B) exprime qu'à tout instant :
  - R donne accès, à partir de toute réalisation a de A,

à n réalisations de B avec $n \in [0, \infty]$

. R permet d'accéder à toute réalisation b de B, à partir de m réalisations de A avec $m \in [0, \infty]$

3.2.2.2. Dynamique de la base de données

- Une relation R (A,B) à membre obligatoire exprime que :
  - . La création d'une réalisation b de l'entité B exigera l'existence et l'identification préalable d'au moins une réalisation a de l'entité A à laquelle elle va être reliée.
  - . La création d'une réalisation b de l'entité B entraînera automatiquement la création de l'occurrence (a,b) de la relation R (A,B).
  - . La suppression de toutes les réalisations a_i de l'entité A auxquelles une réalisation b de l'entité B est reliée par R entraînera automatiquement la suppression de la réalisation b.

- Une relation (A,B) faible exprime que :
  - . La suppression ou la création d'une réalisation de l'entité A est indépendante de l'existence ou de la non-existence de toute réalisation b de l'entité B et réciproquement.
  - . La suppression ou la création d'une occurrence de la relation R peut s'effectuer indépendamment de la suppression ou de la création des réalisations d'entités intervenant dans cette occurrence.

Remarquons que la suppression d'une réalisation d'une entité entraîne toujours la suppression de toutes les occurrences de relations dans lesquelles elle intervient soit comme source, soit comme but.

3.2.3. Ordre induit par une relation d'accès

Par une relation d'accès R, nous définissons un moyen d'accéder à partir d'une réalisation a de l'entité A, à chaque réalisation b_i de l'entité but B appartenant à l'ensemble R (a) image de a dans l'ensemble $\mathcal{C}_B$ des réalisations de B par la relation R.

L'ordre induit par une relation d'accès sera l'ordre sous lequel on accède aux éléments de cet ensemble R (a). Cet ordre pourra être ascendant ou descendant sur les valeurs de une ou plusieurs propriétés d'ordre de l'entité B.
Il pourra être fonction de l'instant de création des réalisations $b_i \in R$ (a) par exemple : First in - First out
Last in - First out

Remarquons que nous appelons l'instant de création, l'instant physique de création qui dans le cas d'accès multiples à la base de données n'est pas l'instant logique de création.

Une présentation plus détaillée de l'ordre d'accès sera donnée plus loin au niveau de la présentation du modèle d'accès implémentable par les deux systèmes IDS et IMS.

3.3. Représentation graphique du modèle d'accès

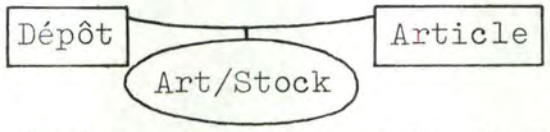
3.3.1. Comme dans le cas de la représentation graphique du modèle relationnel, la structure des informations, vue en terme d'entités et de relation d'accès, peut se représenter par un graphe orienté, dont les sommets sont les entités représentées sous forme de rectangle et dont les flèches (arc orienté) sont les relations d'accès.

Une flèche entre deux rectangles signifie qu'il existe un moyen permettant d'accéder à partir d'une réalisation de l'entité représentée par le rectangle source de la flèche, à n réalisations de l'entité représentée par le rectangle but de la flèche ; la valeur de n est fonction des caractéristiques de la relation.

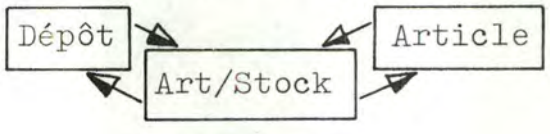
3.3.2. Pour passer du graphe du modèle relationnel au graphe du modèle d'accès, il faut :

- Remplacer les arcs par des flèches en les dédoublant si la relation inverse existe au niveau du modèle d'accès et donner un nom à cette relation inverse.
- Remplacer un cercle par un rectangle et tracer des flèches entre ce rectangle et les deux rectangles qui étaient reliés par l'arc portant le cercle, cet arc n'est pas remplacé par des flèches. Ceci exprime que toute relation d'association du type Many to Many (les propriétés associées à une relation ne sont nécessaires que pour ce type de relation) possédant des propriétés est transformée en une entité possédant ces propriétés et en deux relations "One to Many" reliant les deux entités, associées par la relation "Many to Many", à la nouvelle entité. Il faut aussi donner un nom à cette nouvelle entité.

  Exemple : La structure du modèle relationnel représentée ci-après :

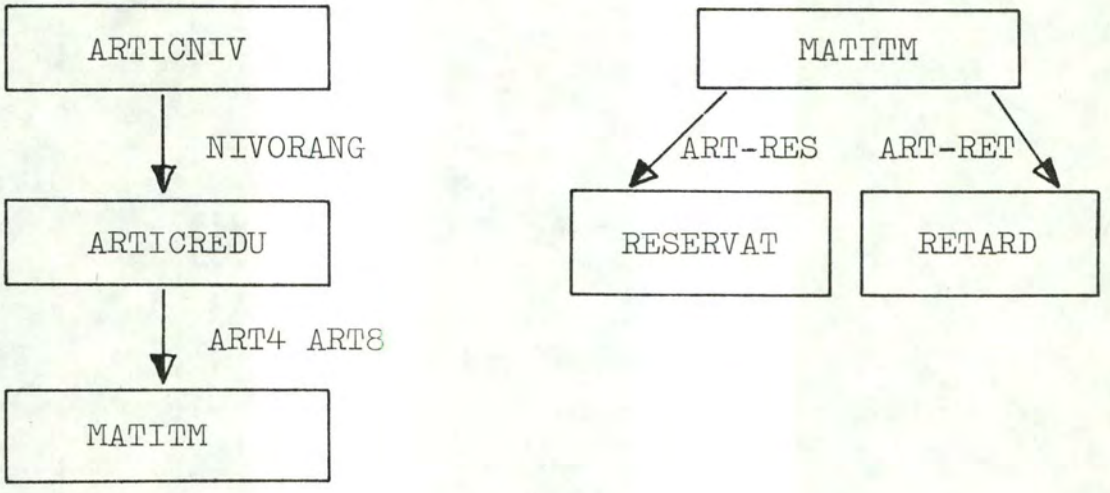


devient dans le graphe du modèle d'accès

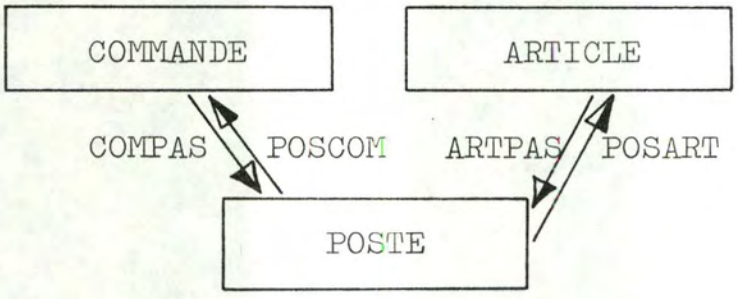


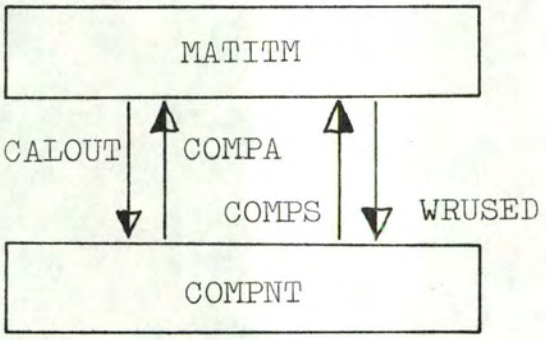
3.3.3. A partir des symboles de base qui sont le rectangle et la flèche nous présenterons quelques structures de base.

Structure hiérarchique

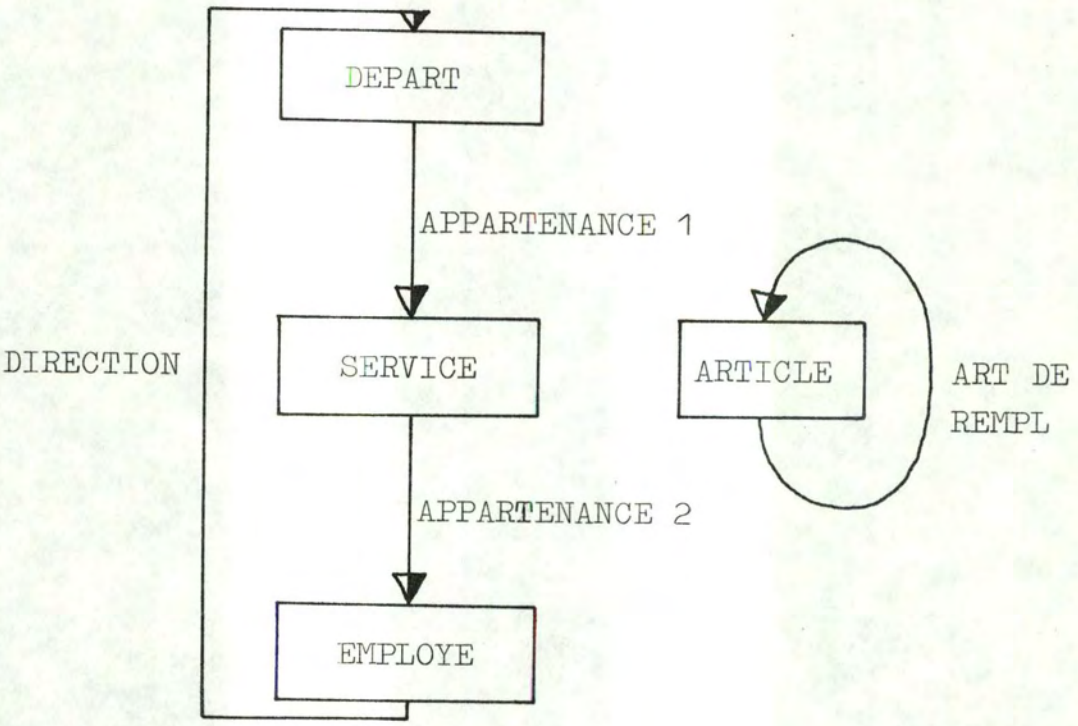


Structure en réseau sans cycle





Structure en réseau avec cycle



3.4. Entités ou propriétés

A la définition de la base de données, l'analyste et l'administrateur de celle-ci, éprouvent généralement des difficultés pour établir la distinction entre les propriétés qui peuvent être considérées comme des propriétés d'entité et celles qui peuvent être considérées comme des entités. Il n'existe pas de règles strictes permettant de réaliser cette distinction, nous présenterons donc la méthode mise au point par le service informatique de la société UCB. Cette méthode est heuristique.

3.4.1. Etablir une liste exhaustive de toutes les propriétés élémentaires (indécomposables), si la propriété considérée n'est pas élémentaire, il faut la décomposer en propriétés élémentaires et mettre la liste à jour.

Donner pour chaque propriété élémentaire :

- Un nom (symbole) que nous retrouverons au niveau des programmes d'application.
- Une description de la propriété
  - . son contenu
  - . sa classe et sa longueur - alphanumérique
    - numérique
  - . estimation du nombre de valeurs qu'elle peut prendre :
    - minimum
    - moyen
    - maximum

3.4.2. Répartir cette liste en deux classes de propriétés :

- Classe des propriétés "Indicatif"
  Une propriété "Indicatif" peut exister individuellement.

- Classe des propriétés "Libellé"
  Une propriété "Libellé" ne peut exister qu'en association avec une propriété "Indicatif".

Remarquons qu'une même propriété peut être à la fois une propriété "Indicatif" et une propriété "Libellé".

Exemple : La propriété "Code condition de paiement" est une propriété "Libellé" au niveau de la définition d'un client et elle est une propriété "Indicatif" qui identifie les libellés "Condition de paiement" qui sont imprimés sur la facture dans la langue d'établissement de celle-ci.
Ces propriétés sont considérées comme des propriétés "Indicatif".

3.4.3. Création d'une fiche par propriété "Indicatif".
Cette fiche est constituée comme suit :

<table>
<tr><td colspan="3">Nom et numéro de la propriété</td></tr>
<tr><td colspan="2">Liste des relations</td><td rowspan="2">Liste des propriétés "Libellé" qui dépendent de la propriété "Indicatif" considérée.</td></tr>
<tr><td>Type de relation</td><td>propriété "Indicatif avec laquelle la propriété considérée est en relation</td></tr>
</table>

Les propriétés en relation One to One sont regroupées pour former une entité dont les propriétés sont les

éléments de l'union des propriétés "Libellé" associés.

Les propriétés dont la longueur est $\leqslant$ à quatre caractères sont considérées comme propriétés "Libellé" dont les propriétés "Indicatif" sont les propriétés avec lesquelles elles étaient en relation.

Sur base de ce regroupement de propriétés "Indicatif", recréer une nouvelle liste des propriétés "Indicatif" et examiner cette liste d'un point de vue application en créant des sous-ensembles de propriétés fréquemment utilisées ensembles.

Examiner la possibilité de réaliser de ces sous-ensembles une seule entité.

La procédure employée pour établir la différence entre propriétés et entités, conduit à une structuration du réel perçu et représente le graphe du modèle relationnel de la base de données à créer.

* * * * *

4. IMPLEMENTATION D'UN SOUS-ENSEMBLE DU MODELE D'ACCES PAR LES SYSTEMES IDS ET IMS

4.1. Les entités. Type de segment. Type d'article

4.1.1. Pour le système IDS, une entité sera représentée par un type d'article.

Tous les articles (réalisations du type d'article) de même type auront la même longueur fixe, c'est-à-dire que l'ensemble des propriétés caractérisant un type d'article est fixe (nous ne pouvons pas ajouter ou supprimer de propriétés à un type d'article) et toutes les valeurs de chaque propriété ont même nombre de caractères.

Les propriétés seront définies à deux niveaux :
- Le premier niveau est celui de gestion de la base de données.
- Le deuxième niveau est celui d'utilisation dans un programme d'application.

Ces deux niveaux sont définis en langage COBOL dans chaque programme d'application utilisant la base de données (le langage COBOL est le seul langage hôte accepté par le système IDS). Les sous-programmes de gestion du système IDS ne connaîtront que les niveaux COBOL 01 et 02 de la définition du type d'article. Les niveaux COBOL 03 à 49 peuvent être utilisés pour définir des sous-rubriques des niveaux 02. Donc toute propriété qui servira comme zone de contrôle ou qui pourra être modifiée par le système IDS devra être définie comme un niveau 02 COBOL, en plus toutes les clauses du COBOL peuvent être utilisées avec ce niveau sauf :
- OCCURS
- EDITING CLAUSES
- RENAMES
- COPY

Le nom du type d'article est donné au niveau 01 COBOL et peut avoir 1 à 30 caractères, cependant les 8 premiers caractères doivent être unique dans le système.

Toutes les propriétés identifiantes, et d'ordre devront être définies au niveau 02 COBOL.

Nous pourrons avoir 999 types d'articles différents dans une base de données.

Exemple : Description d'un type d'article

```
01 I422.   Connu d'IDS
02 R14     PICTURE X (26). Connu de COBOL et d'IDS
02 DAT422  PICTURE X (4).
02 S422    PICTURE X(50).
03 R62     PICTURE X. Connu uniquement de COBOL
03 R15     PICTURE X (25).
03 R20     PICTURE X (4).
03 R16     PICTURE X(20).
```

Si nous voulions employer la propriété R20 dans un contrôle ou un appel IDS, nous devrions la définir au niveau CODOL 02 car telle qu'elle est définie ici, elle n'est pas connue du système IDS.

4.1.2. Pour le système IMS, une entité sera représentée par un type de segment.

Tous les segments (réalisations de types de segment) de même type auront la même longueur fixe.

Les propriétés d'une entité seront représentées par

les rubriques de la définition du type de segment.

Ces rubriques seront définies à deux niveaux :

- Le niveau de gestion de la base de données par le système IMS.
  Les sous-programmes de gestion du système IMS ne connaîtront que les types de segments et les rubriques (propriétés) définis à ce niveau. Celui-ci sera spécifié dans un bloc appelé Data Base Definition ou DBD.

Exemple :

```
SEGM NAME  = I422    ,   BYTES = 80
FIELD NAME = R14     ,   BYTES = 26   ,   START = 01 , TYPE = C
FIELD NAME = DAT422  ,   BYTES = 04   ,   START = 27 , TYPE = C
FIELD NAME   S422    ,   BYTES = 50   ,   START = 31 , TYPE = C
```

- Le deuxième niveau est celui d'utilisation dans un programme d'application.
  Si un programme d'application est sensible (cette sensibilité est déterminée dans un Program Control Block ou PCB) à ce type de segment, alors celui-ci est décrit dans le programme, en reprenant les noms donnés aux types de segment et aux rubriques dans le DBD pour plus de compréhension mais cela est indépendant du système, toutes les possibilités du langage hôte pourront être utilisées.

Exemple : En reprenant l'exemple précédant en COBOL, nous aurions :

```
01 I422.
02 R14     PICTURE X (26).
02 DAT422  PICTURE X (4).
02 S422.
03 R62     PICTURE X.
03 R15     PICTURE X (25).
03 R20     PICTURE X (4).
03 R16     PICTURE X (20).
```

Ceci n'est qu'un exemple, nous aurions pu découper le type de segment d'une toute autre façon.

Toutes les propriétés identifiantes et d'ordre devront être définies au niveau du DBD par un FIELD NAME. L'ensemble de ces propriétés doit être tel qu'il représente une zone continue de caractères dans le type de segment, l'ordre des propriétés dans cet ensemble est important, il détermine quelle est la propriété majeure, moyenne, mineure de la structure d'ordre. La zone continue, représentant l'ensemble des propriétés d'ordre, sera appelée zone de séquence et aura un maximum de 256 bytes.
Nous pourrons avoir 255 types de segment par base de données physique.

4.2. Les relations d'accès. Implémentation

Nous reprendrons chaque type de relation rencontré dans la présentation du modèle d'accès, et nous analyserons la ou les possibilités offertes par les systèmes IDS et IMS pour réaliser l'implémentation et nous donnerons les points communs et les différences de l'implémentation d'un système par rapport à l'autre ce qui nous permettra, dans le chapitre suivant, de présenter un modèle d'accès implémentable à la fois par les deux systèmes.

4.2.1. Les relations One to One à membre obligatoire

4.2.1.1. Implémentation par le système IDS

La relation One to One à membre obligatoire n'est

pas réalisée comme telle par le système IDS, mais il offre la possibilité de l'implémenter.

a. Nous réserverons à la description d'une entité et en début des zones données, autant de fois des rubriques de 4 caractères que l'entité n'intervient dans des relations One to One à membre obligatoire, par exemple si l'entité intervient dans deux relations de cetype, nous réserverons deux rubriques de 4 caractères, la première rubrique correspondant à la première relation déclarée et ainsi de suite.
Nous appellerons ces rubriques des rubriques "pointeur", pour plus de facilité de description, nous considérerons que l'entité intervient dans une relation "One to One" à membre obligatoire.

. A la création d'une réalisation source de la relation, nous mettrons zéro dans la rubrique pointeur.

. A la création d'une réalisation but de la relation, il faut donner une réalisation source de la relation à laquelle elle doit être rattachée. Nous rechercherons cette réalisation, si elle existe et si sa rubrique "pointeur" égale zéro nous acceptons la création, nous plaçons le contenu de la DIRECT-REFERENCE (qui contient le "reference code" de la réalisation source) dans la rubrique "pointeur" de la réalisation but et nous la <u>créons</u> dans la base de donnée ; la zone DIRECT-REFERENCE contient alors le "reference code" de la réalisation créée, nous plaçons son contenu dans la rubrique "pointeur" de la réalisation source que nous modifions.
Si la réalisation source n'existait pas dans la base de données ou si sa rubrique "pointeur" était différente de zéro, nous rejeterons la création.

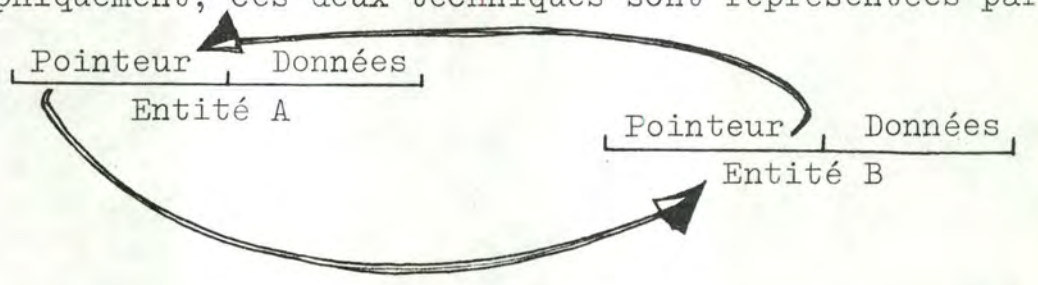
. A la suppression de la réalisation but de la relation, nous recherchons la réalisation source correspondante par un RETRIEVE DIRECT après avoir placer la rubrique "pointeur" dans la zone DIRECT-REFERENCE et nous modifions cette réalisation source après avoir remis sa rubrique "pointeur" à zéro.

. A la suppression de la réalisation source de la relation, nous réalisons un traitement inverse au traitement précédant sauf que nous supprimons la réalisation but au lieu de simplement modifier sa rubrique pointeur.

. La recherche de la réalisation source ou but correspondante à partir de la réalisation but ou source se réalise comme suit :
MOVE rubrique "pointeur" TO DIRECT-REFERENCE
RETRIEVE DIRECT
Cette technique n'est possible que si les entités intervenants dans la relation, possèdent d'autres relations d'accès, en effet pour pouvoir placer une réalisation dans la base de données, il faut avoir définit l'entité comme type d'article, celui-ci devant posséder une option d'accès IDS.

b. Une autre possibilité similaire à la précédente est donnée pour les types d'articles "primaire", "calculé" ou "secondaire ordonné" définit avec MATCH-KEY et SELECT UNIQUE, en remplaçant la notion "reference code" par une adresse symbolique, la rubrique "pointeur" ayant alors la longueur de l'adresse symbolique de l'autre entité de la relation.

Graphiquement, ces deux techniques sont représentées par



Les zones "pointeur" étant soit le "reference code" soit l'adres-se symbolique de la réalisation correspondante.
Ces techniques sont employées dans les programmes "utilisateur" pour établir des relations "One to One" à membre obligatoire entre des entités appartenant à des SUB-FILE différents.
Pour notre modèle, nous n'employerons pas ces techniques, mais la méthode suivante :

c. Nous implémenterons les relations "One to One" à membre obligatoire à l'aide des relations "One to Many" à membre obligatoire présentées par le système IDS.
Il suffit lors de la création d'une réalisation de l'entité but de parcourir la relation "One to Many" correspondante, de vérifier qu'aucune occurence de cette relation n'existe déjà pour accepter la création de cette réalisation.
Tous les problèmes posés par la recherche d'une réalisation but ou source, par la suppression d'une réalisation et par la création d'une réalisation source, sont pris en charge par le système IDS lui-même.

4.2.1.2. Implémentation par le système IMS

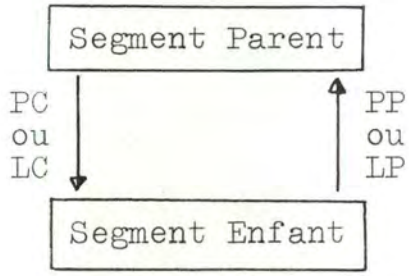
a. La méthode (a) utilisée pour l'implémentation par le système IDS, n'est pas applicable au système IMS, celui-ci ne permettant pas de connaître le "reference code" (adresse logique) du segment retrouvé et ne fournit aucune instruction du type RETRIEVE DIRECT
Par contre, la méthode utilisant l'adresse symbolique des réalisations à relier peut être utilisée suivant le même principe pour le système IMS.
Cette méthode n'est réalisable que si les entités en relation sont déclarées comme type de segment racine ou comme type de

segment dépendant dont la CONCATENED KEY est complète et unique. (La CONCATENED KEY est la concaténation des zones de séquence de tous les types de segment se trouvant sur le chemin qui conduit du type de segment racine au type de segment dépendant considéré).
La recherche de la réalisation correspondante s'effectuerait comme suit : .MOVE rubrique pointeur symbolique TO zone FIELD du SSA qualifié correspondant au type d'article recherché avec le code commande C.
.GET UNIQUE avec ce SSA.

Graphiquement nous avons la même représentation que précedemment.

b. De même que pour le système IDS, nous implémenterons les relations "One to One" à membre obligatoire à l'aide des relations "One to Many" à membre obligatoire présentées par le système IMS, en restreignant la portée de ces relations par un test sur l'existence ou la non-existence d'une occurence de la relation lors de la création d'une réalisation de l'entité but.

Graphiquement nous aurons :



Pour plus de détails, nous renvoyons à la présentation des relations "One to Many" à membre obligatoire.

4.2.2. Les relations "One to Many" à membre obligatoire

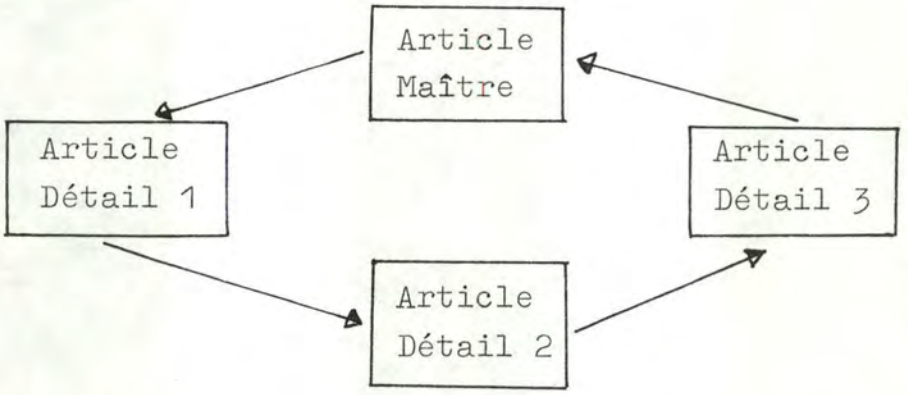
4.2.2.1. Implémentation par le système IDS

a. La relation One to Many à membre obligatoire est la relation de base de l'IDS où elle est représentée par la notion de chaîne, dans laquelle le type d'article source est appelé MASTER et le type d'article but, DETAIL de la relation.

La chaîne est constituée par une suite d'articles; le premier article est l'article MASTER qui contient un pointeur vers le premier article détail, chaque article détail possède un pointeur vers l'article détail suivant, le dernier article détail pointant vers l'article maître. Donc tous les articles d'une chaîne sont associés en une boucle fermée dont les caractéristiques sont :

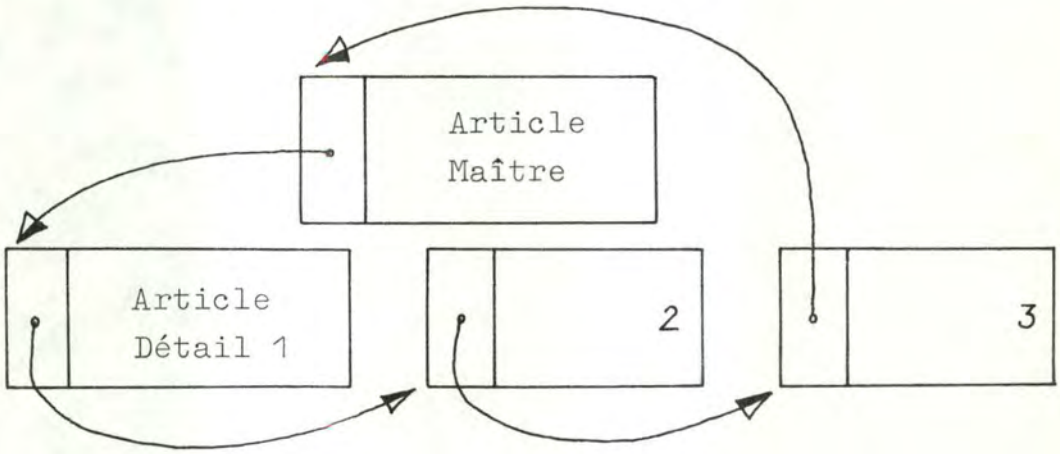
- contient un article "maître" et un nombre quelconque d'article "détail"
- Il faut parcourir la chaîne pour retrouver (accéder) un article détail.

Schématiquement, nous avons donc :

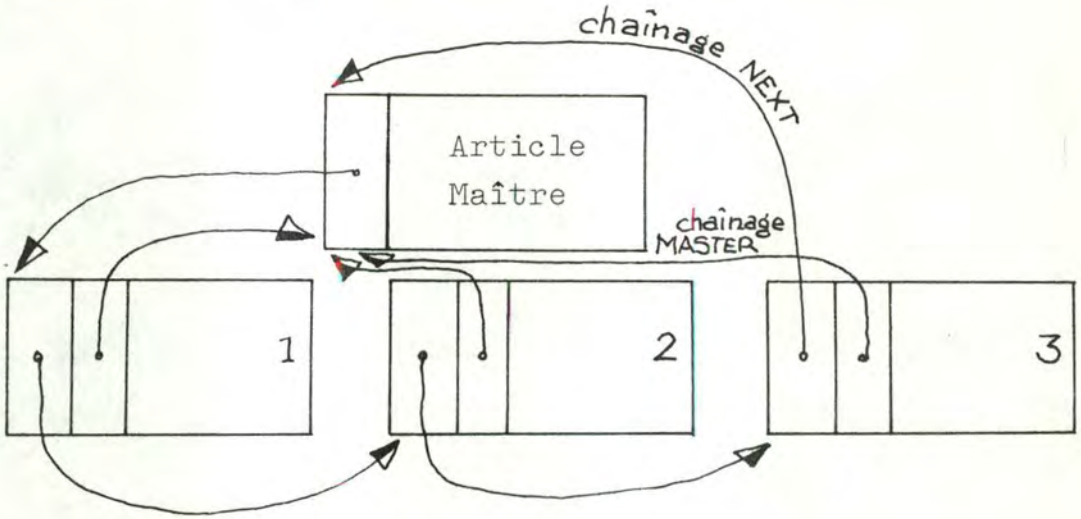


Cette chaîne est réalisée à l'aide de pointeurs qui sont placés dans une zone de l'article IDS inaccessible à

l'utilisateur et appelée zone de chaînage. Chaque pointeur contient un code de référence de 4 caractères représentant l'adresse logique de l'article vers lequel il pointe.
La relation "One to Many" à membre obligatoire est réalisée à l'aide des pointeurs NEXT.

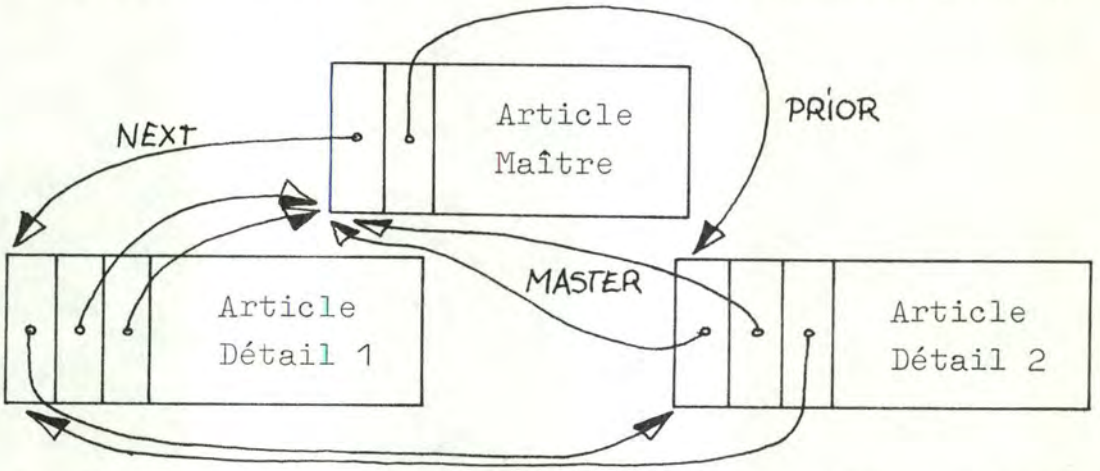


Ce pointeur est le seul pointeur obligatoire.
La relation inverse existe toujours implicitement du fait que le dernier article détail pointe vers le Maître, mais cette relation inverse peut être explicitement demandée, ce qui s'exprimera par l'option "LINKED TO MASTER" et sera réalisée par un pointeur, appelé pointeur MASTER, placé dans tous les articles détails et contenant le code de référence de l'article maître de la chaîne.



L'IDS a, en plus, donné à l'utilisateur la possibilité de demander un autre pointeur optionnel qui réalise le chaînage inverse du chaînage NEXT et qui est appelé pointeur PRIOR.

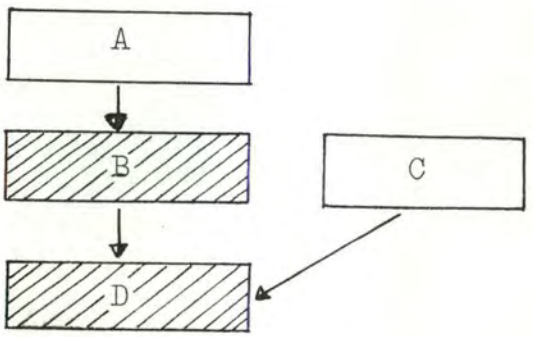
Ces trois types de pointeurs peuvent être définis pour la même relation ; dans ce cas nous avons schématiquement :



b. Comme la relation est à membre obligatoire, lui sont associées des propriétés d'insertion et de suppression d'articles.

- Pour insérer un article "Détail", il faut que l'article "Master" correspondant existe préalablement dans la base de données.
- Si un article "Master" est supprimé, tous les articles "Détail" sont supprimés, cette procédure est récursive c'est à dire que tous les articles "Détail" des articles "Détail" supprimés sont aussi supprimés et ainsi de suite.

Exemple :



Si nous supprimons l'article B, alors les articles D sont supprimés, les articles A et C existent toujours, mais les liaisons AB et CD sont supprimées.

4.2.2.2. Implémentation par le système IMS

Le système IMS présente quatre types d'organisation qui sont HSAM, HISAM, HIDAM et HDAM et deux types généraux de pointeurs qui sont les pointeurs "Direct" et les pointeurs "Symbolic".

Les pointeurs "Direct" ne peuvent être utilisés que dans les deux types d'organisation HIDAM et HDAM ; ce seront ces organisations et ces pointeurs que nous présenterons.

Les pointeurs "Symbolic" peuvent être utilisés dans les quatre organisations et sont construits à partir de données de la base de données elle-même. Ce sont, en fait, les clés concaténées des segments c'est-à-dire l'ensemble des clés de tous les segments hiérarchiquement plus haut dans la base des données.

a. La relation One to Many à membre obligatoire est la relation de base de l'IMS où elle est représentée par les notions :
- PHYSICAL PARENT - PHYSICAL CHILD - PHYSICAL TWIN

et

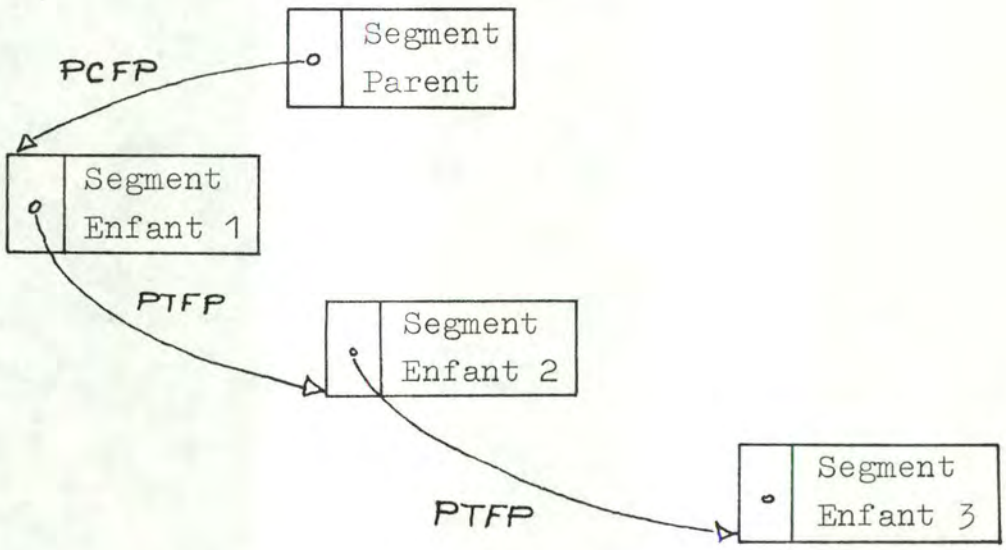
- LOGICAL PARENT - LOGICAL CHILD - LOGICAL TWIN

Le type de segment source est appelé type de segement PARENT
Le type de segment but est appelé type de segment ENFANT.

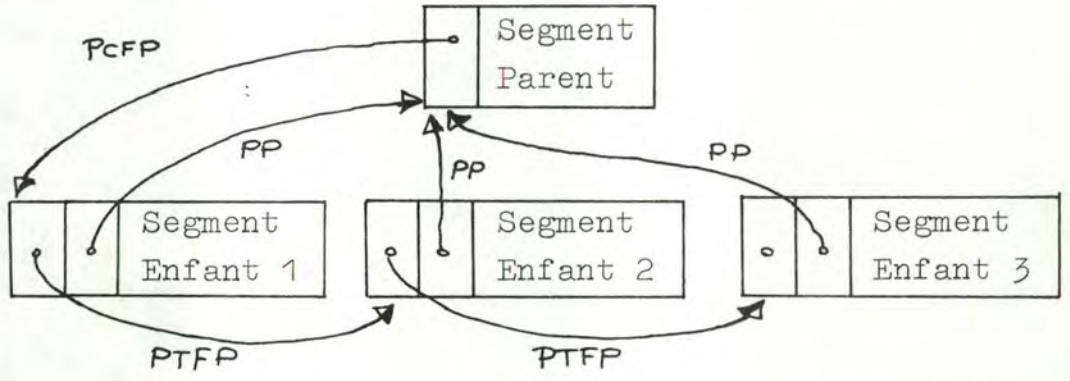
. A un segment Parent (réalisation) seront associés n segments ENFANT avec n pouvant varier de zéro à l'infini, cette liaison sera réalisée sous forme de liste simple formée par des pointeurs placés dans une zone du segment IMS inaccessible à l'utilisateur et appelée zone des pointeurs.

. Dans le système IMS, la relation One to Many à membre obligatoire porte deux noms différents suivant que le segment enfant est un segment enfant physique ou un segment enfant logique ; nous aurons la relation physique ou la relation logique.
La liste ouverte représentant la relation physique est réalisée à l'aide de deux pointeurs ; le segment parent possède un pointeur vers le premier segment enfant ; ce pointeur s'appellera donc le "Physical Child first pointer".
Chaque segment enfant possède un pointeur vers le segment enfant suivant dans la liste, le dernier segment enfant de la liste aura son pointeur à zéro ; ce pointeur s'appellera le "Physical Twin forward pointer".

Schématiquement nous aurons donc :



Pour retrouver un segment enfant, il faudra parcourir la liste à partir du segment parent.
Remarquons que si l'ensemble de ces pointeurs donne la même liaison que le pointeur NEXT de l'IDS, il ne donne pas implicitement la relation inverse.
Cette relation inverse, qui est indispensable dans le cas de segment d'intersection, est réalisée explicitement par le système IMS lorsque justement le segment enfant est un segment qui participe à une relation logique, c'est-à-dire est un segment d'intersection.
Dans ce cas, chaque segment enfant aura un pointeur contenant le code de référence du segment parent et s'appelant "Physical parent pointer", nous aurons alors :



Comme pour le système IDS, l'utilisateur a la possibilité de définir deux types de pointeurs optionnels qui réaliseront la même fonction que le pointeur PRIOR de l'IDS.
Ce sont les pointeurs :

- Physical child last pointer qui du segment parent pointe vers le dernier segment enfant
- Physical twin backward pointer qui de chaque segment enfant pointe vers le précédent, le premier segment enfant ayant ce pointeur à zéro.

. La relation logique possède les mêmes possibilités et les mêmes pointeurs que la relation physique ; seul le nom des pointeurs change, nous l'obtenons en remplaçant le terme "Physical" par "Logical".
Nous avons donc pour la relation logique, les pointeurs suivant :

- Ensemble obligatoire : - Logical child first pointer
  - Logical twin forward pointer
- Ensemble optionnel : - Logical child last pointer
  - Logical twin backward pointer
- Cas particulier : - Logical parent pointer.

b. Nous avons vu qu'aux relations à membre obligatoire sont associés des propriétés d'insertion et de suppression de segments.
Dans le cas du système IMS, il y a des différences dans les règles d'insertion et de suppression suivant que le segment considéré est un segment enfant physique ou un segment enfant logique.
Les segments racines et les segments enfant physique suivent les règles d'insertion et de suppression relatives aux relations à membre obligatoire.
Les segments enfant logique et les segment en relation avec ceux-ci, c'est-à-dire les parents physiques et les parents logiques de ces enfants logiques, possèdent des possibilités particulières.

- Insertion d'un segment enfant logique
  Il existe deux possibilités qui sont :
  Règle physique d'insertion :
  - L'insertion d'un segment enfant logique ne peut se réaliser que si le segment destination parent concerné se trouve déjà dans la base de données.

Règle logique et virtuelle d'insertion:

- L'insertion d'un segment enfant logique peut toujours se réaliser et si le segment destination parent concerné ne se trouve pas encore dans la base de données, il sera également inséré.

Rappelons que le segment "destination parent" peut être :
- le segment parent logique
- le segment parent physique

suivant le chemin parcouru pour obtenir le segment enfant logique, et qui sera respectivement :
- le chemin physique
- le chemin logique

- Suppression d'un segment "enfant logique"

Il existe trois possibilités, qui sont :

Règle physique de suppression

Il faut supprimer deux fois le segment "enfant logique", d'abord en l'accédant via le chemin logique, ensuite en l'accédant par le chemin physique.

Règle logique de suppression

Il faut supprimer deux fois le segment "enfant logique" L'ordre dans lequel les suppressions sont réalisées n'a pas d'importance.

Règle virtuelle de suppression

Il suffit de supprimer une fois le segment "enfant logique". Le système IMS supprime automatiquement toutes les occurrences des relations dans lesquelles ce segment intervient.

- Suppression d'un segment "destination parent"

  Règle physique de suppression :

  La suppression d'un segment "destination parent", dont le type de segment a été déclaré avec la règle de suppression physique, ne peut s'effectuer que si la suppression de tous les segments "enfant logique" avec lesquels il était en relation, a été préalablement réalisée.

  Règle logique ou virtuelle de suppression :

  La suppression d'un segment "destination parent" sera toujours exécutée, elle entraînera la suppression de tous les segments "enfant logique" avec lesquels il était en relation.

Pour réaliser l'implémentation des relations "One to Many" à membre obligatoire vu leurs implications sur la création ou la suppression des segments intervenants dans leurs occurrences nous devons choisir les règles suivantes :

- Règle d'insertion physique pour un segment "destination parent" ou pour un segment "enfant logique".
- Règle de suppression virtuelle pour un segment" enfant logique"
- Règle de suppression logique pour un segment "destination parent".

En plus, les règles que nous retenons sont les règles que le système IMS recommande d'utiliser car il n'y a pas alors de différence dans le résultat d'une action suivant que le système d'implémentation des relations logiques est le "Physical Pairing" ou le "Virtual Pairing".

L'explication de ces termes sera donnée dans la partie consacrée à l'implémentation des relations "Many to Many" faibles.

4.2.3. Les relations "Many to Many" à membre obligatoire

Les relations "Many to Many" à membre obligatoire ne sont pas implémentées par les systèmes IDS et IMS et nous ne réaliserons pas l'implémentation de ce type de relation mais bien l'implémentation des relations "Many to Many" faibles qui sont d'un emploi beaucoup plus courant, les entités en relation "Many to Many" ayant bien souvent une existence indépendante.

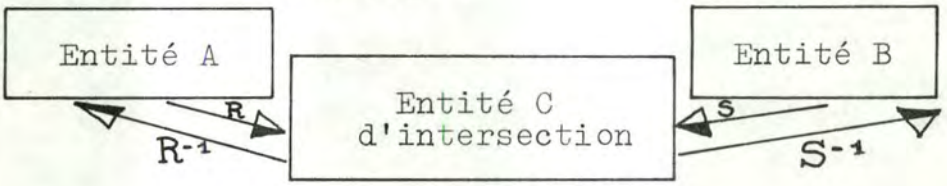
En plus, ces entités sont habituellement dans des bases de données physiques séparées et leurs réalisations sont, de ce fait, créées et réorganisées, par le système IMS, de façon indépendante, ce qui augmente la difficulté d'implémenter les relations "Many to Many" à membre obligatoire.

4.2.4. Les relations "Many to Many" faibles

4.2.4.1. Généralités

Une relation "Many to Many" faible peut être réalisée grâce à deux relations "One to Many" à membre obligatoire et une entité dite d'intersection.

Schématiquement nous aurons :



Les deux entités A et B sont reliées par une relation "Many to Many" faible, les relations R et S étant des relations "One to Many" à membre obligatoire.

- Relation "Many to Many"

  En effet, à une réalisation a de l'entité A correspondent n réalisations c_i avec $i \in [o,n]$ de l'entité C par la relation R et à une réalisation c_i de l'entité C correspond une et une seule réalisation b_i de l'entité B par la relation inverse de la relation S, donc à une réalisation a de l'entité A correspondent n réalisations b_i de l'entité B et réciproquement à une réalisation b de l'entité B correspondent m réalisations a_i de l'entité A par les relations S et R^{-1}.
  L'image d'une réalisation a de l'entité A dans l'ensemble des réalisations de l'entité B par la relation "Many to Many" est S^{-1} (R(a)), et l'image d'une réalisation b de l'entité B dans l'ensemble des réalisations de l'entité A par la relation inverse "Many to Many" est R^{-1} (S(b)).
  Pour les relations "Many to Many", la relation inverse sera toujours réalisée.

- Relation faible

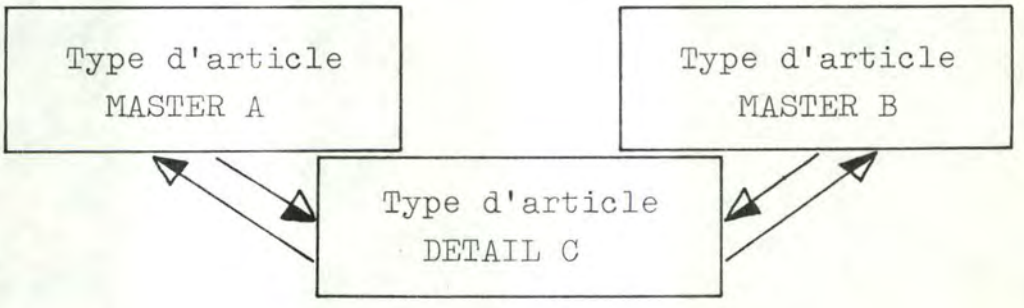
  La suppression d'une réalisation a de l'entité A entraîne automatiquement la suppression de toutes les réalisations c_i de l'entité C qui lui sont associées car la relation R est une relation "One to Many" à membre obligatoire, mais n'a aucune influence sur les réalisations de l'entité B.
  La suppression d'une réalisation a de l'entité A entraîne seulement la suppression des occurrences de la relation avec l'entité B, dans lesquelles elle intervient.
  Les réalisations des entités A et B peuvent exister dans

la base de données indépendamment les unes des autres, c'est-à-dire que la création d'une réalisation d'une des deux entités n'est pas subordonnée à l'existence préalable de l'autre.
La création d'une occurrence (a,b) de la relation entre les entités A et B s'effectue par la création d'une réalisation c de l'entité C et par la création automatique (propriété des relations "One to Many" à membre obligatoire) des occurrences (a,c) et (b,c) des relations R et S.

La suppression d'une occurrence de la relation est réalisée par la suppression de la réalisation de l'entité C qui intervient pour établir cette occurrence.

4.2.4.2. Implémentation par le système IDS

La technique d'implémentation par le système IDS des relations "Many to Many" faible est celle appliquée dans les généralistés.
Les trois types de pointeurs IDS sont permis dans l'implémentation des relations R et S, mais remarquons que le pointeur MASTER est un pointeur optionnel, le pointeur NEXT réalisant la relation et son inverse. Mais pour des problèmes de temps d'accès, il est conseillé d'implémenter explicitement les relations inverses à l'aide de ce pointeur.
Graphiquement nous aurons donc :

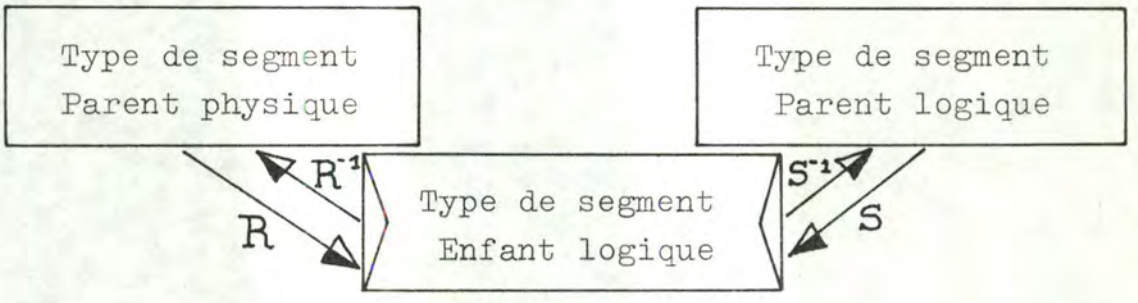


Dans le système IDS, le type d'article C d'intersection n'a aucun statut particulier, aucune relation n'est donnée pour son utilisation.

4.2.4.3. Implémentation par le système IMS

a. Les relations "Many to Many" faible sont réalisées à l'aide de types de segment d'intersection appelés type de segment "enfant logique" et de deux relations "One to Many" à membre obligatoire, comme expliqué dans les généralités. L'une des relations "One to Many" à membre obligatoire est une relation physique c'est-à-dire établie entre un type de segment "parent physique" et un type de segment "enfant physique", l'autre est une relation logique c'est-à-dire établie entre un type de segment "parent logique" et un type de segment "enfant logique".

Graphiquement nous avons, en employant les symboles graphiques de l'IMS qui représente un type de segment "enfant logique" par un hexagone pour le différentier des autres types de segment :



La relation R est réalisée par les pointeurs :

- Physical child first pointer
- Physical twin forward pointer

La relation S est réalisée par les pointeurs :

- Logical child first pointer
- Logical twin forward pointer

Les relations R^{-1} et S^{-1} sont implémentées par le système IMS respectivement par les pointeurs :

- Physical Parent pointer

et - Logical Parent pointer

Ici, ces deux pointeurs ne sont pas optionnels car les relations inverses des relations R et S ne sont pas implicitement implémentées par les pointeurs réalisant ces relations R et S.

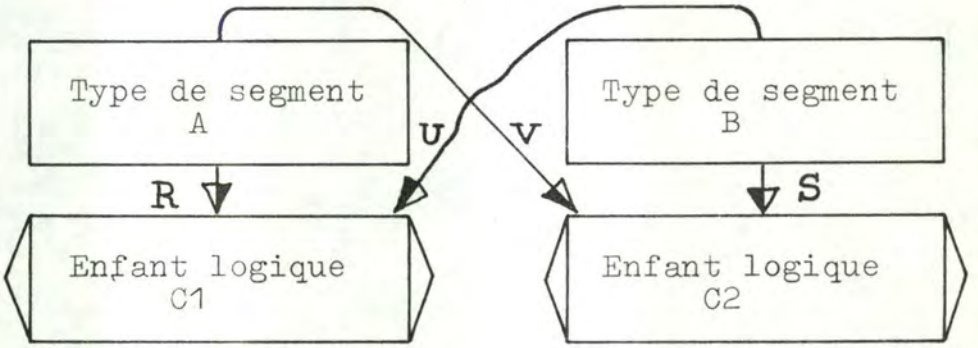
b. Le système IMS dispose de deux techniques pour réaliser les relations logiques :

- Le physical pairing
- Le logical pairing

Le physical pairing

Le physical pairing est une technique qui établit une relation bidirectionnelle (la relation et son inverse) entre deux entités à l'aide de deux types de segments "enfant logique".

Graphiquement nous avons :

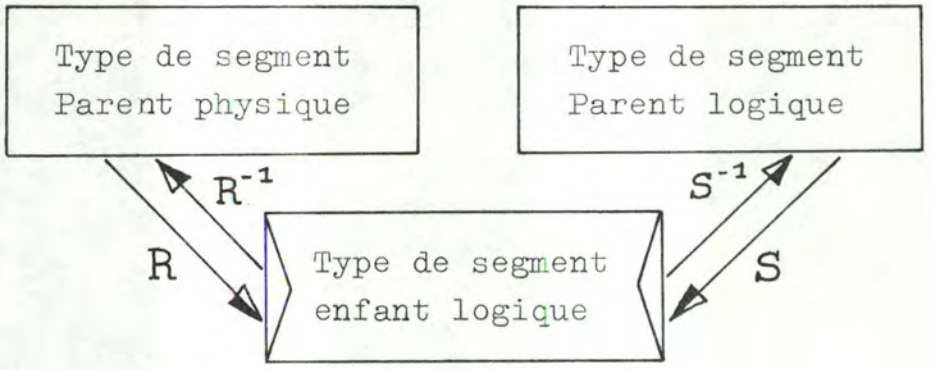


La relation R entre les types de segment A et C1 et la relation S entre les types de segment B et C2 sont des relations physiques.
Les relations U et V sont des relations logiques.
Il existe donc physiquement deux types de segment "enfant logique" dans la définition de la base de données. A la création d'une réalisation d'un des types de segment "enfant logique", le système IMS crée automatiquement la réalisation correspondante de l'autre type de segment "enfant logique" pour établir la relation inverse.

Le virtual pairing

Le virtual pairing signifie que la relation bidirectionnelle entre deux types de segment est réalisée à l'aide d'un seul type de segment "enfant logique", l'autre type de segment "enfant logique" étant virtuel.

Graphiquement nous aurons :



La relation R et son inverse sont des relations physiques
La relation S et son inverse dont des relations logiques

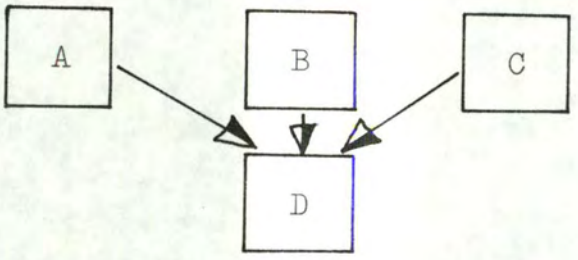
Avec les règles d'insertion et de suppression de segments "enfant logique" et des segments en relation avec ceux-ci, que nous avons choisi, les deux techniques "Physical pairing" et "Virtual Pairing" ont la même portée sémantique.
Dans la suite, nous nous limiterons donc à la technique "Virtual Pairing".

c. Si pour le système IDS, un type d'article d'intersection n'a aucun statut spécial et est considéré comme un type d'article normal, il n'en va pas de même pour le système IMS.
Rappelons que le système IMS présente deux structures.
La première structure est appelée structure physique et peut être considérée comme une structure en réseau sans cycle, la deuxième structure, appelée structure logique et représentée par une structure hiérarchique, est déduite de la première structure et représente la vue que le programmeur d'application a de la base de données. La transformation de la structure en réseau en sous-structures hiérarchiques a été voulue pour faciliter le travail et la responsabilité du programmeur qui dans le cas d'une base de données en réseau, en citant Bachman, devient un navigateur; nous lui restituerons d'ailleurs ce rôle dans notre modèle d'accès défini au chapitre suivant.
Cependant cette transformation, qui a comme pivot les types de segment "enfant logique", a introduit certaines règles restrictives sur l'utilisation de ceux-ci.
- Un type de segment "enfant logique" doit avoir un type de segment "parent physique" c'est-à-dire qu'il ne peut pas être un type de segment racine. Il ne peut donc pas être un point d'entrée dans la base de données.
- Un type de segment "enfant logique" ne peut avoir qu'un type de segment "parent physique" et qu'un type de segment "parent logique". Cela signifie que le système IMS ne permet pas qu'un segment soit dépendant dans plus de deux relations.

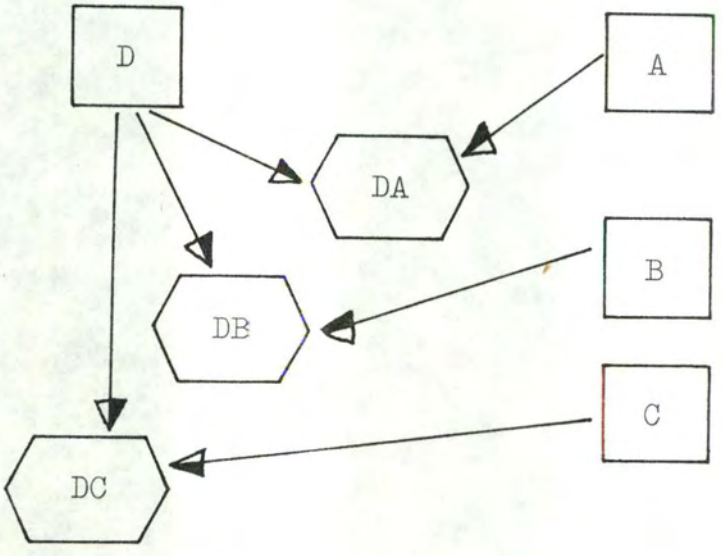
La multiplicité des relations sera obtenue par la multiplicité des types de segments "enfant logique" avec évidemment un changement du contenu sémantique.

Exemple :

Structure du modèle d'accès



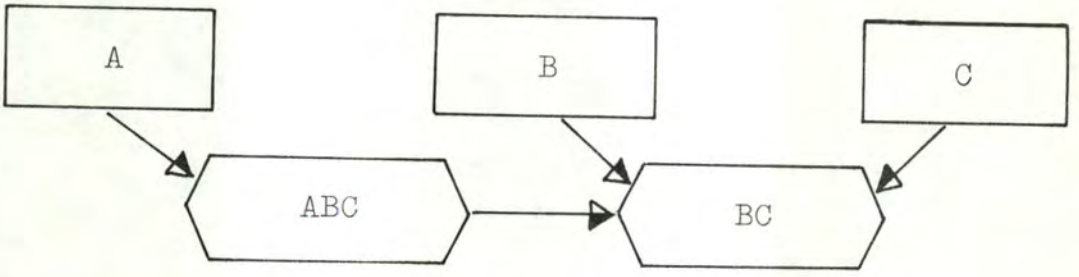
Structure IMS



Les relations inverses ne sont pas représentées pour faciliter la lecture du graphe, mais elles existent.

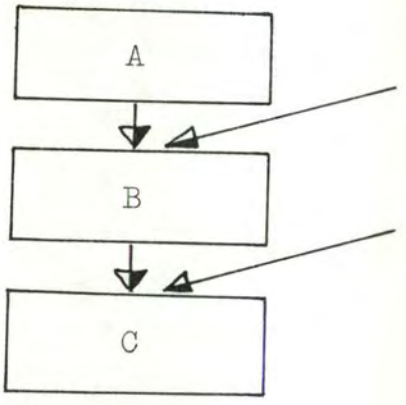
- Un type de segment d'une base de données physique ne peut être à la fois un type de segment "enfant logique" et un type de segment "parent logique"

Exemple : Cette structure est interdite par IMS



- Un type de segment "enfant logique" ne peut avoir aucun type de segment dépendant physique qui soit lui-même un type de segment "enfant logique"

Exemple :

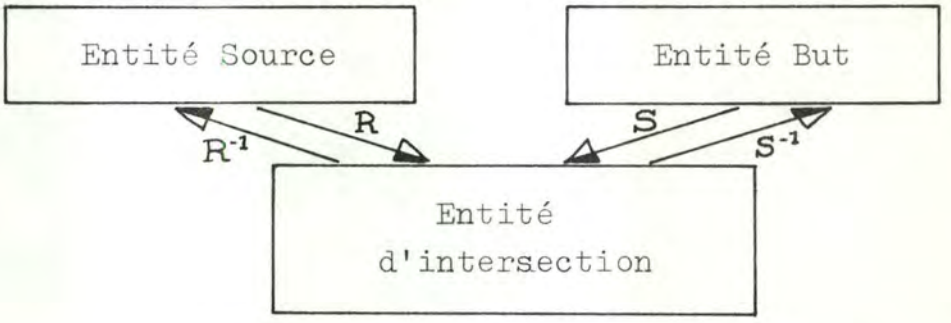


4.2.5. Les relations "One to Many" faibles

Les relations "One to Many" faibles seront considérées comme des cas particuliers des relations "Many to Many" faibles.
Elles seront implémentées ainsi que leurs inverses grâce à

une entité dite d'intersection, une relation "One to Many" à membre obligatoire entre l'entité source et l'entité d'intersection, une relation "One to One" à membre obligatoire établie entre l'entité but (qui est l'entité source de cette relation) et l'entité d'intersection et leurs inverses.

Graphiquement nous aurons donc :



La relation R sera du type "One to Many" à membre obligatoire
La relation S sera du type "One to One" à membre obligatoire.

4.2.6. Les relations "One to One" faibles

Les relations "One to One" faibles seront implémentées de la même façon que les relations faibles vues jusqu'à présent, c'est-à-dire par l'intermédiaire d'une entité d'intersection.
Les deux relations à établir entre cette entité d'intersection (but) et les deux entités en relation "One to One" faible, seront des relations "One to One" à membre obligatoire.
A une réalisation de l'entité source correspond une réalisation de l'entité d'intersection qui est associée à une et une seule réalisation de l'entité but et réciproquement pour la relation inverse.

4.3. Nombre de relations possibles entre deux entités

Nous analyserons la possibilité offerte par les deux systèmes IDS et IMS, de définir plusieurs relations entre deux entités. Ces relations seront du type "One to Many" à membre obligatoire, étant les seules implémentées par ces systèmes, les autres types de relations possibles entre deux entités seront examinées dans la présentation du modèle d'accès puisque pour ce modèle, ces autres types existeront.

4.3.1. Le système IDS

Le système IDS donne la possibilité d'établir plusieurs relations différentes entre deux types d'articles. Cette possibilité est basée sur les notions de :

- SELECT CURRENT
- SELECT UNIQUE
- MATCH-KEY et SYNONYM

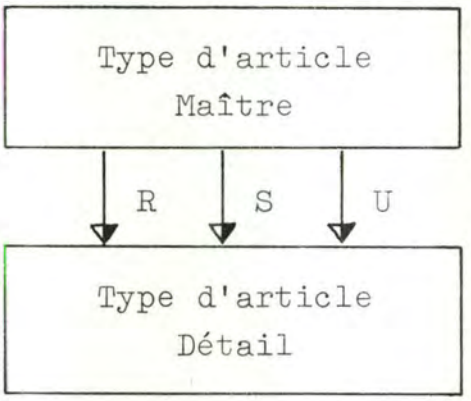
dont la signification est :

- Pour insérer un nouvel article "Détail" dans une chaîne, il faut avoir sélectionné l'article "maître" de cette chaîne. Il existe deux règles pour effectuer la selection d'un article maître pour insérer un nouvel article détail.
  - Select current Master : le dernier article traîté de ce type d'article maître est choisi pour être l'article maître du nouvel article détail.
  - Select unique Master : Le système utilise le critère de recherche établi dans la définition du type d'article maître pour retrouver l'article maître à partir de valeurs placées dans les zones de contrôle de ce

type d'article. Ces zones de contrôle sont appelées Match Key.

Comme plusieurs relations peuvent exister entre deux types d'articles, il faut pouvoir garnir ces "Match Key" par des valeurs différentes ; pour cela, elles porteront plusieurs noms associés par le terme SYNONYM.

Graphiquement nous aurons par exemple :

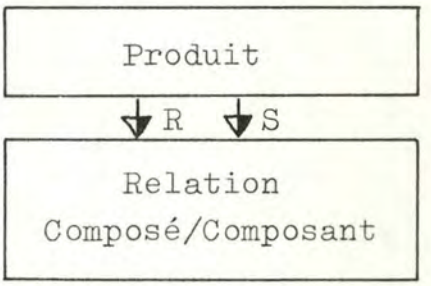


Un des types de chaînes peut être spécifié SELECT CURRENT
Les autres types de chaînes doivent être spécifié SELECT UNIQUE.
Les zones de Match Key du type d'article Maître sont définies avec l'option SYNONYM.

Exemple : 1) Le problème de la nomenclature d'un produit sera résolu par la définition de deux types de chaîne entre deux types d'article, nous aurons :

R est la relation des constituants composé/composant, c'est en fait l'explosion de la nomenclature

S est la relation des cas d'emploi composant /composé, c'est l'implosion de la nomenclature.

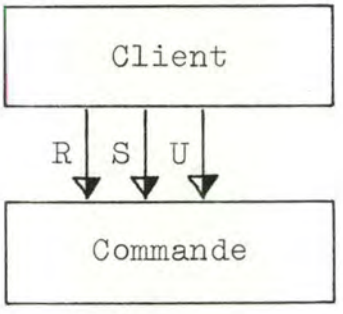


2) Associations entre client et commande

Soient les deux types d'article "Client" et "Commande", nous définirons trois relations différentes entre ces deux types d'article :

- La relation R dont une occurrence donnera le client qui passe la commande
- La relation S dont une occurrence donnera le client qui payera la facture
- La relation U dont une occurrence donnera le client qui recevra la marchandise.

Nous aurons :



4.3.2. Le système IMS

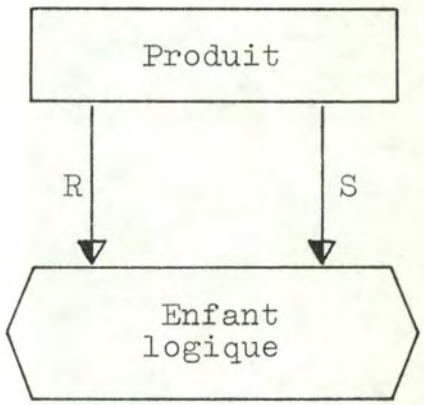
Le système IMS permet d'établir deux relations différentes entre deux types de segment si le type de segment but des deux relations est un type de segment "enfant logique", c'est-à-dire est un type de segment d'intersection.

C'est pour cela que nous considérons ce cas comme n'étant pas différent de l'implémentation d'une relation "Many to Many" faible que nous avons déjà examinée.

Exemple : La nomenclature d'un produit, nous donnera comme graphe :

La relation physique R représente l'explosition de la nomenclature

La relation logique S représente l'implosition de la nomenclature

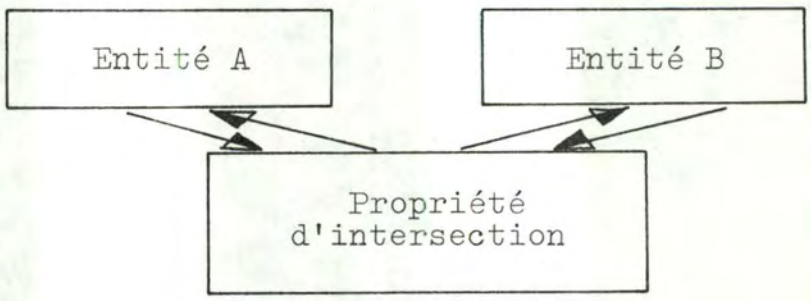


4.4. Association de propriétés ou d'entités à une relation

4.4.1. Association de propriétés à une relation

Nous avons vu précedemment que l'association de propriétés à une relation n'était nécessaire que dans le cas des relations "Many to Many". La technique utilisée pour implémenter ces relations permet d'associer facilement des propriétés à une relation "Many to Many", il suffit de donner ces propriétés à l'entité d'intersection.

Nous aurons donc :



L'entité d'intersection sera :

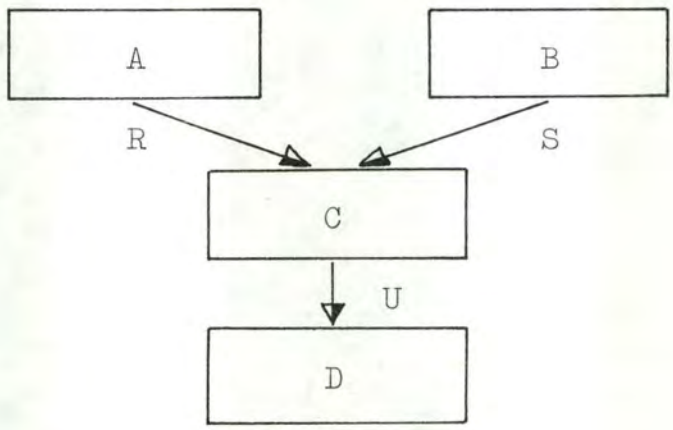
- un type d'article quelconque en IDS
- un type de segment "enfant logique" en IMS.

4.4.2. Association d'entités à une relation

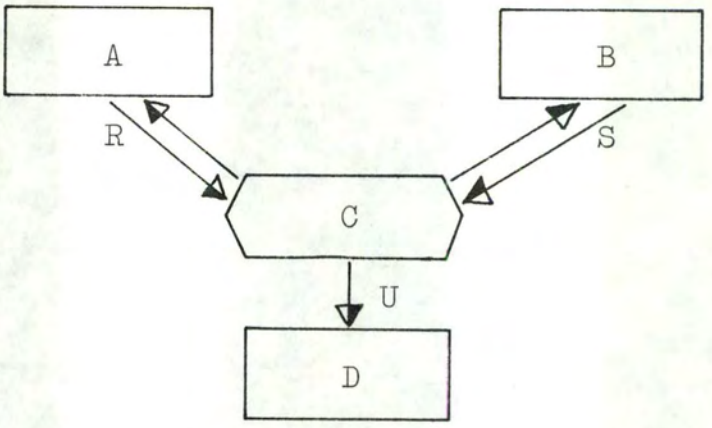
Nous avons vu qu'il était possible d'associer des propriétés à une relation "Many to Many" par le biais d'une entité d'intersection. Toutes les réalisations d'un individu devant avoir la même longueur fixe, si plusieurs valeurs peuvent être données aux propriétés pour la même occurrence de la relation, il n'est pas souhaitable de réserver toute la place nécessaire dans l'entité d'intersection, ce qui signifierait beaucoup de place inutilisée.
Nous pourrons résoudre ce problème en plaçant les propriétés d'intersection variables dans une ou plusieurs entités qui seraient les entités but de relations à membre obligatoire dont l'entité d'intersection serait l'entité source.

En représentation graphique nous aurons :

Dans le système IDS :



Dans le système IMS :



Nous aurons autant de réalisations de l'entité D que nécessaire pour mémoriser les valeurs des propriétés d'intersection.

4.5. Ordre induit par une relation d'accès

4.5.1. Le système IDS

Le système IDS donne six possibilités à l'administrateur de la base de données pour définir un ordre d'accès aux articles à l'intérieur d'une chaîne :

a) CHAIN-ORDER SORTED : avec cette option tous les articles d'une chaîne sont accédés en séquence ascendante ou descendante quelque soit leur type d'article détail. La zone de séquence doit être au même endroit dans tous les articles.

b) CHAIN-ORDER SORTED WITHINTYPE : avec cette option les articles d'une chaîne sont accédés en séquence suivant leur type d'article, indépendamment des autres types.

c) CHAIN-ORDER FIRST : avec cette option chaque nouvel article détail est inséré comme premier article de la chaîne.

d) CHAIN-ORDER LAST : représente l'option inverse de la précédente, chaque nouvel article détail est inséré comme dernier article de la chaîne.

e) CHAIN-ORDER BEFORE : Le nouvel article détail est inséré juste devant l'article courant de la chaîne.

f) CHAIN-ORDER AFTER : représente l'option inverse de la précédente, le nouvel article détail est inséré juste derrière l'article courant de la chaîne.

Chaque propriété d'ordre doit être un niveau 02 COBOL dans la définition de ce type d'article et doit être désignée comme étant une zone de séquence dans la définition du type de chaîne par l'option

ASCENDING
DESCENDING KEY IS nom de la propriété d'ordre.

Par type de chaîne, nous pourrons définir 10 propriétés d'ordre.

Une option existe pour voir si le système peut accepter dans une chaîne, des articles ayant même valeur des propriétés d'ordre associés à ce type de chaîne. Si les doubles sont acceptés, il faut définir l'option d'insertion de tout nouvel article détail.
Nous avons le choix entre :

DUPLICATES
- NOT ALLOWED : les doubles ne sont pas permis
- FIRST : les doubles sont permis et une nouvelle insertion sera le premier article de l'ensemble des articles ayant même valeur des propriétés d'ordre.
- LAST : Les doubles sont permis et une nouvelle insertion sera le dernier article de l'ensemble.

4.5.2. Le système IMS

Pour réaliser l'inventaire des possibilités du système IMS, il faut scinder la présentation en deux parties suivant que le type de segment considéré est un type de segment "enfant logique" ou n'en est pas un.

a. Le type de segment n'est pas un type de segment "enfant logique".
Rappelons que le système IMS n'accepte qu'une zone de séquence d'un maximum de 256 bytes.

- Si le type de segment considéré est un type de segment racine, une zone de séquence doit être spécifiée.
  Dans les organisations HISAM et HIDAM, les clés doubles ne sont pas autorisées, dans l'organisation HDAM, les clés doubles sont autorisées.
- Si le type de segment considéré est un type de segment dépendant, une zone de séquence peut être spécifiée et les clés doubles sont autorisées.
- Pour les types de segment sans zone de séquence, le système propose trois options :
  - FIRST : le nouveau segment est inséré au début de la liste. C'est une structure d'ordre du type LAST IN - LAST OUT.
  - LAST : c'est l'option inverse de l'option précédente, le nouveau segment est inséré en fin de liste, c'est une structure d'ordre du type LAST IN - LAST OUT.
  - HERE : le nouveau segment est placé devant le dernier segment de même type traité par le système IMS.
    Le positionnement sur le dernier segment traité peut être provoqué explicitement par le programme, mais il peut être réalisé par le système IMS et dans ce cas l'option HERE est identique à l'option FIRST.
- Pour les types de segment avec zone de séquence et clé double autorisée, le système propose trois options pour la gestion de l'ensemble des segments ayant la même valeur de la zone de séquence.

Ces options sont : - FIRST
- LAST
- HERE

Leur signification est la même que celle vue précédemment, leur portée étant restreinte à l'ensemble des segments ayant même valeur de la zone de séquence.

b. Le type de segment est un type de segment "enfant logique" Le système permet de ne pas établir de séquence pour ces types de segment.

- Séquence dans la relation physique:

La zone de séquence doit être une zone ininterrompue de caractères.
Elle peut être la clé concaténée du type de segment "parent logique", une partie de celle-ci, ou une zone des données d'intersection.

Les règles pour réaliser l'ordre induit par la relation physique sont les mêmes que pour les autres types de segment.

- Séquence dans la relation logique :

L'ordre peut être réalisé à partir de plusieurs zones de séquence, toutes ces zones doivent faire partie des données d'intersection.

Les règles pour réaliser l'ordre induit par la relation logique sont les mêmes que pour les autres types de segment.

c. Dans tous les cas, l'ordre induit ne pourra être qu'ascendant et la déclaration d'une propriété ou d'un ensemble de propriétés contigües comme zone de séquence sera faite à la description du type de segment dans un DBD par la proposition

NAME = (nom de la zone, SEQ, U ou M) avec U pour clé unique et M pour clés doubles autorisées.

4.6. Remarques

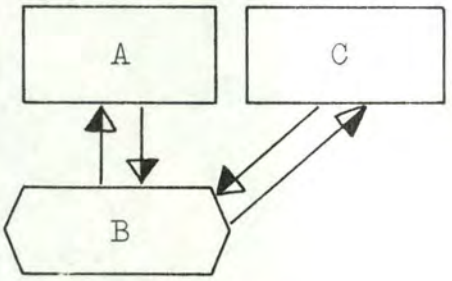
4.6.1. Nous avons rappelé que le système IMS présente la possibilité de définir des relations entre structures hiérarchiques. Ces relations signifient qu'il est possible de définir, en IMS, des structures en reseau sans cycles. Cependant, le programmeur est seulement autorisé à suivre une sous-structure hiérarchique où il est toujours possible de définir un chemin comme le parcours du haut vers le bas et de gauche à droite, alors que le programmeur doit aller pas à pas à travers une structure en réseau.
Nous avons aussi rappelé que le type de segment "enfant logique" était le pivot autour duquel s'articule toute la déduction structure physique - structure logique (hiérarchie)

Nous présentons rapidement les règles de déduction :
- Le type de segment racine d'une structure logique de données doit être le type de segment racine d'une base de données physique
- La structure de la base de données physique dont le type de segment racine est considéré comme type de segment racine de la structure logique, peut être reprise intégralement dans celle-ci.

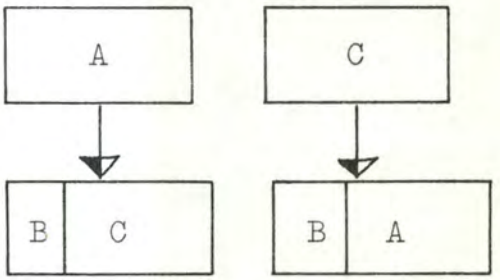
- Un type de segment "enfant logique" peut toujours être concaténé avec son "destination parent"

Exemple :

Structure physique



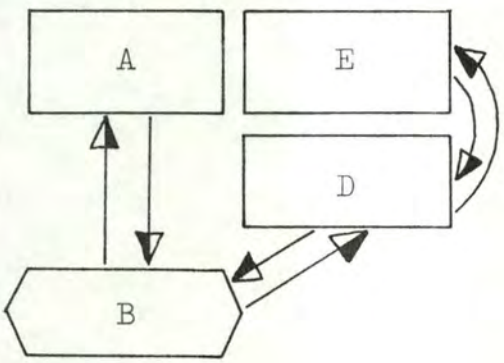
Sous-structures logiques



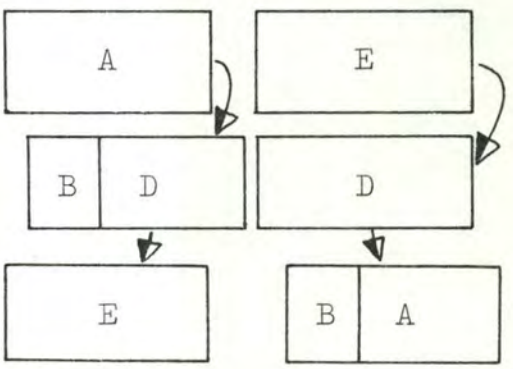
- A partir d'un segment "parent logique" ou d'un segment "parent physique" il est toujours possible de remonter vers le segment racine de la base de données physique.

Exemple :

Structure physique



Sous-structures logiques



- Si un segment "parent logique" est atteint via un segment "enfant logique" les dépendants physiques du segment "parent logique" pourront être repris comme dépendants dans

la structure logique à condition :

- . que la relation logique LC/LP soit la première rencontrée
- . que la relation logique LC/LP soit la répétition de la première relation logique rencontrée
- . que les dépendants physiques soient les variables d'intersection de la relation logique.

- Lorsqu'un segment "enfant logique" est atteint via son segment "parent logique" les dépendants physiques du segment "parent physique" pourront être repris dans la structure logique.

Toutes ces règles paraissent contraignantes, mais la possibilité de définir, dans un même programme, plusieurs vues logiques (PCB) permet de retrouver au niveau de ce programme la structure physique c'est-à-dire en réseau de l'IMS. Nous utiliserons cette possibilité pour définir le langage de manipulation de données dans le chapitre suivant.

4.6.2. Remarques sur les similitudes et les différences d'implémentation par les systèmes IDS et IMS

- Les types de segment racines sont équivalents aux types d'article calculés ; ce sont des entités possédant une propriété identifiante.
- Un type de segment d'intersection ne peut pas être un type de segment racine alors qu'un type d'article d'intersection peut être un type d'article calculé.
- Un type de segment ne peut avoir que deux types de segment Parent et ce type de segment est alors un type de segment "enfant logique" alors qu'un type d'article peut avoir un nombre quelconque de types d'article maître.

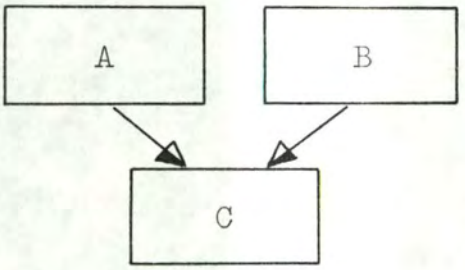
- Le système IMS n'accepte qu'une double relation entre deux types de segment, le système IDS accepte plus de deux relations entre deux types d'article.

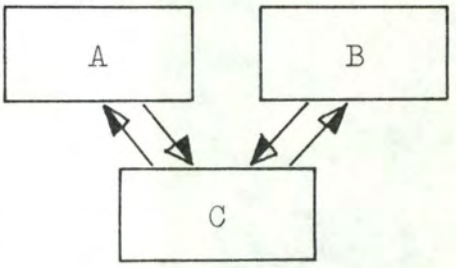
Exemple : Donnons pour chaque réflexion, un exemple graphique

Structure IDS

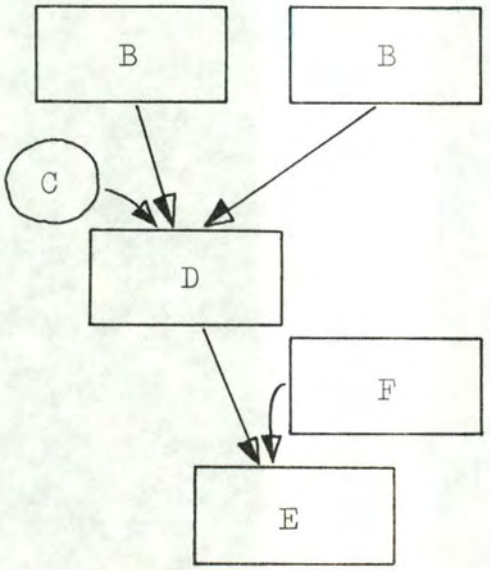
Structure IMS

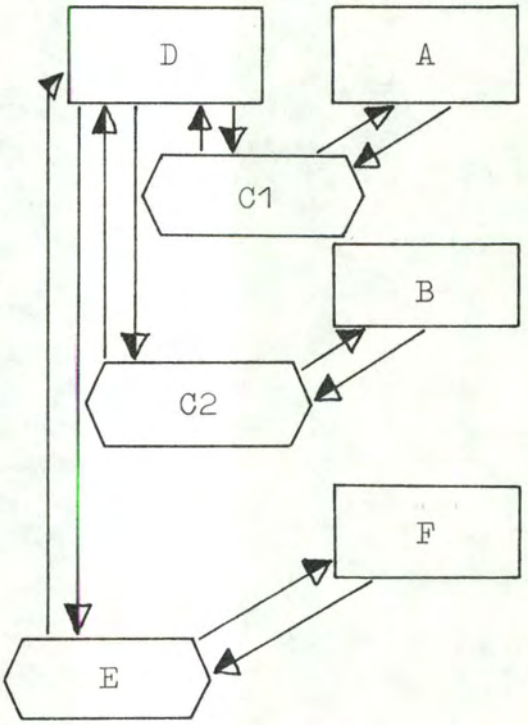
a) Entité d'intersection





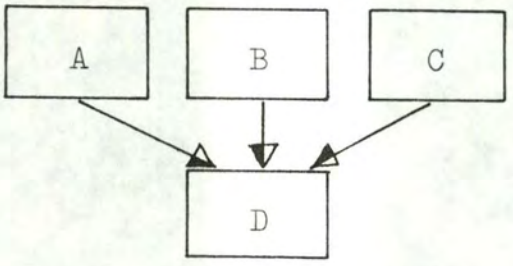
b) Entité possédant une propriété identifiante



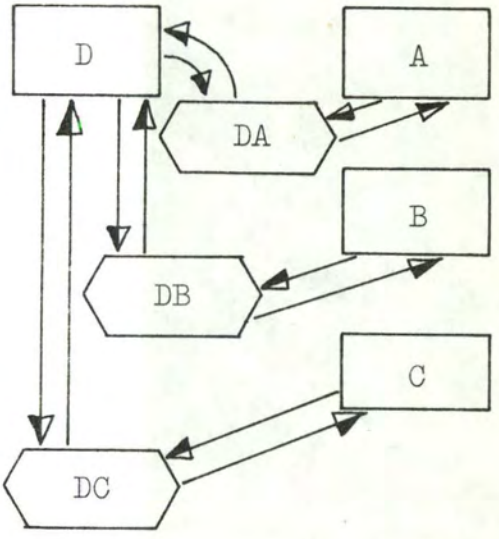


c) Entité but de plus de deux relations

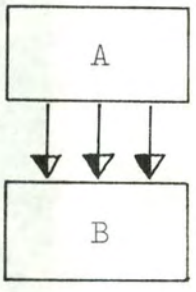
Structure IDS

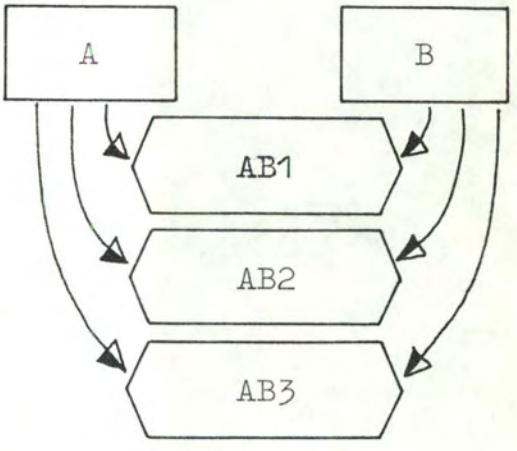


Structure IMS



d) Relations multiples entre deux entités





Pour la compréhension du graphe, les relations inverses bien qu'existantes n'ont pas été représentées.

Remarquons que si les mêmes entités sont en relation dans les deux systèmes, le contenu sémantique de ces relations est différent suivant le système.
Nous aurions pu établir des relations IDS sémantiquement identiques aux relations IMS, la réciproque est fausse.

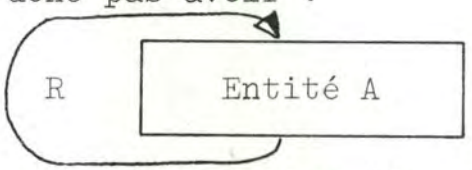
4.6.3. Nombre d'entités au sein d'une relation

Nous avons toujours considéré jusqu'à présent qu'une relation associait deux entités.
Nous examinerons, maintenant, les possibilités offertes par les deux systèmes pour relier un nombre quelconque d'entités dans une même relation.

4.6.3.1. Relation entre les réalisations d'une même entité.

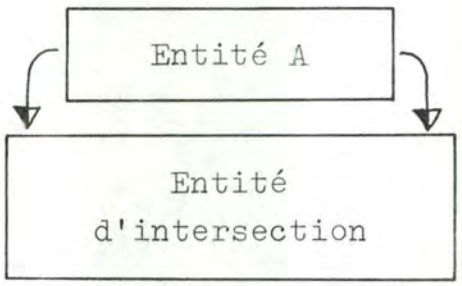
Les boucles sont interdites aussi bien dans le système IDS que dans le système IMS, ceux-ci n'implémentant que des relations à membre obligatoire, alors qu'une boucle ne peut être réalisée que par une relation faible vu la dynamique de la base de données.
Il est, en effet, impossible de déclarer une entité "type d'article maître" ou "type de segment parent" d'une relation et de déclarer cette même entité comme "type d'article détail" ou "type de segment enfant" dans la même relation.

Nous ne pouvons donc pas avoir :



où la relation R est une relation "One to Many" à membre obligatoire.

La seule possibilité de réaliser une relation entre les réalisations d'une même entité est de passer par l'artifice qui nous a permis d'implémenter les relations faibles, c'est à-dire définir une entité d'intersection et deux relations "One to Many" à membre obligatoire entre l'entité (source) et l'entité d'intersection (but).
Nous aurons, en utilisant la possibilité d'établir deux relations entre deux entités

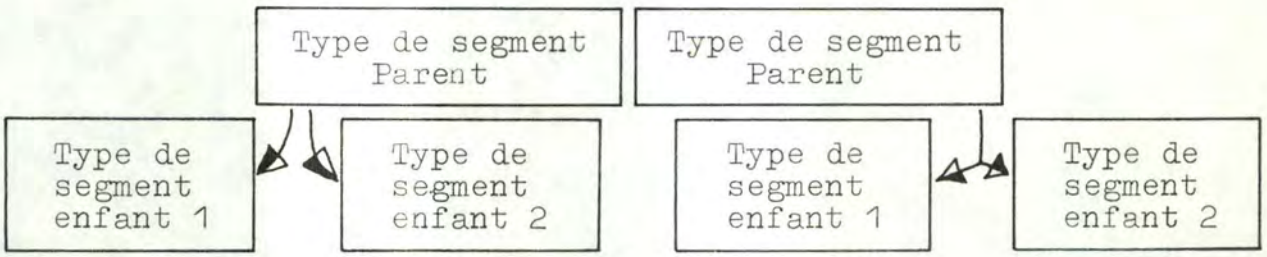


Il est ainsi possible d'obtenir une réalisation d'une entité en relation avec elle-même.

4.6.3.2. Relation entre plus de deux individus

a. Le système IMS

Le système IMS ne connait pas les relations en tant qu'éléments de son langage ; il ne connait que les types de segment et n'offre pas de ce fait la possibilité de réunir dans une même relation plus de deux types de segment.
Nous aurons donc toujours: et non :



b. Le système IDS

Le système IDS donne la possibilité d'avoir une relation entre un type d'article déclaré "Maître" et plusieurs autres types d'article déclarés "détail" de la relation. Cette caractéristique permet au système IDS de réaliser l'économie de une ou plusieurs zones pointeurs dans l'article "maître" ; cependant elle oblige le programmeur à effectuer un test sur le type de l'article auquel il a accédé par une RETRIEVE NEXT.
Cette caractéristique est utilisée par le système IDS pour réaliser le chaînage des articles calculés puisqu'il permet de définir plusieurs types d'articles calculés dans un même bloc physique. Ce bloc physique est définit par l'option PAGE-RANGE.
Un type d'article déclaré appartenir à une PAGE-RANGE, a toutes ses réalisations (tous ses articles) à l'intérieur des limites fixées par ce PAGE -RANGE.
La notion PAGE-RANGE IDS correspond plus ou moins à la notion de base de données physique IMS.

Remarquons que si les mêmes entités sont en relation dans les deux systèmes, le contenu sémantique de ces relations est différent suivant le système.
Nous aurions pu établir des relations IDS sémantiquement identiques aux relations IMS, la réciproque est fausse.

4.6.4. La notion "Structure physique des données" du système IMS représente des bases de données physiques interreliées par des relations logiques. Ces bases de données

physiques ont l'avantage de pouvoir être employées indépendamment les unes des autres par des programmes d'application qui n'accèdent qu'aux segments de types de segment appartenant tous à la même base de données physiques.

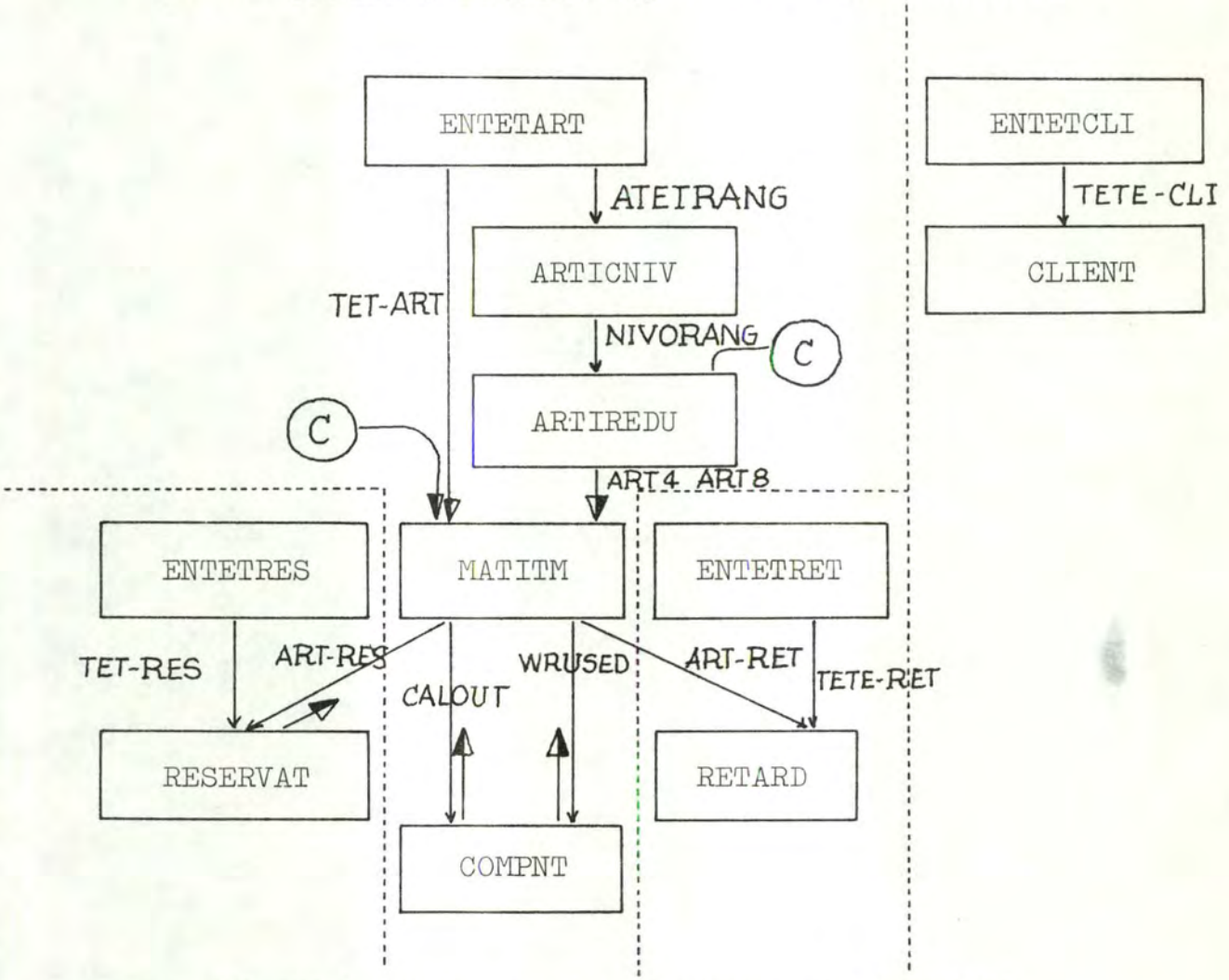
Le même avantage est offert par le système IDS avec l'option "PAGE-RANGE" qui permet une découpe du fichier physique IDS en blocs physiques. Chacun de ces blocs physiques correspond à un "PAGE-RANGE" et contiendra tous les articles de tous les types d'articles déclarés appartenir à ce PAGE-RANGE. Les programmes d'application qui n'accèdent qu'à des articles dont les types sont déclarés appartenir à un ou plusieurs "PAGE-RANGE" ne doivent définir que le contenu de ces PAGE-RANGE.
La notion "PAGE-RANGE" du système IDS, comme la notion "base de données physiques" du système IMS qui, au départ réalise une découpe physique de fichier, seront utilisés pour réaliser une découpe logique de la structure de données de ces systèmes.

4.7. Exemples de graphes de structure

Les exemples présentés sont les bases de données du secteur pharmaceutique et du secteur chimique de la société UCB.

4.7.1. Graphes de structure IDS

4.7.1.1. Le fichier IDS DIPHA (secteur pharmaceutique)



Le pointillé représente la découpe en PAGE-RANGE

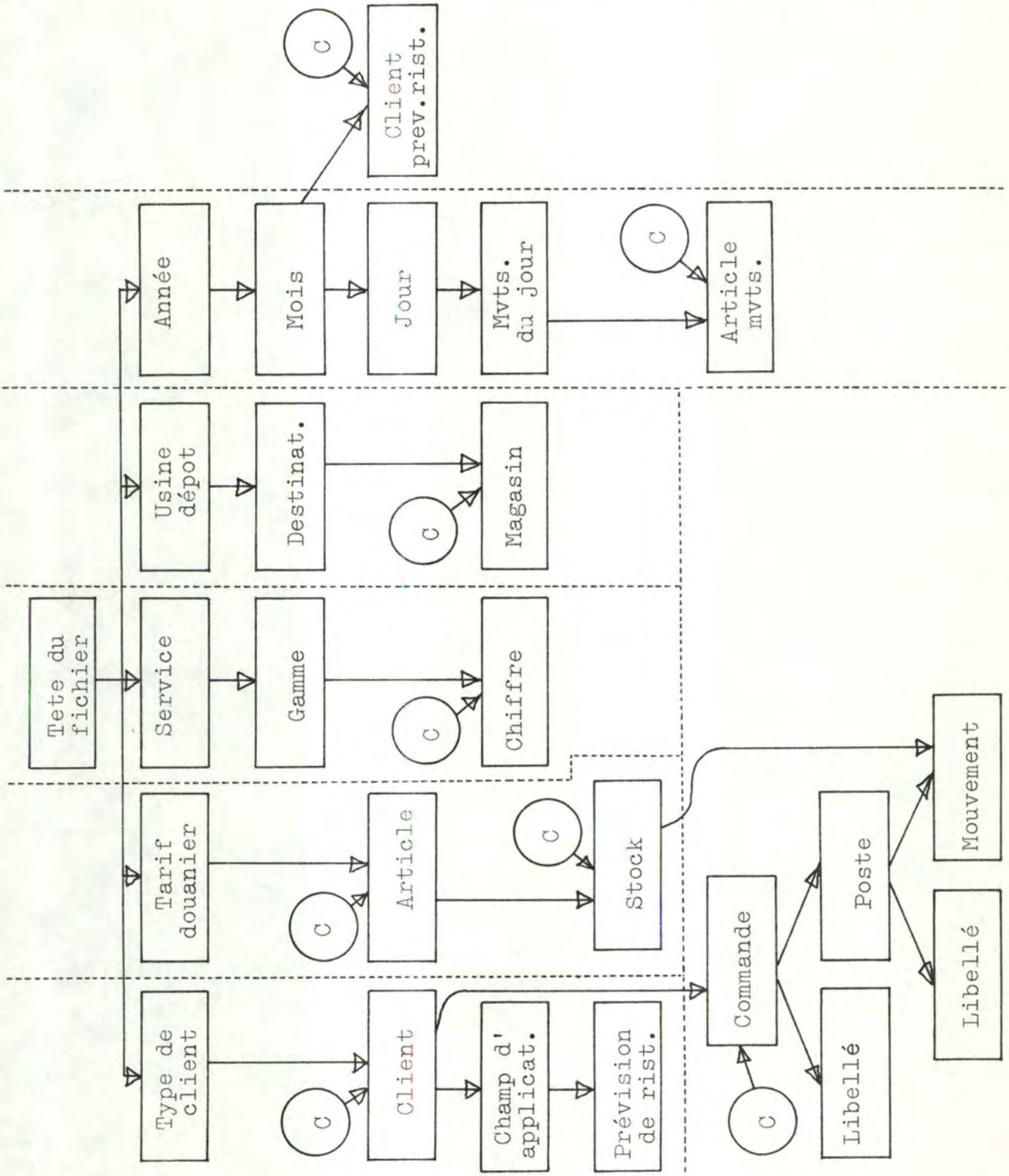
Ce fichier est utilisé dans les applications suivantes :

- Administration des ventes
  - . Facturation
  - . Tenue des stocks
  - . Statistiques

- Gestion de la production
  - . Calcul des prix de revient
  - . Liste des cas d'emploi
  - . Besoins bruts en composants : Planning des achats
  - . Emission des bons de travail (ordre de fabrication)
  - . Suivi de la production : Ecarts techniques
  - . Tenue des stocks composants
  - . Besoins nets en composants

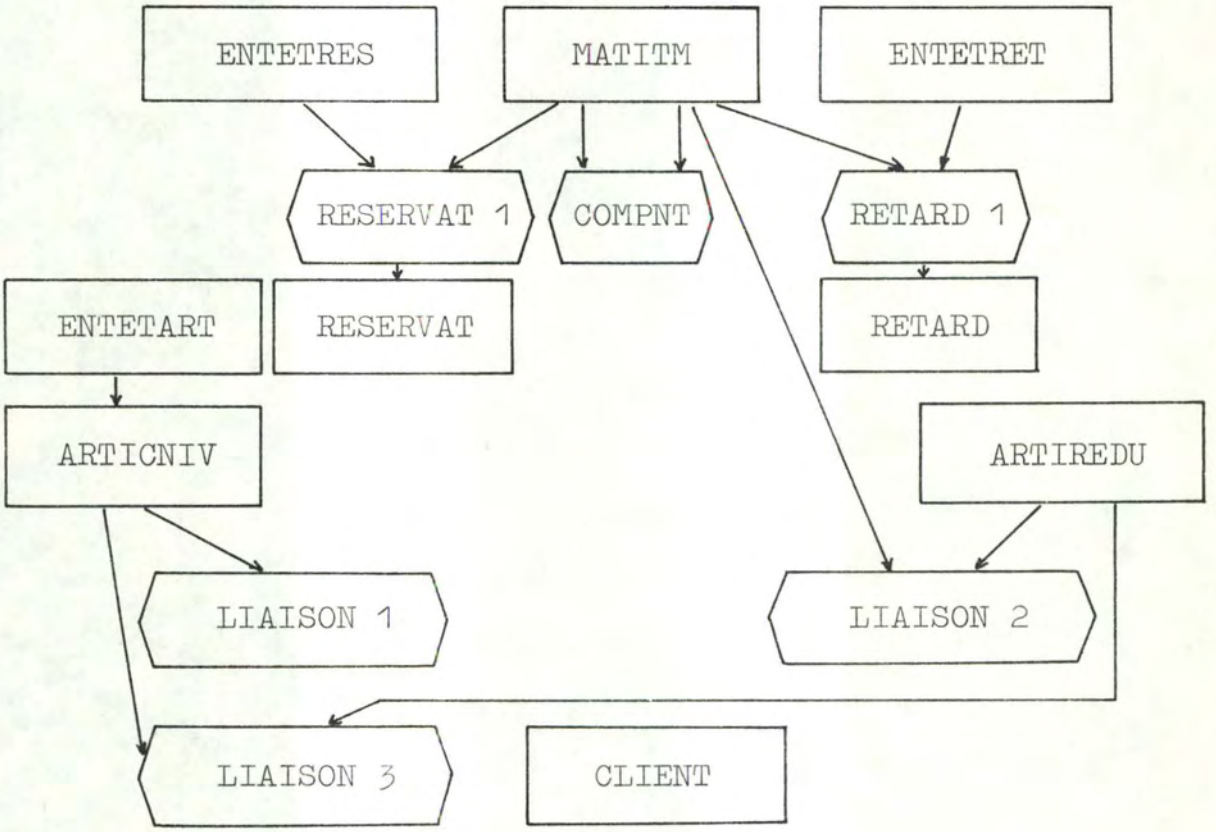
- Inventaire composés/composants de fin d'année.

4.7.1.2. Le fichier IDS SPECHIM (secteur chimique)

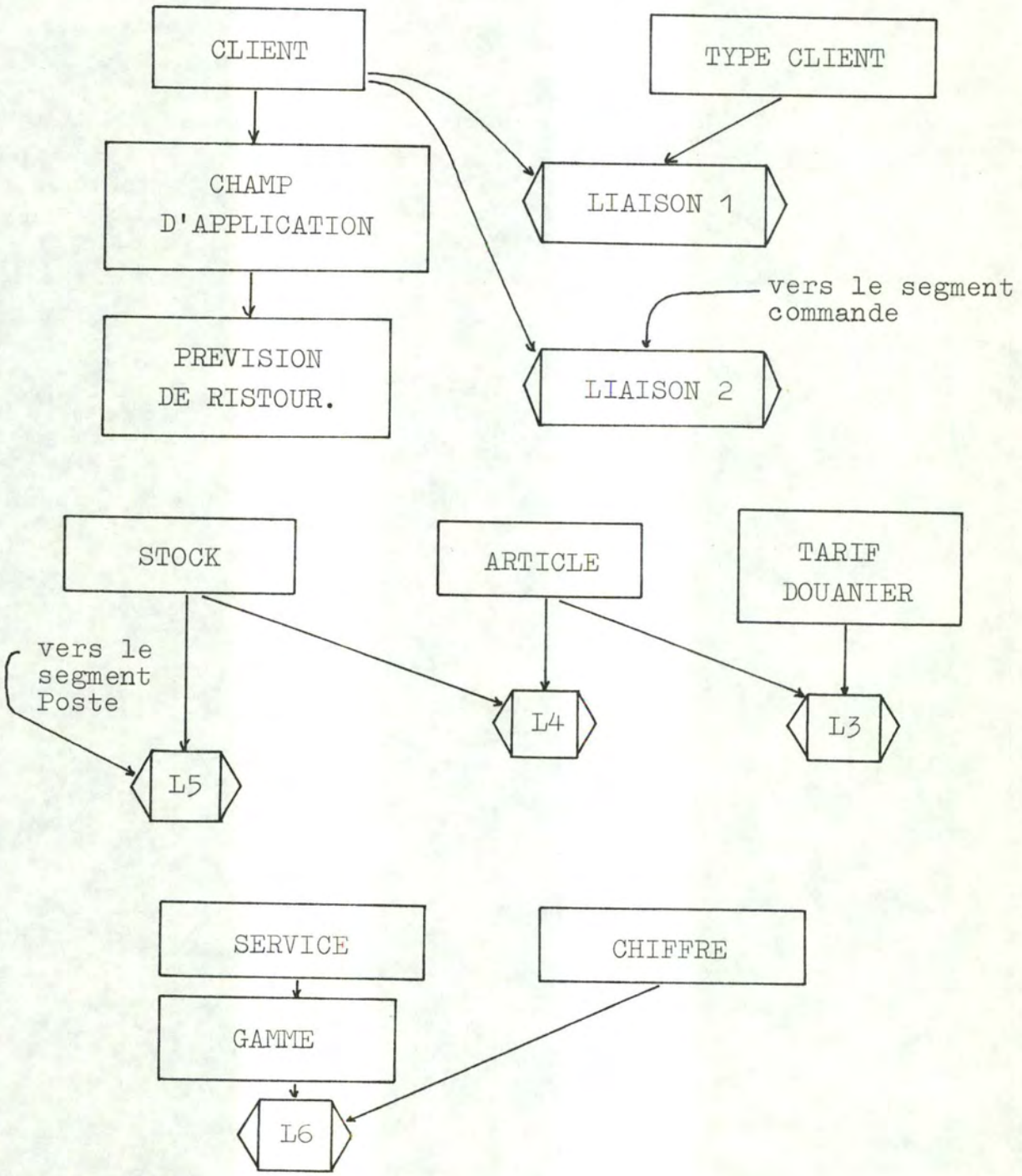


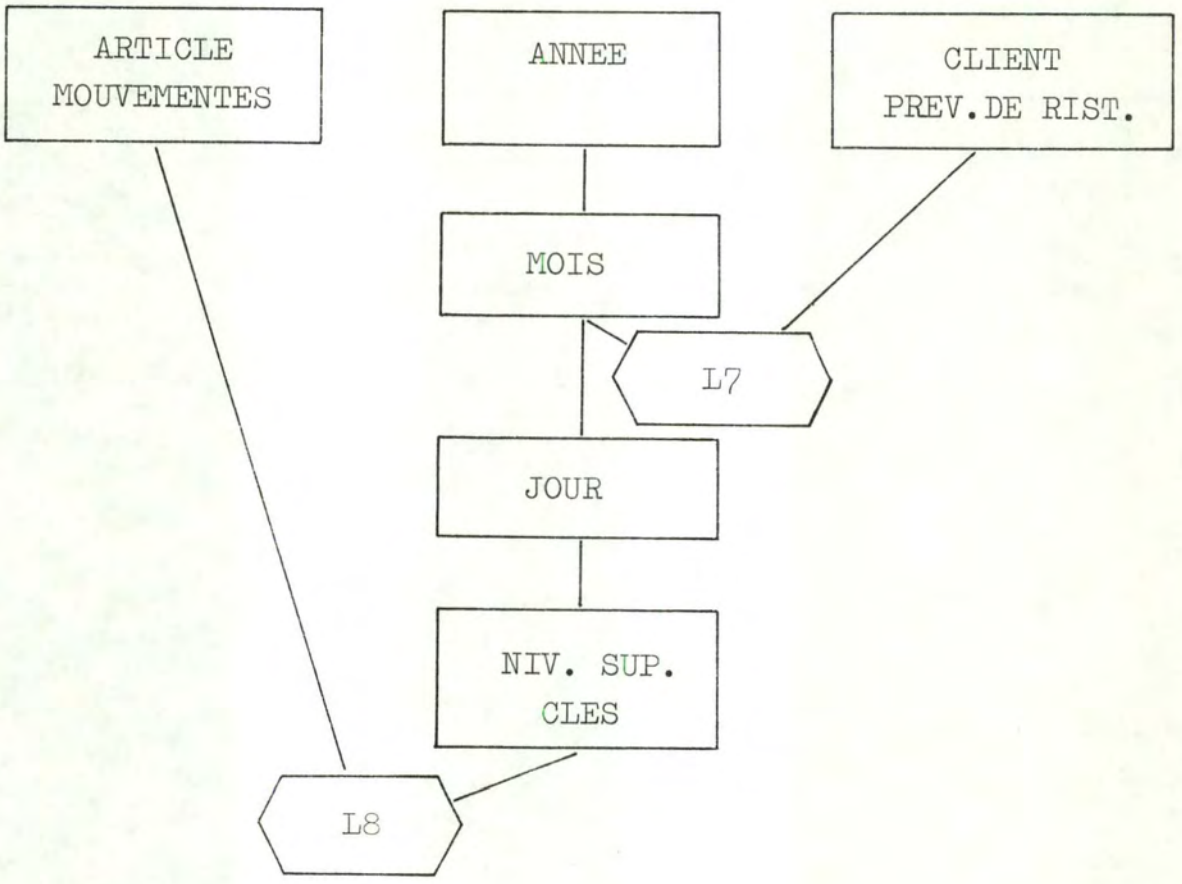
4.7.2. Graphes de structure IMS

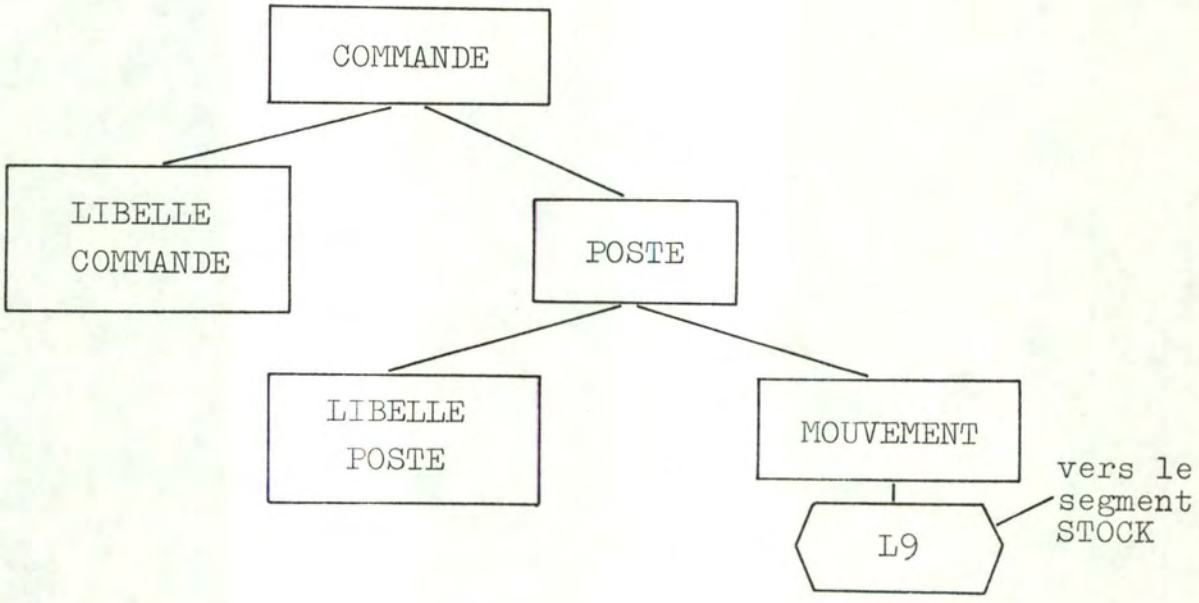
4.7.2.1. Le fichier IMS DIPHA



4.7.2.2. Le fichier IMS SPECHIM




ARTICLE
MOUVEMENTES
ANNEE
CLIENT
PREV.DE RIST.
MOIS
L7
JOUR
NIV. SUP.
CLES
L8


COMMANDE
LIBELLE
COMMANDE
POSTE
LIBELLE
POSTE
MOUVEMENT
vers le
segment
STOCK
L9

5. DEFINITION D'UN MODELE D'ACCES MIS EN OEUVRE PAR LE SYSTEME IDS ET PAR LE SYSTEME IMS.

5.1. Introduction

Le sous-ensemble du modèle d'accès général que nous présentons dans ce chapitre est déduit de l'étude entreprise au chapitre précédent et présente un outil de structuration et d'exploitation de bases de données implémentables à la fois par les systèmes IDS et IMS.

En association avec ce sous-ensemble (par la suite, l'expression modèle d'accès référenciera ce sous-ensemble et non plus le modèle d'accès général), nous définirons un langage de manipulation de données. Nous indiquerons également les techniques d'implémentation retenues dans les deux systèmes réalisant la mise en oeuvre.

5.2. Le modèle d'accès

Rappelons brièvement la définition d'un modèle d'accès.
L'analyste et l'administrateur de la base de données voient la base de données comme étant un ensemble de types d'informations appelés "entités" et un ensemble de relations d'accès entre ces types.

5.2.1. Les entités

Une entité est définie, à chaque instant, par son

nom, l'ensemble de ses propriétés et l'ensemble des relations d'accès auxquelles elle participe.

L'ensemble des propriétés caractérisant une entité est fixe, c'est-à-dire que nous ne pouvons pas ajouter ou supprimer des propriétés à une entité sans réorganiser la base de données. En plus toutes les valeurs de chaque propriété contient toujours un même nombre de caractères. Ces valeurs seront tronquées ou complétées par des caractères de remplissage, automatiquement par le système.
Les caractères de remplissage sont les caractères blanc ou zéro suivant la définition du type alphanumérique ou numérique des valeurs des propriétés, le remplissage s'effectuant à droite ou à gauche pour que ces caractères soient ou non significatifs.

Le système ne gènera que des entités dont les réalisations ont même longueur. Nous profiterons de l'indépendance des propriétés, qui ne servent pas aux modules de gestion du système pour réaliser des contrôles ou des accès, par rapport au système, pour permettre l'ajoute de propriétés à une entité en définissant des propriétés de réserve qui ne prendront de signification que lors d'une évolution du système et uniquement au niveau des programmes d'application.

En résumé, toutes les réalisations d'une entité donnée, ont la même longueur (le même nombre de caractères) ; les propriétés qui interviennent dans les procédures de contrôle et d'accès du modèle seront fixées à la définition de l'entité, les autres propriétés ne prendront de signification qu'au niveau du programme utilisateur.

L'ensemble des entités de la base de données est

subdivisée en trois sous-ensembles disjoints :

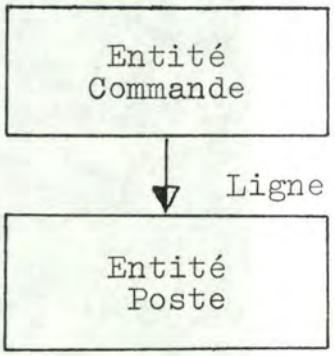
- L'entité en-tête
- Les entités racines
- Les entités dépendantes

L'étude de ces entités ne pouvant se réaliser indépendamment de l'ensemble des relations d'accès auxquelles elles participent, nous présenterons d'abord l'étude des relations d'accès.

5.2.2. Les relations d'accès

5.2.2.1. Une relation d'accès est à tout instant, définie par son nom qui doit être unique et par l'ensemble des couples qui constituent les occurrences de la relation.
Soit R (A,B) une relation d'accès définie de l'entité source A vers l'entité but B ; si le couple (a,b) est une occurrence de la relation R, alors il existe un moyen d'accéder à la réalisation b de l'entité but B à partir de la réalisation a de l'entité source A.
Pour chaque réalisation a de l'entité source A, la relation d'accès R détermine un sous-ensemble éventuellement vide des réalisations de l'entité but B; ce sous-ensemble est l'image de la réalisation a de l'entité A dans l'ensemble des réalisation b_i de B par la relation R et nous le noterons R (a).

Exemple :



La relation d'accès "Ligne", établie de l'entité "Commande" vers l'entité "Poste" exprime la possibilité d'accéder à un ensemble de réalisations de l'entité "Poste" à partir d'une réalisation de l'entité "Commande".

Soient les relations d'accès R(A,B) et S (B,A), la relation S (B,A) sera dite relation inverse de la relation R (A,B), si nous avons, à tout instant :

$$b \in R(a) \Longleftrightarrow a \in S(b)$$

où - R(a) est l'image de la réalisation a de l'entité A dans l'ensemble des réalisations de l'entité B par la relation R.

- S(b) est l'image de la réalisation b de l'entité B dans l'ensemble des réalisations de l'entité A par la relation S.

C'est-à-dire, si à partir d'une réalisation a de l'entité A la relation R donne accès à une réalisation b de l'entité B, alors la relation S donne accès à la réalisation a à partir de la réalisation b et réciproquement.
Toutes les relations d'accès retenues dans ce modèle d'accès possède une relation inverse fournie par le système.

5.2.2.2. Caractéristiques des relations

A tout instant, une relation d'accès sera une relation du type :

- One to One à membre obligatoire
- One to Many à membre obligatoire
- One to One faible
- One to Many faible
- Many to Many faible

5.2.2.3. Ordre induit par une relation d'accès

Nous avons vu que par une relation d'accès R, nous définissions un moyen d'accéder à partir d'une réalisation a

de l'entité source A, à chaque réalisation b_i de l'entité but B appartenant à l'ensemble R(a).

Si nous considérons l'ordre sous lequel nous accèdons aux éléments de cet ensemble R(a), nous établissons toujours sur cet ensemble une structure d'ordre.
La structure d'ordre sera toujours déclarée explicitement par l'utilisateur.
Trois possibilités sont offertes pour définir cette structure.

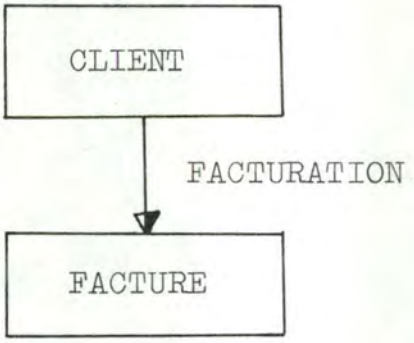
a) Associer à la relation des propriétés d'ordre de l'entité but ; ces propriétés devront être telles qu'elles définissent une zone continue dans la définition de l'entité, d'une longueur maximum de 256 caractères et que nous appellerons zone de séquence.
Si ces propriétés sont identifiantes ou localement identifiantes le système établit un ordre ascendant strict.
Si ces propriétés sont des propriétés d'ordre simple, elles établissent une partition de l'ensemble R(a), chaque sous-ensemble $R_s(a)$ de la partition correspondant à une valeur de la zone de séquence et le système établit un ordre ascendant strict de ces sous-ensembles.
La structure d'ordre établie sur chaque sous-ensemble $R_s(a)$ le sera suivant les deux autres possibilités.

b) Choisir l'ordre d'accès en fonction de l'instant de création des occurrences de l'entité but
La relation d'accès peut être
   - LAST IN - FIRST OUT
     La nouvelle réalisation est placée comme premier élément de l'ensemble R(a) ou $R_s(a)$.
   - LAST IN - LAST OUT
     La nouvelle réalisation est placée comme dernier élément de l'ensemble R(a) ou $R_s(a)$.

Cette possibilité n'est souvent qu'une option d'insertion sans portée sémantique car l'instant de création est l'instant de création physique et non pas l'instant de création logique que connaît l'utilisateur.

c) Laisser au programmeur, la gestion de la structure d'ordre en lui permettant de placer la nouvelle réalisation où il le désire.
Dans ce cas la nouvelle réalisation sera placée devant la dernière réalisation de l'ensemble R(a) ou $R_s(a)$ connue du système. Si celui-ci n'a pas en mémoire une réalisation appartenant à cet ensemble, il considère alors la relation d'accès comme étant du type LAST IN - FIRST OUT.

Exemple : Considérons, dans un environnement compte client, les deux entités "client" et "facture" et la relation "facturation".
Graphiquement nous aurons :



Si l'utilisateur veut accéder aux factures restant ouvertes dans le compte d'un client donné, dans l'ordre de leur date d'échéance (majeur) et de leur condition de paiement (mineur), il définira l'entité "facture" en plaçant l'une près de l'autre et dans l'ordre les propriétés "Date d'échéance" et "condition de paiement", choisira la première possibilité en associant l'ensemble des deux propriétés à la

relation "facturation" qui devient une relation d'ordre simple et définira pour les sous-ensembles de réalisations de "facture" associés à chaque valeur de la zone de séquence l'option d'insertion LAST IN - LAST OUT.
Si l'utilisateur avait définit une structure d'ordre en fonction des propriétés " date d'échéance ", "condition de paiement" et "numéro de facture" dans cet ordre, la relation aurait été une relation d'ordre strict, la propriété "numéro de facture" étant une propriété localement identifiante.

Rappelons qu'une propriété localement identifiante n'est pas telle que ses valeurs établissent une correspondance biunivoque entre chaque valeur et chaque réalisation de l'entité. En reprenant l'exemple précédent, la valorisation de la propriété "numéro de facture" ne définit pas une et une seule réalisation de l'entité "facture", il faut en plus donner une valeur à la propriété "numéro de client" identifiante pour l'entité client.

5.2.2.4. Dynamique des relations et règles du système de gestion

Nous avons vu qu'une relation déclarée d'un certain type à la définition de la base de données, gardait, à tout instant, ce type et que l'évolution de l'ensemble des occurrences (ajoute ou suppression d'occurrence) devait se réaliser en accord avec la déclaration de type initiale. Le système contrôlera cette évolution pour maintenir la cohérence de la base de données.

Vu les techniques d'implémentation retenues pour

réaliser les relations, aussi bien dans le système IDS que dans le système IMS, la dynamique des relations est complètement contenue dans la dynamique des entités.
Rappelons que nous avons appelé "entité d'intersection", une entité but de plus d'une relation à membre obligatoire.

- La suppression ou l'addition d'occurrences de relation d'accès faible se résoud, au niveau du système, par la suppression ou l'addition d'une entité d'intersection.
  Les modifications du genre remplacer une réalisation source ou une réalisation but pour une occurrence de relation se résoud au niveau du programmeur par la suppression de l'occurrence suivie par la création de la nouvelle occurrence, ce qui se réalisera au niveau du système, par la suppression de la réalisation de l'entité d'intersection établissant l'occurrence de la relation à modifier, suivie de la création d'une réalisation de cette entité réalisant la nouvelle occurrence demandée.

- La suppression ou l'addition d'occurences de relation à membre obligatoire est réalisée par la suppression ou l'addition de réalisations d'entité.
  Les modifications du genre remplacer une réalisation source ou une réalisation but par une autre, ne sont pas réalisées par le système et il est dangereux de laisser le programmeur résoudre le problème par suppression - création vu les implémentations de la suppression d'occurence à membre obligatoire.
  Ceci est une restriction du système d'implémentation IMS, le système IDS autorisant lui le remplacement d'une réalisation source par une autre dans les occurrences de relations "One to Many" à membre obligatoire.

5.2.3. L'entité en-tête

Une entité en-tête sera caractérisée par l'ensemble des relations auxquelles elle participe :

- Elle ne peut être une entité but d'aucune relation.
- Les seules relations dont elle peut être la source, sont des relations du type "One to Many" à membre obligatoire dont l'entité but est une entité "racine". Ces relations permettront l'accès dans un ordre donné à toutes les réalisations des entités racines.
- Elle ne possèdera aucune propriété et aucune action ne sera permise sur les réalisations de cette entité. Seul le parcours des relations "One to Many" est autorisé pour obtenir les réalisations des entités racines.

Une entité en-tête est représentée, dans le système IDS par un type d'article primaire, et dans le système IMS par la base de données index correspondante aux types de segment racine en organisation HISAM ou HIDAM et par le "Root Anchor point" en organisation HDAM.
Elle est nécéssaire car si pour le système IMS, il est possible d'accéder aux segments racines en séquentiel, pour le système IDS, il n'est pas possible d'avoir un accès séquentiel sur les articles calculés sans qu'ils ne soient articles "détail" d'une chaîne.

5.2.4. Les entités racines

5.2.4.1. Définition

Une entité racine est définie par son nom, l'ensemble de ses propriétés, l'ensemble des relations auxquelles elle

participe et les opérations que l'on peut effectuer sur ses réalisations.

- Elle possèdera toujours une propriété identifiante.
- Elle sera l'entité but d'une et d'une seule relation "One to Many" à membre obligatoire dont l'entité source sera l'entité en-tête.
- Si on veut accéder aux réalisations d'une entité racine dans un ordre donné, cet ordre ne peut être qu'ascendant sur la propriété identifiante qui sera alors associée à la relation entre l'entité en-tête et l'entité racine pour établir l'accès ordonné.
- Elle pourra être l'entité d'un nombre quelconque de relation de tout type et l'entité but de plusieurs relations faibles.

5.2.4.2. Dynamique des entités racines

a. A la création d'une réalisation d'entité racine, le programmeur devra donner le nom de l'entité et avoir spécifier une valeur de la propriété identifiante.
Si une réalisation existe déjà pour cette valeur de la propriété identifiante, le système refusera la création, sinon il insérera la nouvelle réalisation et créera l'occurrence de la relation la reliant à la réalisation de l'entité en-tête en respectant l'ordre d'insertion demandé par cette relation.

b. La suppression d'une réalisation d'entité racine entraînera la suppression de toutes les occurrences de relations auxquelles elle participait. Ces suppressions dans le cas de relations à membre obligatoire entraîne la suppression des réalisations des entités but dont elle était la source et ainsi de suite en cascade à travers toute la base de données.

Ce traitement sera entièrement pris en charge par le système.

Exemple : Si nous reprenons l'exemple précédent, la suppression d'un client entraînera la suppression de toutes les factures de ce client.

c. L'utilisateur pourra modifier les valeurs des propriétés d'une entité racine sauf les valeurs de la propriété identifiante ce qui obligerait le système à revoir les relations d'accès et à contrôler l'unicité de la nouvelle valeur.

Exemple : Si la propriété identifiante de l'entité client est le code client, l'utilisateur ne peut donner une autre valeur à ce code pour un client existant dans la base de données.

5.2.5. Les entités dépendantes

5.2.5.1. Définition

Une entité dépendante est définie par son nom, l'ensemble de ses propriétés, l'ensemble des relations auxquelles elle participe et par les opérations que l'on peut effectuer sur ses réalisations.

- Elle peut avoir des propriétés localement identifiantes ou d'ordre.
- Elle sera l'entité but d'au moins une et ou plus deux relations "One to Many" à membre obligatoire.
  Pour chacune de ces relations, l'entité source sera soit une entité racine, soit une entité dépendante.
  Une des deux relations sera déclarée relation principale de recherche et c'est par cette relation que le système

effectuera les accès implicites à cette entité.

- Si elle est l'entité but d'une et d'une seule relation "One to Many" à membre obigatoire, alors :
  - . elle peut être l'entité but ou source d'un nombre quelconque de relations faibles
  - . elle peut être l'entité source d'un nombre quelconque de relations à membre obligatoire.
- Si elle est l'entité but de deux relations "One to Many" à membre obligatoire, alors :
  - . elle ne peut pas être l'entité but ou source d'une relation faible
  - . elle peut être l'entité source d'un nombre quelconque de relations à membre obligatoire mais aucune de ces relations ne peut avoir une entité but qui soit aussi l'entité but d'une autre relation à membre obligatoire.

L'ensemble des relations principales de recherche qui permet d'accéder à partir d'une entité racine à une entité dépendante sera appelé chemin principal de recherche ou chemin hiérarchique.

Nous appellerons clé concaténée d'une entité dépendante l'ensemble formé de la propriété identifiante et des propriétés localement identifiantes de l'entité racine et des entités dépendantes du chemim hiérarchique de cette entité.

Dire qu'une entité possède une clé concaténée, suppose que toutes les entités de son chemin hiérarchique possède une propriété localement identifiante, l'entité racine possédant toujours une propriété identifiante.

Donc toutes les entités, se trouvant sur le chemin hiérarchique d'une entité possédant une clé concaténée, possèdent une clé concaténée.

- Toute entité dépendante, qui est l'entité but ou source d'une relation faible, doit posséder une clé concaténée.
- Toute entité dépendante, qui est l'entité source d'une relation à membre obligatoire dont l'entité but est l'entité but d'une autre relation à membre obligatoire, doit posséder une clé concaténée.

Remarquons que les valeurs de la clé concaténée d'une entité sont telles qu'il existe une correspondance biunivoque entre ces valeurs et les réalisations de l'entité, c'est-à-dire pour chaque valeur de cette clé, nous aurons au plus une réalisation de l'entité.

5.2.5.2. Dynamique des entités dépendantes

a. Lors d'une demande de création d'une réalisation d'une entité dépendante, le programmeur devra, soit par un positionnement préalable, soit par le contenu de l'ordre de création, spécifier les réalisations d'entités sources de relations à membre obligatoire pour lesquelles l'entité est l'entité but.

Exemple : Pour la création d'une réalisation de l'entité "facture", l'utilisateur doit préciser à quelle réalisation de l'entité "client", elle doit être reliée.

Le système refusera la création si ces renseignements sont faux ou incomplets ou si la création d'une nouvelle réalisation n'est pas compatible avec les règles données pour une éventuelle structure d'ordre.

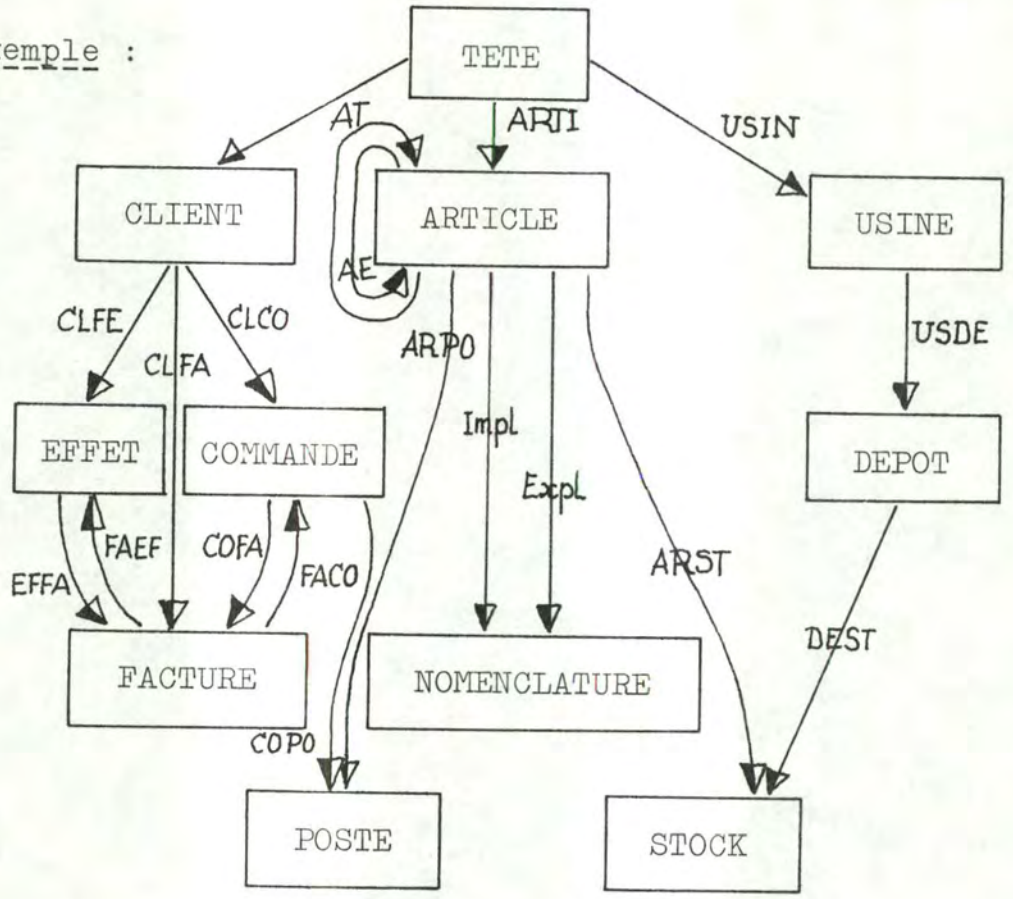
b. La suppression d'une réalisation d'une entité dépendante entraînera la suppression de toutes les occurrences de relations auxquelles elle participait.
Ces suppressions, dans le cas de relation à membre obligatoire, entraîne la suppression des réalisations des entités but dont elle était la source.
Elles se propagent dans toute la base de données.

5.2.6. Le graphe de structure

Comme nous l'avons exprimé précédemment, le modèle d'accès peut se représenter par un graphe orienté, dont les sommets sont les entités représentées sous forme de rectangle et dont les flèches sont les relations d'accès.
En plus des règles de définition des différentes entités, le graphe de structure doit suivre certaines règles pour être cohérent :
Ces règles sont :

- Tous les noms des entités sont distincts.
- Tous les noms de relations sont distincts, les relations inverses des relations à membre obligatoire ne portent pas de nom ; les relations inverses des relations faibles doivent posséder un nom distinct du nom de la relation.
- Le graphe des relations à membre obligatoire ne peut contenir de cycle, c'est-à-dire doit pouvoir être décomposable en niveau, le nombre de niveaux ne peut être supérieur à 15.

Exemple :



Les relations FAEF et FACO sont des relations "Many to Many" faibles dont les relations inverses sont EFFA et COFA.
La relation AT (article remplacé) est une relation "One to Many" faible dont la relation inverse est AE (article remplacant)
Toutes les autres relations sont des relations "One to Many" à membre obligatoire.
L'entité "tête" est l'entité en-tête du fichier.
Les entités "client", "article" et "usine" sont des entités racines.
Les autres entités sont des entités dépendantes.
L'entité "effet" et l'entité "commande" doivent posséder une propriété localement identifiante puisqu'elles sont membre de relations faibles.

5.2.7. Remarque sur l'implémentation

5.2.7.1. L'entité en-tête

A la réalisation d'une entité en-tête correspond une base de données physique du système IMS ou un sous-fichier du système IDS.

- Système IMS : En organisation HISAM et HIDAM, une entité en-tête représente la base de données Index.
  En organisation HDAM, une entité en-tête représente la notion IMS "Root Anchor Point" définit aussi par la base de données physique.

- Système IDS : Une entité en-tête correspond à un type d'article primaire que nous définirons par sous-fichier IDS pour pouvoir accéder aux articles calculés de ce sous-fichier en séquence.

5.2.7.2. Les entités racines

Une entité racine correspondra :
- dans le système IMS, à un type de segment racine
- dans le système IDS, à un type d'article calculé

En effet, une entité racine doit posséder une propriété identifiante, hors seuls ces types de segment ou types d'article autorisent la définition d'une propriété identifiante dans leur description.

5.2.7.3. Les entités dépendantes

Une entité dépendante correspondra :
- dans le système IMS, à un type de segment dépendant
- dans le système IDS, à un type d'article définit par l'option IDS "RETRIEVAL VIA" nom de chaîne CHAIN.

La relation principale de recherche sera, dans le système IMS, la relation physique et, dans le système IDS le type de chaîne définit par "nom de chaîne" de l'option "RETRIEVAL VIA".

5.2.7.4. Les relations

Nous ne présenterons pas l'implémentation des relations, celles-ci ayant été largment exposée dans le chapitre précédent de l'étude.

5.3. Langage de manipulation

5.3.1. Introduction

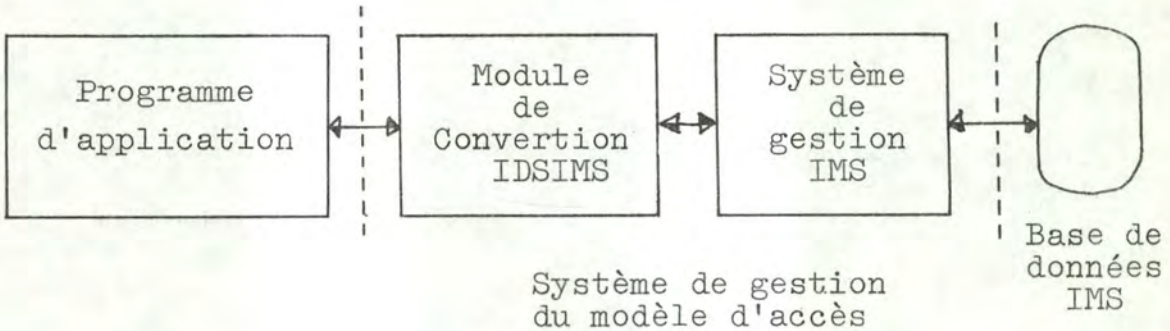
a. Le langage de manipulation de la base de données doit mettre à la disposition du programmeur des outils lui permettant de manipuler les informations contenues dans cette base.
Les ordres de ce langage seront appelés des primitives car ils doivent être exécuter complètement avant que le programme d'application ne puisse à nouveau intervenir.
L'interrogation de la base de données peut se réaliser suivant

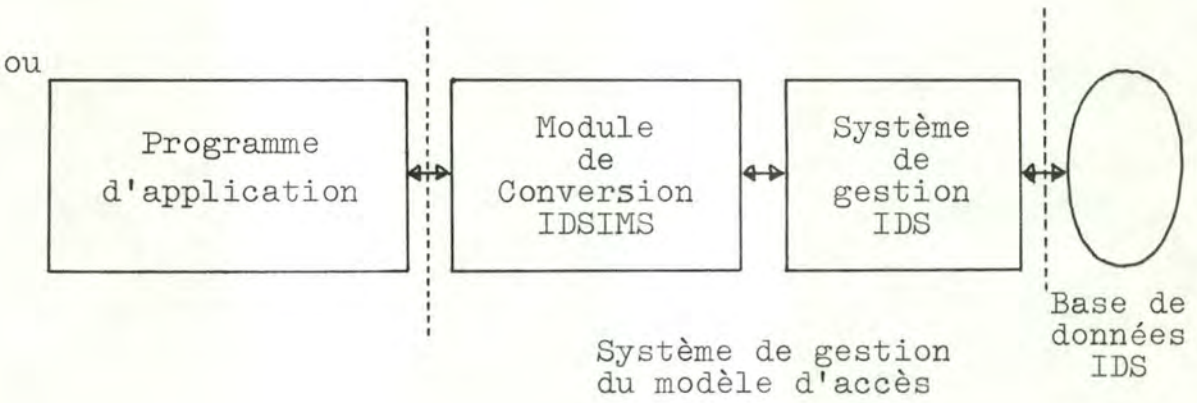
deux possibilités : - -l'interrogation ponctuelle
- l'interrogation de masse

Nous ne considérons que des primitives réalisant l'interrogation ponctuelle qui, par définition, ne fournit au programme d'application qu'une et une seule réalisation d'entité par ordre donné.

Les systèmes IDS et IMS proposant un langage de manipulation composé de primitives simples, nous avons adopté un langage du même type où les primitives seront des appels à un module contenant la description de la base de données et toutes les informations nécéssaires à l'analyse de ces appels et à leur transformation en ordres acceptés par les modules de gestion du système adopté pour réaliser l'implémentation.
Ce module de conversion forme un tampon entre les programmes d'application et le système de gestion de la base de données qui devient alors complètement transparent pour ces programmes.

Schématiquement, nous aurons donc :





Le langage hôte du programme d'application peut être tout langage compatible avec la fonction CALL simple c'est-à-dire sans réinitialisation du module appelé.
En liaison avec le système IDS, le module de convertion doit être écrit en langage COBOL car ce langage est le seul que ce système accepte comme langage hôte.
En liaison avec le système IMS, le module de convertion peut être écrit en langage COBOL, PL1, ou ASSEMBLER.
Tous les exemples que nous donnerons, le seront en langage COBOL.

b. Le langage de manipulation devra posséder des primitives capables de réaliser les six fonctions suivantes :

- Lecture d'une réalisation d'entité

Le programmeur d'application doit avoir la possibilité de retrouver une réalisation en donnant le nom de l'entité et la valeur des propriétés identifiantes, ou en donnant le nom de l'entité, les valeurs des propriétés identifiantes et localement identifiantes de toutes les entités du chemin hiérarchique (chemin principal de recherche) reliant l'entité racine à l'entité dont une réalisation est recherchée, ou encore par des techniques de recherche pas à pas basées sur les relations d'accès telles que :

. accéder à la première réalisation d'une chaîne
. accéder à la réalisation suivante dans une chaîne
. accéder à la dernière réalisation d'une chaîne
. accéder à la réalisation source d'une chaîne
. .....

- Modification d'une réalisation d'entité

Le programmeur d'application doit pouvoir, après avoir retrouvé une réalisation, en modifier les valeurs des propriétés et la replacer dans la base de données.
Rappelons ici que lors de la présentation de la dynamique des entités, nous avons exclu la possibilité de modifier le contenu des propriétés identifiantes, localement identifiantes et d'ordre simple.

- Insertion d'une nouvelle réalisation d'entité

L'addition d'une nouvelle réalisation d'une entité doit être possible. Le système doit s'occuper automatiquement de la création de l'occurrence des relations à membre obligatoire dans lesquelles l'entité est l'entité but.
En plus, pour réaliser l'insertion, il doit contrôler l'évolution cohérente de la base et éventuellement rejeter l'insertion pour garder cette cohérence. Il doit aussi respecter les contraintes d'un accès ordonné si des relations d'ordre ont été spécifiées.

- Suppression d'une réalisation d'entité

La suppression d'une réalisation doit se répercuter dans la base de données conformément à la dynamique des entités et des relations.
Elle entraînera :

. La suppression de toutes les occurrences des relations faibles où elle intervient comme réalisation but ou source.

. La suppression de toutes les occurrences des relations à membre obligatoire où elle intervient comme réalisation but.
. La suppression de toutes les occurrences des relations à membre obligatoire où elle intervient comme réalisation source avec comme conséquence la suppression automatique de toutes les réalisations but de ces occurrences et la propagation de ces suppressions automatiques à travers toute la base de données.

- Insertion d'une nouvelle occurrence de relation faible

Le programmeur d'application doit pouvoir insérer une nouvelle occurrence de relation faible entre deux réalisations d'entités associées par cette relation.
L'insertion doit pouvoir se réaliser en donnant le nom de la relation et les deux réalisations à relier.

- Suppression d'une occurrence de relation faible

La suppression d'une occurrence de relation faible doit pouvoir être réalisée en donnant le nom de la relation et les deux réalisations d'entités qui étaient reliées par cette occurrence.

c. A chaque primitive sera associée une liste de paramètres, ceux-ci dépendront de la fonction que doit remplir la primitive. Nous pourrons avoir :
- Le nom de la zone entrée/sortie du programme utilisateur. Cette zone contiendra :
. Avant une primitive d'insertion ou de modification de réalisation d'entité, la réalisation de l'entité à créer ou à modifier.
. Après une primitive de lecture de réalisation d'entité la réalisation de l'entité recherchée si l'ordre s'est normalement exécuté.

- Des qualificateurs qui détermineront sur quelle réalisation d'entité ou occurrence de relation le programme désire opérer. Ces qualificateurs pourront être :
  . Le nom de l'entité dont une réalisation est recherchée, modifiée, insérée ou supprimée avec ou sans la valeur de la propriété identifiante ou localement identifiante de cette entité.
  . Le nom de la relation dont le programmeur désire obtenir une occurrence.

La fonction de la primitive sera définie par le point d'entrée dans le module de conversion de l'appel.
En plus il existera une zone de communication entre le programme utilisateur et le module de conversion.
Cette zone aura une longueur de 4 caractères et s'appellera "STATUS-ERROR".
Elle contiendra sous forme de code, le diagnostic de l'opération qui viendra d'être réalisée sur la base de données.
Nous aurons donc, dans cette zone, les informations se rapportant à la façon dont la primitive CALL se sera exécutée.
Après une primitive de manipulation, il faudra toujours tester le "STATUS-ERROR" pour voir si cette primitive a été exécutée correctement ou non.

5.3.2. Primitives d'initialisation et de clôture

Le programmeur, à l'aide de ces primitives, va alerter le système de son intention d'accéder aux réalisations d'entités et aux occurrences de relations contenues dans une base de données ou de cesser tout accès à cette base de données.

- Primitive définissant une intention d'accès à une base de données

  OPEN nom de base de données -1, nom de base de données -2, ...

Cette primitive doit être exécutée avant l'exécution d'une autre primitive.
Elle correspondra à une instruction OPEN SUB-FILE IDS avec en plus lecture de la réalisation des entités en-tête intervenant dans ce SUB-FILE.
Dans le système IMS, elle correspondra à la Call ENTRY "DLITCBL" USING nom de PCB-1, nom de PCB-2,...

- Primitive spécifiant la fin des accès à une base de données.
CLOSE nom de base de données -1, nom de base de données -2, ...
Un programme d'application ne peut avoir qu'une primitive CLOSE qui doit être écrite en fin de programme.
Elle correspondra à une instruction CLOSE SUB-FILE IDS, et à une instruction GOBACK IMS.

5.3.3. Accès aux réalisations d'une entité

5.3.3.1. Accès à une réalisation indépendamment des opérations réalisées précédemment.

a. La primitive réalisant cette fonction sera la primitive RETRIEVE UNIQUE
Elle pourra être utilisée :
- pour réaliser un accès direct sur une réalisation d'une entité racine. Le programmeur devra donner le nom de l'entité et la valeur de la propriété identifiante de cette entité qui en définit biunivoquement une réalisation. Comme nous l'avons préciser, cet accès ne peut s'effectuer que pour les entités racines car ce sont les

seules entités à posséder une propriété identifiante.

- pour réaliser un accès par itinéraire.
  Cet itinéraire est, pour chaque entité, un chemin particulier défini à la description de la base de données et que nous avons appelé chemin hiérarchique ou chemin principal de recherche.
  Le programmeur devra donner le nom des entités composant le chemin hiérarchique, ainsi que la valeur de chacune des propriétés identifiantes et localement identifiantes de ces entités. Le système pourra, avec ces informations, parcourir le chemin en partant de la réalisation de l'entité racine référenciée par la valeur de sa propriété identifiante, chacune des autres valeurs déterminant la réalisation à sélectionner des entités dépendantes jusqu'à la réalisation recherchée.
  Cette technique ne peut, évidemment, être appliquée que pour des entités possédant une propriété localement identifiante et dont le chemin hiérarchique les reliant à l'entité racine est composé uniquement d'entités possédant chacune une propriété localement identifiante.
  L'ensemble composé de la valeur de la propriété identifiante de l'entité racine et de chacune des propriétés localement identifiantes des entités dépendantes du chemin hiérarchique jusqu'à et y compris l'entité considérée est ce que nous avons appelé clé concaténée de cette entité. Cette clé est identifiante pour l'entité car à chacune de ses valeurs correspond une et une seule réalisation de l'entité.
  La primitive RETRIEVE UNIQUE sera utilisée pour retrouver une réalisation donnée d'une entité, ou bien pour

établir un positionnement dans la base de données, celui-ci servant par la suite comme point de départ d'une recherche séquentielle.

b. Règles d'utilisation

- Recherche d'une réalisation d'une entité racine.
La primitive RETRIEVE UNIQUE possédera toujours un qualificateur contenant :
  . le nom de l'entité recherchée en 8 caractères
  . la valeur de la propriété identifiante référenciant la réalisation recherchée.

et une zone d'entrée/sortie qui contiendra la réalisation recherchée après l'exécution de la primitive, et donc qui devra être d'une longueur au moins égale à la longueur définie pour l'entité.

Exemple : Considérons l'entité racine "client"
Rechercher le client dont le numéro de client est 518
Nous aurons :

```
01 CLIENT.
   02 NOMSEG PIC X(8) VALUE "CLIENT".
   02 VALCLE PIC X(4).
MOVE 518 TO VACLE OF CLIENT.
CALL UNIQUE      USING IOAREA,CLIENT.
IF STATUS-ERROR NOT=SPACES GO TO ERREURRECHERCHE.
```

- Recherche d'une réalisation d'une entité dépendante.
La primitive RETRIEVE UNIQUE possèdera autant de qualificateurs qu'il y a d'entités sur le chemin hiérarchique c'est-à-dire autant de qualificateurs qu'il y a de niveaux dans le graphe de structure réduit aux relations à membre obligatoire pour l'entité considérée.
Si l'entité dont une réalisation est recherchée est de

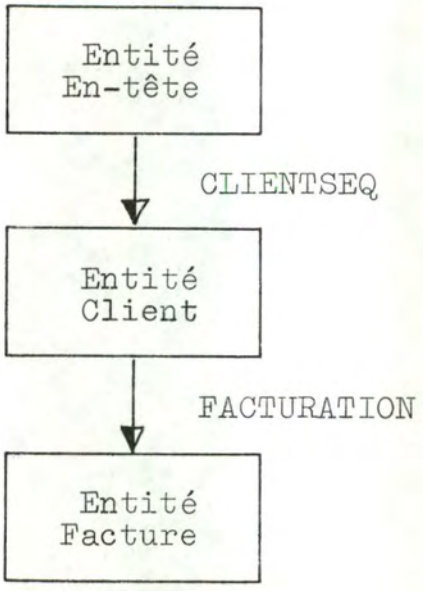
niveau 5, le premier niveau étant le niveau de l'entité racine, il y aura 5 qualificateurs.
Chaque qualificateur contiendra :

- le nom de l'entité qu'il représente
- la valeur de la propriété identifiante ou localement identifiante de cette entité.

En plus, ils seront placés dans l'ordre :

- le premier qualificateur correspond à l'entité racine
- le deuxième qualificateur correspond à l'entité de niveau deux
- .....
- le dernier qualificateur correspond à l'entité dont une réalisation est recherchée.

Exemple : Considérons les deux entités "CLIENT" et "FACTURE" dont le graphe est :



Rechercher la facture dont le numéro de facture est 55672, dépendante du client numéro 518
Nous devrons avoir définit :

- le qualificateur de l'entité CLIENT comme dans l'exemple précédent
- le qualificateur de l'entité FACTURE par

```
01 FACTURE.
02 NOMSEG PIC X(8) VALUE "FACTURE".
02 VALCLE PIC 9(6).
```

Le programme sera :

```
MOVE 518 TO VALCLE OF CLIENT.
MOVE 55672 TO VALCLE OF FACTURE.
CALL UNIQUE USING IOAREA , CLIENT , FACTURE.
IF STATUS-ERROR NOT = SPACES GO TO ERREURRECHERCHE.
```

5.3.3.2. Accès à une réalisation d'entité en fonction des accès précédents

Ce type d'accès sera basé sur une recherche pas à pas le long d'une chaîne de réalisations.
Plusieurs possibilités sont offertes :

a. Accéder à la réalisation but de l'occurrence suivant l'occurrence courante d'une relation.
La primitive réalisant cette fonction sera la primitive RETRIEVE NEXT
Elle pourra être utilisée :

- pour retrouver la réalisation suivante dans une relation à membre obligatoire
- pour retrouver la réalisation suivante dans une relation faible.
  Rappelons que la relation inverse d'une relation faible doit posséder un nom distinct.

Elle possédera toujours un qualificateur qui donnera le nom de la relation dont l'occurrence suivant l'occurrence courante donne la réalisation recherchée.
Dans le cadre de l'exemple précédent, nous écrirons pour retrouver la facture suivante d'un client
CALL NEXT USING IOAREA , LELFACT.
avec 01 LCLFACT PIC X(8) VALUE "FACTURAT".

Si l'ordre a pu être exécuté convenablement le STATUS-ERROR sera égal à des blancs, si l'ordre n'a pu être exécuté parceque l'occurrence courante était la dernière occurrence de la relation source, alors le STATUS-ERROR sera égal à "0001".

La primitive RETRIEVE NEXT sera principalement utilisée pour accéder à une série de réalisations but reliées à une réalisatio source donnée par un positionnement préalablement établi dans la base de données.

Exemple : soit le problème :
"Donner toutes les factures du client numéro 518"
Nous aurons en programmation :

```
MOVE 518 TO VALCLE OF CLIENT.
CALL UNIQUE USING IOAREA , CLIENT.
IF STATUS-ERROR NOT = SPACES GO TO TRAITERROR1.
R. CALL NEXT USING IOAREA , LELFACT.
IF STATUS-ERROR = SPACE GO TO NEXT SENTENCE ELSE
IF STATUS-ERROR = "0001" GO TO FIN-TRAIT ELSE
GO TO TRAITERROR2.

Traitement des factures
GO TO R.
Fin-trait. ......
```

- "Donner toutes les factures du client numéro 518 à partir de la facture dont le numéro est 55672"

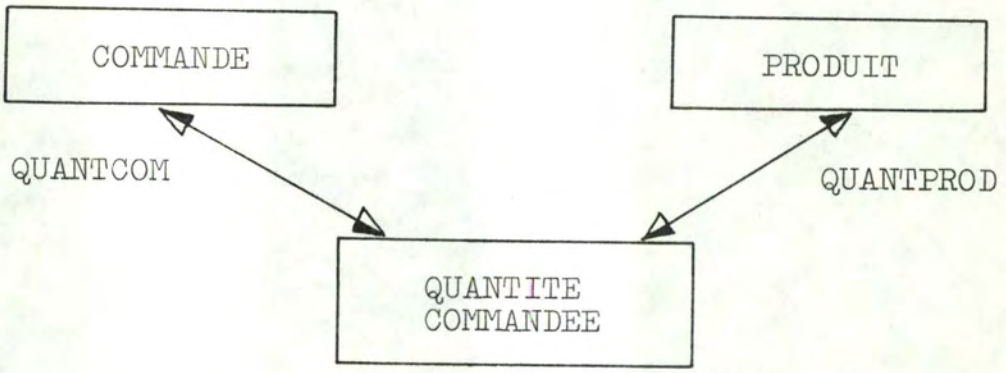
```
MOVE 518 TO VALCLE OF CLIENT.
MOVE 55672 TO VALCLE OF FACTURE.
CALL UNIQUE USING IOAREA , CLIENT , FACTURE.
IF STATUS-ERROR NOT = SPACES GO TO TRAITERROR1.
TRAIT-FACT.
.........
.........
CALL NEXT USING IOAREA , LELFACT.
IF STATUS-ERROR = SPACES GO TO TRAIT-FACT.
IF STATUS-ERROR = "0001" GO TO FIN-TRAIT.
GO TO TRAITERROR2.
```

b. Accéder à la réalisation source de l'occurrence courante d'une relation.

- La primitive réalisant cette fonction sera la primitive RETRIEVE SOURCE
  Elle possédera toujours un qualificateur qui donnera le nom de la relation dont l'occurrence courante contient la réalisation source recherchée.
  Elle ne pourra être employée que pour une relation à membre obligatoire et sera surtout utilisée dans le cas des entités d'intersection pour retrouver l'une des réalisations sources des occurrences des deux relations à membre obligatoire dont l'entité d'intersection est l'entité but.
  Nous écrirons :

```
avec 01 Qualificateur 1 PIC X(8) VALUE
"nom de relation".
```

Exemple : Soient les entités "Commande" , "Quantité commandée" et "Produit", et les relations entre ces entités comme présentées sur le graphe :



Le problème posé pourrait être :
"Donner pour la commande 5035, tous les produits commandés avec leur quantité en commande".
Nous aurions en programmation :

- La primitive RETRIEVE SOURCE positionne la réalisation de l'entité source retrouvée comme étant la réalisation courante de cette entité.
  Mais cela peut être ennuyeux pour l'utilisateur, en effet dans le cas où deux relations associent par l'intermédiaire d'une entité d'intersection deux réalisations d'une même entité, l'utilisateur peut vouloir garder le positionnement sur la réalisation de départ et connaître le contenu de la deuxième réalisation. Ce cas est celui présenté par le problème de la nomenclature d'un produit.
  La primitive qui remplira cette fonction sera la primitive RETRIEVE HEAD
  Son emploi sera le même que la primitive RETRIEVE SOURCE sauf que le positionnement des réalisations but de la relation ne change pas.
  Nous écrirons

c. Accéder à la première réalisation de l'entité but d'une relation.
   La primitive réalisant cette fonction sera la primitive RETRIEVE FIRST
   Elle possédera toujours un qualificateur qui donnera le nom de la relation dont la première occurrence contient la réalisation de l'entité but recherchée.
   Elle ne peut être utilisée que si un positionnement préalable a été effectué - soit sur la réalisation de l'entité source de la relation
   - soit sur une réalisation de l'entité but de la relation.

   Nous écrirons l'ordre d'appel comme suit :
   CALL FIRST USING IOAREA , Qualificateur.
   avec 01 Qualificateur PIC X(8) VALUE "nom de relation"

nom de relation ne pourra pas être le nom d'une relation "One to Many" à membre obligatoire reliant une entité en-tête à une entité racine.
Nous l'emploierons, par exemple, lorsqu'il faut lire plusieurs fois les réalisations buts associées à une réalisation source par les occurrences de la relation nom de relation.

Exemple : En reprenant les conditions d'un exemple précédent, soit le problème "Lire deux fois les factures du client numéro 518".
Nous aurons, le début de la programmation étant le même que dans l'exemple considéré

```
FIN-TRAIT. IF SW1F = 1 GO TO FINI.
           MOVE 1TO SW1F.
           CALL FIRST USING IOAREA , LELFACT.
           IF STATUS-ERROR = "001" GO TO FINI.
           IF STATUS-ERROR NOT = SPACES GO TO
           TRAITERRORS3.
           GO TO Traitement des factures.
```

5.3.4. Insertion d'une réalisation d'entité

La primitive qui réalisera l'insertion d'une nouvelle réalisation d'une entité dans la base de données, sera la primitive INSERT.
Elle réalise l'insertion de la nouvelle réalisation d'entité et aussi l'insertion des occurrences de relations à membre obligatoire dans lesquelles cette entité intervient comme entité but.
La nouvelle réalisation, ainsi que les occurrences de relations automatiquement créées, sont alors disponibles pour d'autres traitements.

Avant d'insérer la nouvelle réalisation, certaines réalisations, avec lesquelles elle compose comme but les occurrences de relations à membre obligatoire, devront exister dans la base de données et être connues du système de gestion de celle-ci.
Le nombre de ces réalisations préexistantes à la création, sera égal à un ou à deux suivant que la nouvelle réalisation n'est pas ou est une réalisation d'une entité d'intersection.
Le programmeur doit permettre au système de gestion de la base de données de retrouver ces réalisations pour vérifier leur présence et créer les occurrences des relations.

- Réalisation d'une entité d'intersection

  Le programmeur doit se charger de rechercher une des deux réalisations qui seront reliées à la nouvelle réalisation. Il réalisera cette fonction, soit par une primitive RETRIEVE UNIQUE soit par tout autre type de primitive RETRIEVE sauf la RETRIEVE HEAD et définira l'autre réalisation par des qualificateurs dans la primitive INSERT.

- Réalisation d'une entité non d'intersection

  Le programmeur a ici deux possibilités :

  . La première est de donner, dans la primitive INSERT, toutes les informations nécessaires à la recherche de la réalisation devant se trouver dans la base de données. Cette possibilité est soumise aux mêmes contraintes que la primitive RETRIEVE UNIQUE.
  Nous écrirons CALL INSERT USING IOAREA , Q_1 , Q_2 ,...,Q_n où chaque qualificateur Q_i contient les mêmes données que pour la primitive RETRIEVE UNIQUE sauf le dernier qualificateur Q_n qui ne contient que le nom de l'entité dont nous voulons créer une réalisation.

. La deuxième possibilité est de déterminer la réalisation par une recherche préalable ce qui établit un positionnement dont se sert le système.
Cette recherche peut être réalisée à l'aide de toute primitive RETRIEVE sauf la RETRIEVE HEAD
Nous écrirons :

CALL INSERT USING IOAREA , Q_1.

où Q_1 représente un qualificateur contenant le nom de l'entité dont nous voulons créer une réalisation.

- Avant de réaliser l'insertion, le système doit vérifier que l'insertion automatique d'occurrences de relation ne modifiera pas le type initiallement fixé pour ces relations.
Si le système refuse l'insertion, le programmeur recevra un code erreur dans la zone STATUS-ERROR.

Exemple : Créer une nouvelle réalisation de l'entité "facture" reliée au client dont le numéro est 518.

```
MOVE 518 TO VALCLE OF CLIENT.
CALL UNIQUE USING IOAREA , CLIENT.
IF STATUS-ERROR NOT = SPACES GO TO TRATERROR.
Garnir la zone IOAREA par la nouvelle réalisation
CALL INSERT USING IOAREA , FACTURE
IF STATUS-ERROR NOT = SPACES GO TO ERRORINSERT.
......
```

5.3.5. Insertion d'une occurrence de relation

5.3.5.1. Insertion d'une occurrence de relation à membre obligatoire

Il n'existe pas d'instruction d'insertion d'une occurrence de relation à membre obligatoire car ces insertions

sont automatiquement exécutée lors de l'insertion d'une réalisation de l'entité but de la relation.

5.3.5.2. Insertion d'une occurrence de relation faible

La primitive qui réalisera l'insertion d'une occurrence de relation faible dans la base de données, sera la primitive

ATTACH

Avant d'insérer la nouvelle occurrence de relation, il faudra spécifier, au système de gestion, qu'elles sont les deux réalisations d'entités intervenant dans cette occurrence. Le programmeur, comme pour l'insertion d'une réalisation d'entité d'intersection se chargera de la recherche d'une des deux réalisations et définira l'autre réalisation par des qualificateurs au niveau de la primititve ATTACH. Nous écrirons :

CALL ATTACH USING Q_1 , Q_2 , ... , Q_n.

où Q_2 , ... , Q_n sont les qualificateurs d'une des deux réalisations et où Q_1 représente un qualificateur contenant le nom de la relation dont nous voulons créer une occurrence. L'occurrence correspondante de la relation inverse sera automatiquement créée.

Remarquons qu'au niveau de l'implémentation, la création d'une occurrence de relation faible se réalisera par la création de la réalisation de l'entité d'intersection qui établit la liaison entre les deux réalisations de cette occurrence.
L'entité d'intersection ne sera qu'une entité "pointeur" sauf si la relation faible est une relation faible, auquel cas

l'entité d'intersection contiendra la propriété d'ordre allociée à la relation. Si en plus la relation inverse est aussi une relation d'ordre, l'entité d'intersection contiendra la propriété d'ordre associée à cette relation.

En résumé, l'entité d'intersection réalisant la relation faible est transparente au programmeur, et peut posséder de zéro à deux propriétés.

La technique d'implémentation des relations faibles, par l'intermédiaire d'une entité d'intersection (entité "pointeur") présente l'avantage suivant : seules des relations à membre obligatoire existent physiquement et ces relations sont gérées automatiquement avec contrôle de la cohérence de l'insertion d'une réalisation d'entité.

5.3.6. Modification d'une réalisation d'entité

La primitive qui réalisera la modification d'une réalisation d'entité sera la primitive MODIFY
Le programmeur aura, à sa charge, la recherche de la réalisation d'entité à modifier si celle-ci n'est pas la réalisation courante.

Rappelons que le système n'admet pas la modification de la valeur d'une propriété identifiante, localement identifiante ou d'ordre. Il vérifiera ce point, rejetera éventuellement la modification et en préviendra le programmeur en plaçant un code rejet dans la zone STATUS-ERROR.
Nous aurons la primitive :

CALL MODIFY USING IOAREA , Q_1.

La zone IOAREA contiendra la réalisation modifiée.
Q_1 est un qualificateur contenant uniquement le nom de l'entité dont nous voulons modifier une réalisation.
Le système rejettera aussi la modification si la réalisation courante n'est pas une réalisation de l'entité dont le nom apparaît dans le qualificateur de la primitive.

5.3.7. Suppression d'une réalisation d'entité.

La primitive DELETE sera la primitive qui réalisera la suppression d'une réalisation d'entité.

Rappelons que la suppression d'une réalisation d'entité entraîne la suppression de toutes les occurrences de relation dans lesquelles intervenait cette réalisation soit comme but, soit comme source, avec une éventuelle propagation des suppressions automatiques dans le cas d'occurrences de relation à membre obligatoire.

Lorsqu'une réalisation est supprimée, elle n'est plus accessible pour aucun programme utilisateur et une tentative pour se refixer à cette réalisation se solde par une erreur signalée sous forme d'un code placé dans la zone STATUS-ERROR.

Pour supprimer une réalisation, le programmeur doit avoir préalablement retrouvé cette réalisation, c'est-à-dire qu'elle doit être courante pour le système. Si elle ne l'était pas lors de la demande de suppression, le système n'effectuerait pas l'opération et en avertirait le programmeur par le le placement d'un code dans la zone STATUS-ERROR.

Nous écrirons :

CALL DELETE USING IOAREA , Q_1.

avec : Q_1 est un qualificateur contenant le nom de l'entité dont une réalisation va être supprimée.

5.3.8. Suppression d'une occurrence de relation faible

La primitive qui réalisera la suppression d'une occurrence de relation faible sera la primitive : DETACH
La détermination d'une occurrence de relation faible s'effectuera par la détermination des deux réalisations intervenant dans cette relation.
Le programmeur sera chargé de faire connaître ces réalisations au système de gestion de la base de données.
Il effectuera la recherche de la réalisation source de l'occurrence et placera la clé concaténée de la réalisation but dans la zone IOAREA de la primitive (cadrage à gauche).
Nous écrirons :

CALL DETACH USING IOAREA , Q_1.

où - la zone IOAREA contient la clé concaténée de la réalisation but de l'occurrence de relation à supprimer
- la zone Q_1 est un qualificateur contenant le nom de la relation.

Au lancement de cette primitive DETACH, il faut que le programmeur aie positionné la réalisation source de l'occurrence comme réalisation courante.
Seule l'occurrence de la relation sera supprimée, les réalisations d'entité la constituant ne seront pas touchés par cette primitive.
Au niveau de l'implémentation la suppression d'une occurrence

de relation se ramène à la suppression de la réalisation créée pour établir la relation.

5.3.9. Remarques sur l'implémentation du langage de manipulation

5.3.9.1. Les primitives du langage de manipulation s'appuie beaucoup sur la notion de réalisations courantes

- Le sytème IDS se base sur des tables de chaînage

  Ces tables sont utilisées par les sous-routines IDS.
  Pour chaque relation (type de chaîne), le système IDS génère une table de chaînage. Celle-ci a quatre points d'entrée qui sont MASTER , PRIOR , CURRENT , et NEXT ; chaque point d'entrée représente un code de référence correspondant à la réalisation de la chaîne représentée par ce point d'entrée.

- Le système IMS se base sur une position courante (réalisation d'entité) dans la base de données

  Deux possibilités sont offertes à l'utilisateur pour définir ce que le système doit regarder comme étant une position courante :

  . Le "Single positionning"
  . Le "Multiple positionning"

  Cette option est spécifiée au niveau du PCB.
  Lorsque l'option "Single positionning" est spécifiée pour un PCB, le système IMS gère une et une seule position courante pour ce PCB.
  Lorsque l'option "Multiple positionning" est spécifiée pour un PCB, le système IMS maintient une position courante par

entité (type de segment) de ce PCB.
Pour tous les PCB que nous devrions définir dans le module de conversion nous choisirions l'option "Multiple positionning".

5.3.9.2. La primitive RETRIEVE UNIQUE

a. - Dans le système IDS, la primitive RETRIEVE UNIQUE sera traduite par les instructions :
- RETRIEVE UNIQUE "nom d'entité" RECORD
- MOVE

- Si "nom d'entité" désigne une entité racine, elle aura été définie avec l'option "RETRIEVAL VIA CALC CHAIN" ou par "RETRIEVAL nom de zone FIELD".
Il suffira donc avant de programmer l'appel IDS de garnir la zone "RANDOMILE KEY" ou la zone principale de recherche.

- Si "nom d'entité" désigne une entité dépendante, elle aura été définie avec un type de chaîne principal de recherche, celle-ci aura été spécifiée "SELECT UNIQUE".
Il faudra, préalablement à l'appel IDS, garnir toutes les zones "MATCH-KEY" du chemin hiérarchique.

b. Dans le système IMS, la primitive RETRIEVE UNIQUE sera traduite par l'appel IMS
CALL "CBLTDLI" USING CF , nom de PCB , IOAREA , SSA1 , SSA2 ,...
où - le code fonction sera GHU
- chaque SSA représentera un niveau de la structure hiérarchique et sera qualifié par la valeur des différentes zones de séquence.

5.3.9.3. La primitive RETRIEVE NEXT

a.-Le nom de relation de la primitive référencie une relation à membre obligatoire, alors la primitive sera traduite par les instructions IDS :
RETRIEVE NEXT RECORD OF nom de chaîne CHAIN
et MOVE
Pour cela, il faut définir dans le module de conversion, une table des types de chaîne contenant :
- le nom de la chaîne
- l'adresse de la chaîne

-Par l'instruction IMS
CALL "CBLTDLI" USING CF,, nom de PCB , IOAREA , SSA1 , SSA2.
avec - la zone CF contient le code fonction GHN
- le nom de PCB est donné par une table en fonction du nom de relation
- la zone SSA1 est un SSA non qualifié du nom de l'entité-source de la relation avec le code commande V.
- la zone SSA2 est un SSA non qualifié du nom de l'entité but de la relation sans code commande.

Pour cela, il faut définir par relation, une table contenant :
- le nom de la relation
- le nom du PCB contenant les entités source et but de la relation
- le SSA de l'entité source de la relation
- le SSA de l'entité but de la relation.

b. Le nom de relation de la primitive référencie une relation faible, alors la primitive sera traduite :

- Par les instructions IDS :
  RETRIEVE NEXT RECORD nom de chaîne 1 CHAIN
  RETRIEVE MASTER RECORD nom de chaîne 2 CHAIN
  MOVE

  La table correspondante sera par relation faible :
  - le nom de la relation
  - le nom et l'adresse de la relation nom de chaîne 1.
  - le nom et l'adresse de la relation nom de chaîne 2.

- Par les instructions IMS :
  CALL "CBLTDLI" USING CF1 , nom de PCB1 , IOAREA , SSA1 ,
  SSA2 avec CF1 = GN.
  CALL "CBLTDLI" USING CF2 , nom de PCB2 , IOAREA , SSA3 avec
  CF2 = GHU.

  La table correspondante contiendra par relation faible :
  - le nom de la relation
  - le nom du PCB contenant l'entité source (PCB1)
  - le nom du PCB contenant l'entité but (PCB2)
  - le SSA1 de l'entité source avec un code commande V.
  - le SSA2 de l'entité d'intersection sans code commande
  - le SSA3 de l'entité but avec un code commande C.

5.3.9.4. La primitive RETRIEVE SOURCE

- Le système IDS :

  RETRIEVE MASTER RECORD OF nom de relation CHAIN
  et MOVE.

- le système IMS,

  CALL "CBLTDLI" USING CF , nom de PCB , SSA1
  avec CF = fonction de la primitive = GHU
  - le nom de PCB est donné en fonction de nom de relation
  - le SSA1 est un SSA qualifié par la clé concaténée de l'entité recherchée contenant l'entité recherchée. Cette clé sera toujours la bonne clé car à chaque fois que l'on retrouvera une occurrence d'une entité d'intersection, l'IMS nous donnera sa clé concaténée que nous mettrons dans la clé concaténée de la table des entités. En plus ce SSA1 sera défini avec le code commande C.

5.3.9.5. La primitive RETRIEVE HEAD

- Le système IDS

  RETRIEVE HEAD OF nom de relation CHAIN
  et MOVE

- Le sytème IMS , le problème est le même que lors du RETRIEVE SOURCE car nous avons définit deux PCB pour la même structure physique , l'un contenant une relation par exemple la relation physique, l'autre contenant alors la relation logique.

5.3.9.6. La primitive RETRIEVE FIRST

a.-Le nom de relation de la primitive référencie une relation à membre obligatoire, alors la primitive sera traduite par :

- Dans le système IDS :

RETRIEVE MASTER RECORD OF nom de chaîne CHAIN
RETRIEVE NEXT RECORD OF nom de chaîne CHAIN
MOVE

- Dans le système IMS :

CALL "CBLTDLI" USING CF , nom de PCB , IOAREA , SSA1 , SSA2.

avec
- la zone CF contient le code fonction GHN
- le nom de PCB est donné par une table en fonction du nom de relation
- la zone SSA1 est un SSA non qualifié du nom de l'entité source de la relation avec le code commande V.
- la zone SSA2 est un SSA non qualifié du nom de l'entité but de la relation avec le code commande F.

a. - Le nom de la relation de la primitive référencie une relation faible alors la primitive sera traduite par :

- Dans le système IDS :

RETRIEVE MASTER RECORD OF nom de chaîne 1 CHAIN.
RETRIEVE NEXT RECORD OF nom de chaîne 1 CHAIN.
RETRIEVE MASTER RECORD OF nom de chaîne 2 CHAIN.
MOVE

- Dans le système IMS :

CALL "CBLTDLI" USING CF1 , nom de PCB1 , IOAREA , SSA1 , SSA2.
CALL "CBLTDLI" USING CF2 , nom de PCB2 , IOAREA , SSA3.

avec : - nom de PCB1 est le nom du PCB contenant l'entité source
- nom de PCB2 est le nom du PCB contenant l'entité but.
- SSA1 est un SSA non qualifié du nom de l'entité source avec le code commande V.
- SSA2 est un SSA non qualifié du nom de l'entité d'intersection établissant la relation faible, avec le code commande F.
- SSA3 est un SSA qualifié du nom de l'entité but et de la valeur de sa clé concaténée avec le code commande C.
- la zone CF1 contient le code function GN
- la zone CF2 contient le code function GHU

5.3.9.7. La primitive INSERT

a. Dans le système IDS, la primitive INSERT sera réalisée par la primitive IDS STORE non d'entité RECORD
Les deux possibilités se traduisent par :
- soit le garnissage de toutes les MATCH KEY
- soit le positionnement des réalisations "maître" réalisée par les primitives IDS RETRIEVE.

b. Dans le système IMS, la primitive INSERT sera réalisée par la primitive IMS,
CALL "CBLTDLI" USING CF , nom de PCB1 , IOAREA , SSA1 , SSA2 , ... , SSA_n.

avec : - la zone CF contient le code fonction ISRT
- si la réalisation à insérer est une réalisation d'entité d'intersection, alors la zone IOAREA

contiendra la clé concaténée du PCB du parent logique en plus des données.

- les SSA sont qualifiés chacun par une partie de la clé concaténée du PCB du parent physique sauf le dernier SSA qui ne contient que le nom de l'entité dont une réalisation est à créer.

5.3.9.8. La primitive MODIFY

a. Dans le système IDS, la primitive MODIFY sera réalisée par la primitive IDS MODIFY CURRENT nom d'entité.

b. Dans le système IMS, la primitive MODIFY sera réalisée par la primitive IMS CALL "CBLTDLI" USING CF , nom de PCB , IOAREA , SSA
où la zone CF contient "REPL".

5.3.9.9. La primitive DELETE

La primitive DELETE sera réalisée par :

a. la primitive IDS :

DELETE CURRENT nom d'entité RECORD

b. la primitive IMS :

CALL "CBLTDLI" USING CF , nom de PCB , IOAREA , SSA1.
avec : - la zone CF contient le code fonction DLET.
- la zone SSA1 est un SSA non qualifié donnant le nom de l'entité dont une réalisation va être supprimée.

5.3.9.10. Les primitives ATTACH et DETACH de manipulation de relation faible.

La technique d'implémentation de ces primitives a été appliqué lors de la définition de ces primitives.

* * * * * *

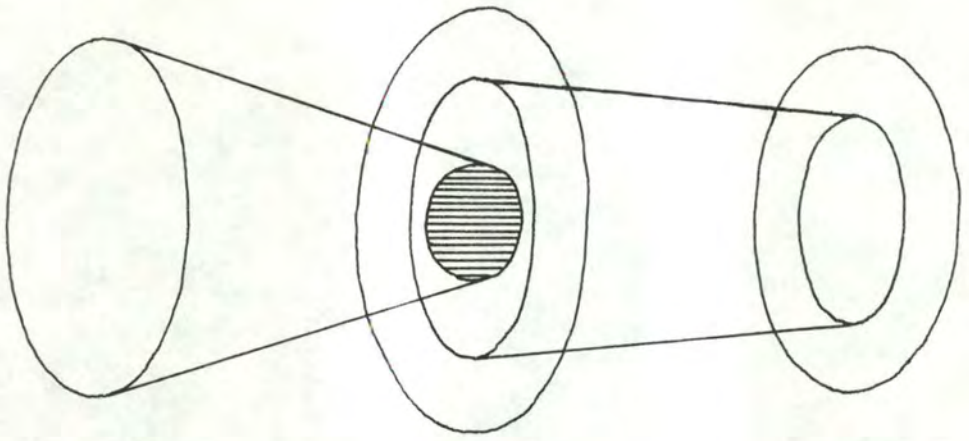
6. CONVERSION D'UNE BASE DE DONNEES ECRITE DANS UN SYSTEME A LA MEME BASE DANS L'AUTRE SYSTEME

6.1. Introduction

Comme nous l'avons annoncé dans l'introduction générale de l'étude, nous allons examiner, rapidement, l'aide que peut apporter la définition de structures sémantiques communes aux systèmes IDS et IMS à la résolution du problème posé par la conversion d'une base de données écrite dans un système à la même base dans l'autre système.

Ce problème, qui normalement demande une autre démarche que celle entreprise dans cette étude, pourrait se trouver simplifié par la présence du modèle d'accès et de son langage de manipulation associé entre les systèmes de gestion IDS et IMS, et même pourrait être résolu dans le sens système IDS vers système IMS si la partie implémentable du modèle d'accès général par le système IMS contient toute la projection du système IDS sur ce modèle d'accès, et réciproquement dans le sens système IMS vers système IDS si la partie implémentable du modèle d'accès général par le système IDS contient toute la projection du système IMS sur ce modèle d'accès.

Cette perspective peut être représentée graphiquement :



Le système source étant le système à convertir
Le système but étant le système dans lequel la conversion doit se réaliser.

6.2. Implémentation du modèle source par le modèle d'accès réduit.

Remarquons que nous avons effectué le travail inverse, c'est-à-dire que nous avons réalisé l'implémentation du modèle d'accès réduit par le modèle source et par le modèle but. De ce fait, si l'implémentation du modèle source par le modèle d'accès réduit était réalisée, la conversion serait terminée, car il suffirait alors d'implémenter le modèle d'accès réduit par le modèle but (ce qui a été réalisé dans cette étude) et d'utiliser les programmes d'application avec les primitives du langage de manipulation associé à ce modèle d'accès.

Mais nous devons bien constaté que l'implémentation du modèle source par le modèle d'accès n'est pas réalisable,

en effet :

- Le système source est le système IMS
  Si l'implémentation des structures de données IMS par les structures de données du modèle d'accès réduit, peut être réalisée, il n'en va pas de même au niveau du langage de manipulation où les fonctions de lecture hiérarchique et les fonctions de traitement par lot des segments (path of segments) sont impossibles à réaliser.

- Le système source est le système IDS
  Si le langage de manipulation associé au modèle d'accès est très proche du langage de manipulation du système IDS, il ne peut pas en remplir toutes les fonctions, par exemple la modification de propriétés d'ordre qui est réalisable dans le système IDS, est impossible dans le modèle d'accès réduit.
  L'implémentation de la structure de données du système IDS par le modèle d'accès réduit est impossible, seule une partie est implémentable, en effet : considérons par exemple toutes les relations à membre obligatoire ayant pour entité but, une entité d'intersection de plus de deux de ces relations ; nous devrions les implémenter par des relations faibles ce qui au niveau de la dynamique de la base de données est différent.

Nous pouvons en conclure qu'à l'aide du modèle d'accès réduit, une conversion automatique de la structure de données et des programmes d'application travaillant sur cette structure est impossible.

Mais le fait que le modèle d'accès soit implémentable par les deux systèmes IDS et IMS, rend ce modèle plus

proche de chacun des systèmes que ces systèmes l'un par rapport à l'autre, ce qui veut dire que la conversion du système source vers le modèle d'accès réduit est plus facile que la conversion du système source vers le système but.

Nous allons le montrer en quelques exemples :

- Supposons que le système IDS soit le système source, le langage de manipulation du système IDS est beaucoup plus proche du langage de manipulation du modèle d'accès réduit que de celui du système IMS.
- Supposons que le système IMS soit le système source, la structure de données du système IMS est comparable à la structure du modèle d'accès, en effet de par la structure hiérarchique logique du système IMS, le programmeur peut ne pas être sensible au segment "enfant logique" et passé directement par une primitive GET NEXT du segment "parent physique" au segment "parent logique" et inversément, ce qui se représente facilement par les relations faibles du modèle d'accès.
  Au niveau du langage de manipulation, les primitives du système IMS GET FIRST, GET NEXT et GET NEXT WITHIN PARENT le long d'une relation logique IMS sont réalisées par les primitives du modèle d'accès RETRIEVE FIRST, RETRIEVE NEXT le long d'une relation faible, primitives qui n'existent pas dans le langage de manipulation du système IDS.

En résumé, nous noterons que si la définition d'un modèle d'accès et d'un langage de manipulation associé implémentable par les systèmes IDS et IMS, ne permet pas d'envisager une conversion automatique entre ces deux systèmes, il

facilite la conversion manuelle, et nous aurons :

Système source $\xrightarrow{1}$ Modèle d'accès Langage de manipulation $\xrightarrow{2}$ Système but

où la conversion 1 est plus facile à réaliser que la conversion directe du système source vers le système but, et où la conversion 2 existe déjà puisque le système but implémente le modèle d'accès.

6.3. Evolution d'une conversion : cas réel

6.3.1. Si le passage par un modèle d'accès ne permet pas une conversion automatique, ne facilitant qu'une conversion manuelle, il a aussi facilité la présentation d'une méthode d'évaluation du coût de cette conversion manuelle.
L'estimation du coût d'une conversion IDS-IMS a été un problème rencontré par la société UCB qui désirait, dans le cadre d'un éventuel changement du matériel informatique, effectuer une estimation du temps nécessaire pour réaliser la conversion de ses programmes d'application du système IDS vers le système IMS.
Cette estimation "temps" a porté sur la conversion de l'application SIDAC qui avec ses 128 programmes travaillant avec le système IDS représentait un bon échantillon de tous les programmes existants dans cette société.
La base de données SIDAC écrite dans le système IDS étant découpée en bloc logique par l'option IDS "PAGE RANGE" nous avons classé les programmes d'application à convertir en deux groupes.
Le premier groupe comprenait les programmes utilisant des

blocs logiques dont la structure de données n'était pas implémentable par le modèle d'accès.
Le deuxième groupe comprenait tous les autres programmes, c'est-à-dire ceux dont la structure de données de leurs blocs logiques était implémentable.

6.3.2. Nous avons considéré quatre classes de difficultés de conversion croissantes :

- conversion immédiate : classe I
- conversion avec vérification des ordres IDS : classe II
- conversion avec vérification des ordres IDS et de la logique de programmation : classe III
- réécriture des programmes : classe IV.

Les programmes du premier groupe ont été placés dans la classe IV.
Les programmes du deuxième groupe ont été répartis dans les trois premières classes. L'appartenance d'un programme à une de ces classes s'est réalisée sur base de la nature des ordres IDS.
Nous avions :

- Classe I : conversion immédiate

  Fe ront partie de cette classe les programmes qui n'emploient que des primitives IDS ayant leur équivalent dans le langage de manipulation du modèle d'accès réduit.
  Ces ordres sont :
  - RETRIEVE calculée
  - Retrieve next
  - Retrieve master
  - Modify mais uniquement des propriétés non significatives pour le système
  - Store

- Classe II : conversion avec vérification des ordres IDS

Feront partie de cette classe les programmes qui emploient en plus des primitives acceptées en classe I les primitives IDS - Retrieve direct
- Retrieve current

Ces primitives peuvent être implémentées par la primitive Retrieve Unique, pour cela il faut que les entités, dont les réalisations sont recherchées par ces primitives, possèdent une clé concaténée.

- Classe III : conversion avec vérification des ordre IDS et de la logique de programmation

Feront partie de cette classe les programmes qui emploient en plus des primitives acceptées dans les autres classes, les primitives IDS :

. Modify d'une propriété d'ordre, identifiante ou localement identifiante
. Delete
. Emploi des tables de chaînage

La primitive "DELETE" se trouve en classe III parceque vu la dynamique des relations à membre obligatoire, cette primitive peut avoir des implications au niveau d'autres blocs logiques de la base de données, implications qu'il faut prendre le temps d'étudier.

6.3.3. Les résultats de cette classification ont été pour l'application SIDAC :

- Classe I : 39 programmes à raison de 6 heures par programme
- Classe II: 33 programmes à raison de 12 heures par programme

- Classe III : 44 programmes à raison de 30 heures par programme
- Classe IV : 12 programmes à raison de 60 heures par programme

Ce qui nous a conduit à l'estimation des temps de conversion suivante :
Total temps programmation SIDAC : 2682 heures.

* * * * *

7. CONCLUSIONS ET PROLONGEMENTS POSSIBLES DE L'ETUDE

7.1. Conclusions

L'idée, qui a guidé le choix des techniques d'implémentation du modèle d'accès général par les systèmes IDS et IMS, était de présenter un système de gestion de base de données suffisamment général pour être intéressant mais assez proche des systèmes IDS et IMS, qui devaient en réaliser l'implémentation, pour rester performant et pour pouvoir employer les modules de gestion de ces systèmes car l'implémentation devait rester accessible à un centre informatique d'une société privée voulant soit :

- Garder une certaine indépendance au niveau de la gestion d'une base de données par rapport aux systèmes particuliers d'implémentation que sont les systèmes IDS et IMS ; un changement de système d'implémentation se résumant alors à la réécriture du module de conversion réalisant l'interphase entre les programmes d'application et ce système.
- Pouvoir utiliser les mêmes programmes d'application utilisant des bases de données sur plusieurs ordinateurs, les uns travaillant sous le système IDS, les autres sous le système IMS, ce cas pouvant se présenter pour les administrations, les sociétés avec filiales ...

Malgré la présence de caractéristiques spécifiques au système IDS et surtout au système IMS, il a été possible de présenter un large sous-ensemble du modèle d'accès général implémentable par ces deux systèmes à la fois, tout en utilisant les modules de gestion de ces systèmes pour réaliser

la dynamique de la base de données et lui garder sa cohérence.

Il est certain que ce sous-ensemble aurait pu être plus général, mais quel aurait été le coût du module de conversion, celui-ci devant alors réaliser de plus en plus de fonctions car plus le système étudié s'éloigne des systèmes d'implémentation moins les modules de gestion de ceux-ci sont utilisables. En plus, nous ne pouvions réaliser l'étude des structures sémantiques communes aux systèmes IDS et IMS qu'en utilisant leurs modules de gestion et donc leurs contraintes propres.

Nous ne reprendrons pas ici toutes les conclusions tirées tout au long de l'étude, mais, en synthèse, nous noterons :

- Qu'au point de vue de la structure des données, le système d'implémentation IMS a imposé la majorité des contraintes, cela étant dû au fait que si le système IMS possède bien comme le système IDS, une structure en réseau sans cycle, il ne présente à l'utilisateur que des sous-structures hiérarchiques déduites de la structure en réseau par des règles restreignant fortement les possibilités d'utilisation des entités but de plus d'une relation à membre obligatoire.

- Au point de vue du langage de manipulation, le système IDS possède un langage adapté à une structure en réseau ; le système IMS possède un langage plus évolué mais uniquement adapté à une structure hiérarchique, nous avons choisi un langage semblable à celui du système IDS en y incorporant des primitives de manipulation des relations faibles.

En résumé, le système de gestion de base de données représenté par le modèle d'accès réduit et le langage de manipulation y associé, est :

- Pour les structures de données plus puissant que le système IMS et moins puissant dans les possibilités d'utilisation des relations à membre obligatoire que le système IDS mais en présentant l'avantage de posséder des relations faibles .

- Pour le langage de manipulation des données, semblable au langage du système IDS avec en plus des primitives de manipulation des relations faibles, et moins puissant que le langage du système IMS en précisant que celui-ci ne travaille que sur des structures hiérarchiques.

7.2. Prolongements possibles de l'étude

7.2.1. En ne considérant que le problème de l'indépendance des programmes d'application vis-à-vis de tout système de gestion de base des données, il est possible de généraliser l'étude en considérant tous les systèmes de gestion et non uniquement les systèmes IDS et IMS. Il est évident que généraliser l'étude des structures sémantiques communes à plus de deux systèmes conduirait à un modèle d'accès si réduit qu'il ne serait plus utilisable, mais en ne reprenant de ces systèmes de gestion que des techniques d'accès et en effectuant la gestion de la base de données au niveau du module d'interphase, il est possible de définir un modèle d'accès implémentable par tout système. En plus pour que ce type d'étude ne soit pas que théorique, il faudrait prendre en considération les problèmes de performance et de coût de réalisation.

7.2.2. En ne considérant que les structures de données des systèmes de gestion de base de données, il doit être possible de définir une structure de données généralisée pouvant contenir toutes les autres, comme nous l'avons réalisé avec le modèle d'accès général mais uniquement pour les structures IMS et IDS. Cette structure de données généralisée pourrait être associée à un langage de description des structures et servirait d'élément central à un système général de transformation des structures pour lequel en entrée, nous trouverions la définition dans le langage de description de la structure à transformer, la définition dans ce même langage de la structure à obtenir, et les données structurées par le système à transformer ; en sortie, nous obtiendrons les données structurées par le système à obtenir.

* * * * *

8. BIBLIOGRAPHIE

8.1. BACHMAN — Data Structure Diagrams
Data Base (journal of ACM SIGBDP) 1, n°2. 1969.

8.2. BACHMAN — The evolution of storage structures
ACM Vol.15, n°7, Juillet 1972.

8.3. BACHMAN — The programmer as Navigator
ACM Vol.16, n°11, Novembre 1973.

8.4. C.J. DATE — An introduction to Data Base Systems
ADDISON-WESLEY PUBLISHING COMPANY

8.5. DEHENEFFE, HAINAUT, TARDIEU
Modèle individuel de Banque de données
Faculté Universitaires Notre-Dame de la Paix, Namur.
Document interne.

8.6. EARLEY — Toward and Understanding of Data Structures
ACM Vol.14, n°10, Octobre 1971.

8.7. SVEN ERIKSEN — The Data Base Concept
Honeywell Bull, Copenhagen, Denmark.

8.8. J.L. HAINAUT, B.LECHARLIER
An Extensible Semantic Model of Data Base and its Data Language.
Proc. IFIP Congress 1974.

8.9. GUIDE/SHARE Data Base Task Force
Data Base Management System
Requirements.
SHARE 25 Broadway, New-York.

8.10. Système de conception et d'exploitation d'une Base de données
Projet de recherche C.I.P.S.
n°1.2/15
Institut d'Informatique - Facultés Universitaires Notre-Dame de la Paix
Namur.

8.11. Integrated Data Store, Order n° BR69
Honeywell Information Systems, 1971.

8.12. Information Management System / 360 version 2
General Information Manual
IBM Form n° GH20-0765

8.13. Information Management System/360 Version 2
Application Programming Reference Manual
IBM Form n°SH20-0912

8.14. Information Management System/360 Version 2
System Application Design Guide
IBM Form n°SH20-0910

* * * * * *