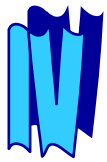

BASES DE DONNÉES

RÉSEAUX ET HIÉRARCHIQUES

1. INTRODUCTION

Les systèmes étudiés dans ce chapitre sont aujourd'hui appelés **systèmes légataires (*legacy systems*)**, car il nous sont légués par le passé. Ils permettent de modéliser des articles stockés dans des fichiers ainsi que les liens entre ces articles. Ces modèles dérivent avant tout d'une approche système au problème des bases de données qui tend à voir une base de données comme un ensemble de fichiers reliés par des pointeurs. Ils privilégient l'optimisation des entrées-sorties. C'est pourquoi nous les appelons aussi modèles d'accès. A ces modèles sont associés des langages de manipulation de données basés sur le parcours de fichiers et de liens entre fichiers, article par article, appelés langages navigationnels. Ces langages sont très caractéristiques des modèles d'accès.

Nous présentons successivement les deux modèles les plus populaires, à savoir le modèle réseau et le modèle hiérarchique, avec le langage de manipulation spécifique associé à chacun d'eux. Le modèle réseau proposé initialement par le groupe DBTG du comité CODASYL fut et reste utilisé avec diverses variantes possibles par de nombreux systèmes tels que IDS.II (Bull), IDMS (Computer-Associate), EDMS (Xerox), DMS/1100 (Univac), DBMS (Digital), PHOLAS (Philips) et TOTAL (Cincom). Le modèle hiérarchique étendu est employé par les systèmes anciens que sont IMS (IBM) et Systems 2000 (MRI-Intel), systèmes encore très répandus dans l'industrie. Nous concluons ce chapitre par une présentation des avantages et inconvénients des modèles d'accès.

2. LE MODÉLE RÉSEAU

Dans cette section, nous introduisons les principaux concepts du modèle réseau pour définir les bases de données et le langage associé du Codasyl.

2.1 Introduction et notations

Le modèle réseau a été proposé par le groupe DBTG du comité CODASYL [Codasyl71]. Des rapports plus récents [Codasyl78, Codasyl81] ont apporté des améliorations notables, en particulier une séparation plus nette entre les concepts de niveau interne et ceux de niveau conceptuel. Malheureusement, ces extensions ne sont que rarement intégrées dans les produits, pour la plupart construits au début des années 70. Le modèle réseau type CODASYL 1971 reste encore aujourd'hui utilisé par plusieurs systèmes, le plus connu en France étant IDS.II de BULL. Notre présentation est basée sur cette implémentation. Vous trouverez une présentation plus générale du modèle dans [Taylor76].

La syntaxe de présentation des clauses utilisées est dérivée de CODASYL, modifiée et étendue comme suit :

- les parenthèses carrées ([]) indiquent des éléments optionnels ;
- les pointillés suivent un élément qui peut être répété plusieurs fois ;
- les crochets groupent comme un seul élément une séquence d'éléments ;
- la barre verticale entre crochets du type {a | b | c | … } signifie que l'une des options a ou b ou c ou … doit être choisie ;
- un exposant plus ($^{+}$) indique que l'élément qui précède peut être répété n fois ($n \geq 1$), chaque occurrence étant séparée de la précédente par une virgule ; l'élément doit donc au moins apparaître une fois : il est obligatoire ;
- un exposant multiplié (*) indique que l'élément qui précède peut être répété n fois ($n \geq 0$), chaque occurrence étant séparée de la précédente par une virgule ; l'élément peut apparaître 0 fois : il est facultatif.

De plus, les mots clés obligatoires sont soulignés. Les paramètres des clauses sont précisés entre crochets triangulaires (<>).

2.2 La définition des objets

Les objets modélisés dans la base de données sont décrits à l'aide de trois concepts : **l'atome, le groupe** et **l'article**. Ces concepts permettent de décrire les données constitutives des objets stockés dans des fichiers.

Notion IV.1 : Atome (*Data Item*)

Plus petite unité de données possédant un nom.

La notion d'atome correspond classiquement au champ d'un article de fichier. Un atome possède un type qui définit ses valeurs possibles et les opérations que l'on peut accomplir sur ces valeurs. Un atome est instancié par une valeur atomique dans la base de données. Par exemple, CRU, DEGRE, AGE et NOM peuvent être des atomes d'une base de données de vins et de buveurs. Le type du CRU sera « chaîne de 8 caractères », alors que celui du degré sera « réel ».

Plusieurs atomes peuvent être groupés consécutivement pour constituer un **groupe** de données.

Notion IV.2 : Groupe (Data Aggregate)

Collection d'atomes rangés consécutivement dans la base et portant un nom.

Il existe deux types de groupes : les groupes simples et les groupes répétitifs. Un groupe simple est une suite d'atomes, alors qu'un groupe répétitif est une collection de données qui apparaît plusieurs fois consécutivement. Un groupe répétitif peut être composé d'atomes ou même de groupes répétitifs. Un groupe répétitif composé d'un seul atome est appelé **vecteur**. Par exemple, MILLESIME, composé d'ANNEE et QUALITE, peut être défini comme un groupe simple apparaissant une seule fois dans un vin. Au contraire, ENFANT composé de PRENOM, SEXE et AGE pourra apparaître plusieurs fois dans le descriptif d'une personne : il s'agit d'un groupe répétitif. Une personne pouvant avoir plusieurs prénoms, la donnée PRENOM pourra être un vecteur apparaissant au plus trois fois. Notons que le modèle réseau impose de limiter le nombre de répétitions possibles d'un groupe. Atomes et groupes permettent de constituer les **articles**.

Notion IV.3: Article (Record)

Collection d'atomes et de groupes rangés côte à côte dans la base de données, constituant l'unité d'échange entre la base de données et les applications.

Un article peut à la limite ne contenir aucune donnée. Les occurrences d'articles sont rangées dans des fichiers (AREA). Par exemple, un fichier de vins contiendra des articles composés d'atomes NUMERO, CRU et du groupe répétitif MILLESIME, répété au plus cinq fois.

Les articles sont décrits au niveau du type dans le schéma au moyen d'une clause :

RECORD NAME IS <nom-d'article>.

Par exemple, le type d'article VINS sera introduit par la clause **RECORD NAME IS** VINS. Suivront ensuite les descriptions détaillées des données de l'article.

Chaque atome ou groupe est défini par un nom précédé d'un niveau éventuel. Les données d'un article sont donc définies au moyen d'un arrangement hiérarchique

de groupes dans l'article. Le groupe de niveau 1 est l'article proprement dit. Les données de niveau 2 sont constituées par tous les atomes non agrégés dans un groupe. Les groupes de niveau 3 sont composés de tous les groupes n'étant pas à l'intérieur d'un autre groupe. Viennent ensuite les groupes internes à un groupe (niveau 4), etc. Pour chaque atome, une spécification de type précise le domaine de la donnée (décimal, binaire, caractères). Ainsi, la clause de définition d'un atome est :

[<niveau>] <nom-d'atome> **TYPE IS** <spécification-de-type>

alors que celle de définition d'un groupe est simplement :

[<niveau>] <nom-de groupe>.

Les types de données possibles sont les suivants (version minimale sous IDS.II):

- décimal signé condensé ou non (n1 et n2 précisent respectivement le nombre de chiffre total et celui après la virgule) :

  SIGNED [{ PACKED | UNPACKED }] DECIMAL <n1>, [<n2>]

- entier binaire long (signe plus 31 bits) ou court (signe plus 15 bits) :

  SIGNED BINARY { 15 | 31 }

- chaîne de caractères de longueur fixe (n1 caractères) :

  CHARACTER <n1>.

Un atome ou un groupe peuvent être répétés plusieurs fois (cas des vecteurs et des groupes répétitifs). Cela est précisé par la clause **OCCURS** du langage de description de données (n1 est un entier quelconque) :

OCCURS <n1> **TIMES**.

Afin d'illustrer ces différentes clauses, nous décrivons (figure IV.1) un type d'article VINS regroupant les cinq derniers millésimes (caractérisés par une année et un degré) d'un vin défini par un numéro (NV) et un nom de cru (CRU). Nous décrivons également un type d'article PRODUCTEURS composé d'un numéro (NP), un nom (NOM), un prénom (PRENOM) et un groupe simple ADRESSE, composé de RUE, CODE et VILLE.

```
RECORD NAME IS VINS;
02 NV TYPE IS SIGNED PACKED DECIMAL 5;
02 CRU TYPE IS CHARACTER 10;
02 MILLESIME OCCURS 5 TIMES;
03 ANNEE TYPE IS SIGNED UNPACKED DECIMAL 4;
03 DEGRE TYPE IS BINARY 15.


RECORD NAME IS PRODUCTEURS;
02 NP TYPE IS SIGNED PACKED DECIMAL 5;
02 NOM TYPE IS CHARACTER 10;
02 PRENOM TYPE IS CHARACTER 10;
02 ADRESSE
      03 RUE TYPE IS CHARACTER 30;
      03 CODE TYPE IS SIGNED PACKED DECIMAL 5;
      03 VILLE TYPE IS CHARACTER 10;
```

Figure IV.1 — Description des données de deux types d'articles

2.3 La définition des associations

Les possibilités offertes pour modéliser les associations entre objets constituent un des éléments importants d'un modèle de données. Historiquement, le modèle réseau est issu d'une conceptualisation de fichiers reliés par des pointeurs. De ce fait, il offre des possibilités limitées pour représenter les liens entre fichiers. Avec les recommandations du CODASYL, il est seulement possible de définir des associations entre un article appelé propriétaire et n articles membres. Une instance d'association est le plus souvent une liste circulaire d'articles partant d'un article propriétaire et parcourant n articles membres pour revenir au propriétaire. Ces associations, qui sont donc purement hiérarchiques mais qui, utilisées à plusieurs niveaux, peuvent permettre de former aussi bien des arbres, des cycles que des réseaux, sont appelées ici **liens** (en anglais *set*).

> **Notion IV.4 : Lien (Set)**
>
> Type d'association orientée entre articles de type T1 vers articles de type T2 dans laquelle une occurrence relie un article propriétaire de type T1 à n articles membres de type T2.

Un type de lien permet donc d'associer un type d'article propriétaire à un type d'article membre ; une occurrence de lien permet d'associer une occurrence d'article propriétaire à n occurrences d'articles membres. Généralement, les articles membres seront d'un type unique, mais la notion de lien peut théoriquement être étendue afin de supporter des membres de différents types.

Un lien se représente au niveau des types à l'aide d'un **diagramme de Bachman**. Il s'agit d'un graphe composé de deux sommets et d'un arc. Les sommets,

représentés par des rectangles, correspondent aux types d'articles ; l'arc représente le lien de 1 vers n [Bachman69]. L'arc est orienté du type propriétaire vers le type membre. Il est valué par le nom du type de lien qu'il représente et chaque sommet par le nom du type d'article associé.

Par exemple, les types d'articles VINS et PRODUCTEURS décrits ci-dessus seront naturellement reliés par le type de lien RECOLTE, allant de PRODUCTEURS vers VINS. Une occurrence reliera un producteur à tous les vins qu'il produit. La figure IV.2 schématise le diagramme de Bachman correspondant à ce lien.

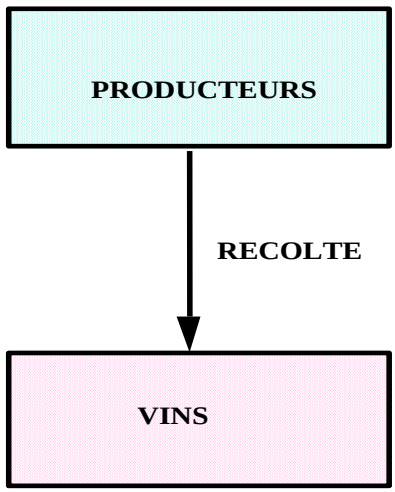

Figure IV.2 — Exemple de diagramme de Bachman

Deux limitations sont importantes : (i) un type d'article ne peut être à la fois propriétaire et membre d'un même lien ; (ii) une occurrence d'article ne peut appartenir à plusieurs occurrences du même lien. Par exemple, deux producteurs ne pourront partager la récolte d'un même vin, comme le montre la figure IV.3. Par contre, un type d'article peut être maître de plusieurs liens. Il peut aussi être membre de plusieurs liens. Deux types d'articles peuvent aussi être liés par des types de liens différents.

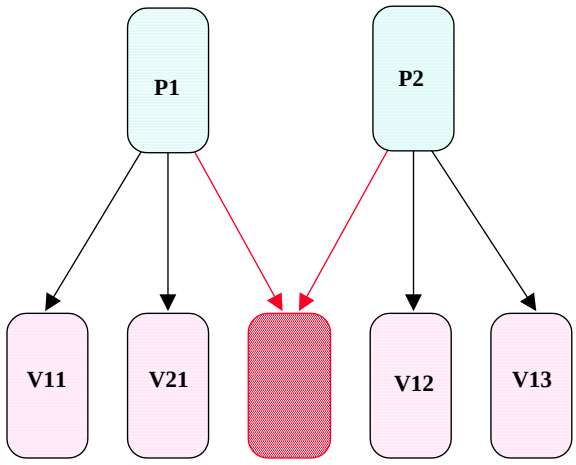

Figure IV.3 — Exemple de partage d'occurrences de membres interdit

Afin d'illustrer les concepts introduits, la figure IV.4 présente le graphe des types d'une base de données vinicole composée des articles :

- VINS décrits ci-dessus,
- BUVEURS composés des atomes numéro de buveur, nom et prénom,
- ABUS décrivant pour chaque vin la quantité bue par un buveur et la date à laquelle celle-ci a été bue,
- PRODUCTEURS définissant pour chaque vin le nom et la région du producteur,
- COMMANDES spécifiant les commandes de vins passées par les buveurs aux producteurs.

Les liens considérés sont :

- RECOLTER qui associe un producteur aux vins récoltés,
- RECEVOIR qui associe un vin aux commandes correspondantes,
- PASSER qui associe un buveur à ses commandes,
- BOIRE qui associe un buveur à une quantité de vin bue,
- PROVOQUER qui associe un vin à toutes les quantités bues par les différents buveurs.

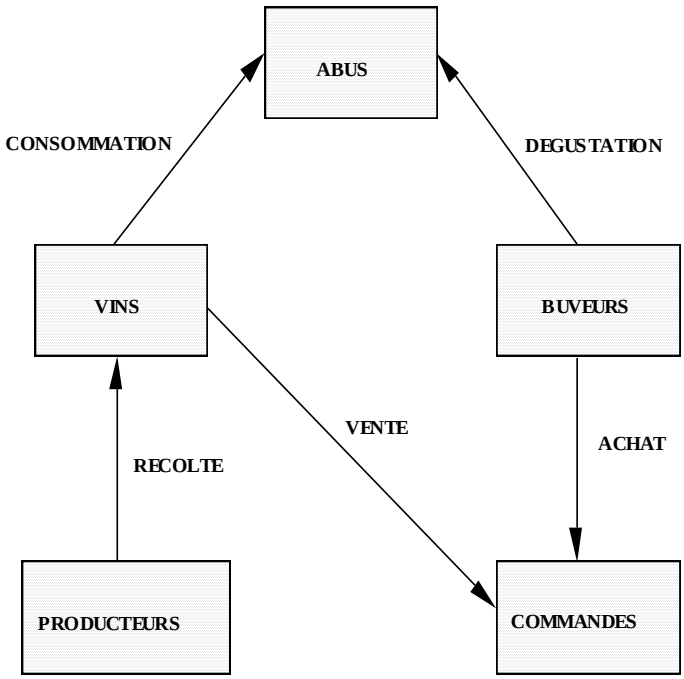

Figure IV.4 — Exemple de graphe des types

Afin de mieux faire comprendre la notion de lien, il est possible de représenter une base de données réseau par un graphe au niveau des occurrences. Dans ce cas, les articles d'une occurrence de lien sont reliés par un cycle. Par exemple, la figure IV.5 représente des occurrences des articles VINS, ABUS et BUVEURS par des carrés arrondis, et des occurrences des liens BOIRE et PROVOQUER par des chaînes d'occurrences d'articles.

<voir livre Maîtriser les BD>

Figure IV.5 — Exemple de graphe des occurrences

En CODASYL, la clause de définition d'un lien utilisée au niveau de la description d'un schéma, qui se situe bien évidemment au niveau des types, est structurée comme suit :

SET NAME IS <nom-de-lien>

OWNER IS <nom-d'article>

[**MEMBER IS** <nom-d'article>] *

Nous verrons ci-dessous les détails des sous-clauses OWNER et MEMBER. Plusieurs types de membres peuvent être spécifiés par répétition de la clause MEMBER.

CODASYL autorise la définition de liens singuliers avec une seule occurrence. Pour cela, il suffit d'utiliser la définition de propriétaire :

OWNER IS SYSTEM

Tous les articles membres sont alors chaînés entre eux dans une seule occurrence de lien (une seule liste dont la tête de liste est gérée par le système). Cela permet de rechercher un article parmi les membres comme dans une occurrence de lien normale, et aussi de chaîner des articles singuliers.

2.4 L'ordonnancement des articles dans les liens

Les articles dans les occurrences de lien sont ordonnés. L'ordre est maintenu lors des insertions et les articles sont présentés dans l'ordre d'insertion aux programmes d'application. Lors de l'insertion d'un article dont le type est membre d'un lien, il faut tout d'abord choisir l'occurrence de lien dans laquelle l'article doit être placé. Les méthodes possibles pour ce choix sont présentées ci-dessous. Ensuite, l'article doit être inséré dans la suite ordonnée des articles composants les membres de l'occurrence de lien. Les choix possibles de la position de l'article sont les suivants :

- en début (FIRST) ou en fin (LAST) de la suite des membres,
- juste avant (PRIOR) ou juste après (NEXT) le dernier article de la suite des membres auquel a accédé le programme effectuant l'insertion,

- par ordre de tri (SORTED) croissant ou décroissant d'une donnée des articles membres ; dans ce cas, la donnée doit être définie comme une clé (KEY) dans la clause membre ; les cas de doubles doivent être prévus ; de même, si le lien comporte plusieurs types d'articles, l'ordre des types doit être précisé.

La figure IV.6 représente ces diverses possibilités pour l'insertion dans une occurrence du lien BOIRE.

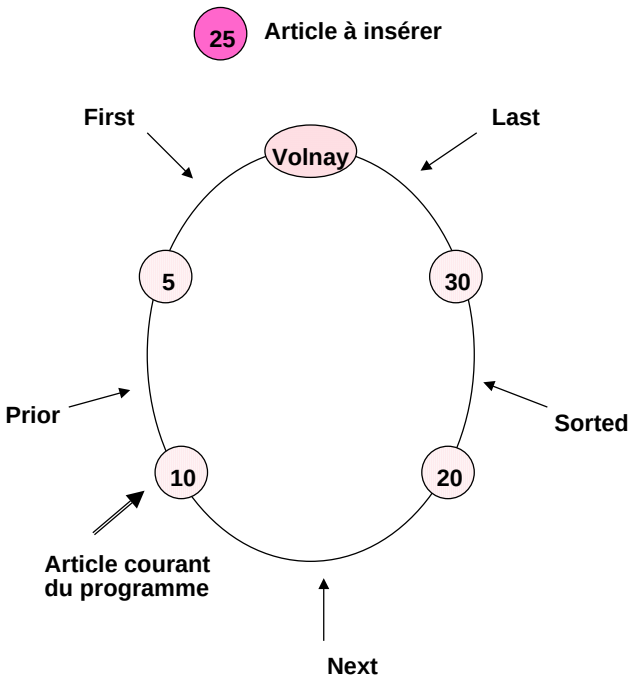

Figure IV.6 — Possibilités d'insertion dans une occurrence de lien

Ces possibilités doivent être définies au niveau du schéma à l'aide de la clause ORDER. Celle-ci est placée dans la définition du type de lien (clause SET) juste après la définition du type du propriétaire (sous-clause OWNER). Sa syntaxe simplifiée est la suivante :

ORDER IS [PERMANENT] INSERTION IS

{ FIRST | LAST | PRIOR | NEXT | SORTED <spécification de tri>}.

Une spécification de tri est définie comme suit :

<spécification de tri> ::=

[RECORD-TYPE SEQUENCE IS <nom-d'article>$^{+}$]

BY DEFINED KEYS

[DUPLICATES ARE { FIRST | LAST | NOT ALLOWED }]

L'option PERMANENT (obligatoire pour les liens triés) précise qu'un programme d'application ne peut modifier l'ordre d'un lien. Ainsi, tout

changement effectué restera local au programme et ne perturbera pas l'ordre des articles déjà enregistrés dans la base.

Si un ordre de clé est précisé à l'intérieur de chaque type d'article (option SORTED), il faut tout d'abord préciser l'ordre entre les types d'articles dans le cas où le lien est hétérogène (plusieurs types d'articles membres). C'est l'objet de la clause RECORD TYPE SEQUENCE IS. Ensuite, la clause DUPLICATES permet de spécifier si les doubles sont possibles pour les clés définies au niveau de la clause MEMBER (clause KEY ci-dessous) et, si oui, comment ils sont traités. L'option FIRST place les doubles avant ceux déjà existants ; l'option LAST les place après.

Pour chacun des types d'articles membres, si l'option SORTED BY DEFINED KEYS est retenue, il faut préciser la clé de tri ; celle-ci est spécifiée par la clause optionnelle KEY. La clé peut être utilisée pour un tri ascendant ou descendant. Elle est définie par la clause suivante :

KEY IS { ASCENDING | DESCENDING } <nom-de-donnée>

2.5 La sélection d'occurrence d'un type de lien

Il existe autant d'occurrences d'un lien que d'articles propriétaires. Lors d'une insertion ou d'une recherche, il y a donc nécessité de sélectionner l'occurrence choisie. Cela est effectué par parcours d'un chemin dans le graphe des occurrences de liens depuis un article point d'entrée, jusqu'à l'article propriétaire de l'occurrence de lien cherchée.

La sélection peut être manuelle ou automatique. Si elle est manuelle, elle est effectuée par le programme qui peut par exemple parcourir les liens de proche en proche pour trouver le bon propriétaire. Si elle est automatique, le mode de sélection doit être précisé par la clause SET SELECTION.

Nous décrivons maintenant la clause de sélection d'un chemin pour atteindre le propriétaire d'une occurrence de lien, utilisée notamment afin d'insérer automatiquement un article membre dans une occurrence de lien. Cette clause est une des plus complexe du langage de description CODASYL. Elle aboutit au lien dont une occurrence doit être sélectionnée (nom-de-lien1) en partant d'un point d'entrée (propriétaire du nom-de-lien2). La descente s'effectue de lien en lien en précisant comment doit être sélectionné le propriétaire à chaque niveau. La syntaxe simplifiée de la clause de sélection est la suivante :

SET SELECTION [FOR <nom-de-lien1>] **IS**

THRU <nom-de-lien2> **OWNER IDENTIFIED BY**

{ **APPLICATION**

| **DATA-BASE-KEY [EQUAL TO** <nom-de-paramètre1>]

| **CALC KEY [EQUAL TO** <nom-de-paramètre2>] }

[**THEN** **THRU**<nom-de-lien3>**WHERE** **OWNER IDENTIFIED BY**

{<nom-de-donnée3> [**EQUAL** **TO** <nom-de-paramètre3>]} $^{+}$] …

La première partie de la clause (THRU) permet de spécifier le mode de sélection du point d'entrée. Deux approches sont possibles :

- par application, ce qui signifie que l'article propriétaire a dû être, préalablement à la recherche ou à l'insertion, repéré par le programme d'application, comme nous le verrons lors de l'étude du langage de manipulation ;

- par clé (DATA-BASE-KEY ou CALC KEY) fournie par le programme en paramètre ; le premier type de clé est une adresse base de données d'article (adresse relative dans le fichier, gérée par le système) et le second une clé de hachage (CALC KEY) servant au placement de l'article dans le fichier par application d'une fonction de hachage, comme nous le verrons ci-dessous.

La deuxième partie (THEN THRU) est optionnelle dans le cas où l'on désire parcourir un chemin de longueur supérieure à un dans le graphe des liens. Elle permet, de manière récursive lorsqu'elle est répétée, de sélectionner une occurrence de lien par recherche du propriétaire dans les membres de l'occurrence précédemment sélectionnée. Cette recherche s'effectue par balayage de l'occurrence de lien de niveau supérieur jusqu'au premier article ayant pour valeur de la donnée nom-de-donnée3 la valeur contenue dans nom-de-paramètre3 ; cette donnée (nom-de-donnée3) doit être discriminante dans l'occurrence de lien (pas de doubles autorisés). Ainsi, il est possible de faire choisir l'occurrence de lien dans lequel un article est automatiquement inséré par le SGBD, à partir d'un point d'entrée préalablement sélectionné et de paramètres fournis par le programme.

2.6 Les options d'insertion dans un lien

Lors de l'insertion d'un article dans la base, celui-ci peut, pour chacun des liens dont son type d'article est déclaré membre :

- être inséré automatiquement dans la bonne occurrence du lien sélectionné par le système en accord avec la SET SELECTION (option INSERTION IS AUTOMATIC) ;

- ne pas être inséré automatiquement par le système; dans ce cas (option INSERTION IS MANUAL), le programme devra, s'il le désire, faire l'insertion par une commande spéciale du langage de manipulation que nous étudierons plus loin..

De plus, une contrainte d'intégrité spécifique peut être précisée : l'association entre deux types d'articles reliés par un lien est déclarée « obligatoire » (option MANDATORY) ou « facultative » (option OPTIONAL). Dans le premier cas, tout article du type membre d'un lien sera forcément membre d'une occurrence de

ce lien, alors qu'il ne le sera pas forcément dans le second. L'option MANDATORY correspond à un lien fort (qui doit forcément exister) alors que l'option OPTIONAL correspond à un lien faible (qui peut exister).

Ces options d'insertion sont précisées pour chaque type de membre, après la clause MEMBER, par la clause :

INSERTION IS { AUTOMATIC | MANUAL }

RETENTION IS { MANDATORY | OPTIONAL }

2.7 Le placement des articles

Une base de données CODASYL est placée dans un ensemble de fichiers appelé AREA (ou REALM dans la nouvelle version). Ces fichiers sont soit des fichiers relatifs (adressage par numéro de page et par numéro d'octet dans la page), soit des fichiers aléatoires (adresse relative calculée par une fonction de hachage). Chaque article est repéré dans la base par une **clé base de données** (*database key*) qui lui est affectée à sa création et permet de l'identifier jusqu'à sa disparition. Un SGBD CODASYL gère donc l'identité d'objets par des clés base de données.

> **Notion IV.5 : Clé Base de Données (D*atabase key*)**
>
> Adresse invariante associée à un article lors de sa création et permettant de l'identifier sans ambiguïté.

Bien que le comité DBTG CODASYL n'ait pas spécifié le format des clés base de données, la plupart des systèmes réseaux attribuent une place fixe aux articles, si bien qu'une clé base de données peut être un numéro de fichier, suivi d'un numéro de page et d'un déplacement dans la page permettant de retrouver l'en-tête de l'article. Certains systèmes gèrent en fin de page un index des en-têtes d'articles contenus dans la page, si bien que le déplacement dans la page est simplement le numéro de l'en-tête en fin de page.

Le **placement** d'un article consiste à calculer son adresse dans la base, ainsi que sa clé base de données qui en découle en général directement. Le mode de placement est défini dans le schéma pour chaque type d'article selon plusieurs procédés possibles.

> **Notion IV.6 : Placement CODASYL (*CODASYL location mode*)**
>
> Méthode de calcul de l'adresse d'un article et d'attribution de la clé base de données lors de la première insertion.

De manière surprenante, la clé base de données peut être fournie directement par l'utilisateur comme un paramètre du programme. Ce mode, appelé **placement direct** (mot clé DIRECT), est en général réservé aux programmeurs système. Le mode le plus facilement utilisable est le placement aléatoire classique : une clé, pas forcément discriminante, est précisée et le système calcule l'adresse de l'article à l'aide d'une procédure de hachage. Ce mode de placement, appelé **placement calculé**, est spécifié par les mots clés CALC USING.

Un mode plus complexe, mais permettant en général de bonnes optimisations, est le mode par lien, spécifié par le mot clé VIA. Il possède deux variantes selon que l'article est placé dans le même fichier que son propriétaire via le lien considéré, ou dans un fichier différent. Dans le premier cas, l'article est placé **à proximité** du propriétaire, alors que dans le second il est placé dans un autre fichier que celui du propriétaire **par homothétie**.

Le placement à proximité consiste à placer l'article aussi près que possible du propriétaire du lien retenu pour le placement, dans la même page ou dans la première page voisine ayant de la place disponible.

Le placement par homothétie consiste à calculer l'adresse de la page dans laquelle on place l'article (dénotée Adresse Article AA) à partir de l'adresse de la page du propriétaire (dénotée Adresse Propriétaire AP) par la formule AA =AP * TA/TP, où TA désigne la taille du fichier contenant l'article et TP la taille du fichier contenant le propriétaire. L'avantage de cette méthode est qu'en général tous les articles ayant le même propriétaire via le lien considéré sont placés dans la même page (ou des pages voisines). Ainsi la méthode par homothétie réduit le temps de parcours des membres d'une occurrence de lien sans accroître le temps de parcours du type d'article membre, alors que la méthode par proximité réduit le temps de parcours des occurrences de lien mais en général accroît celui du type d'article membre.

La figure IV.7 illustre les différents types de placements couramment utilisés : placement calculé (CALC), à proximité et par homothétie. Le choix entre ces différents modes est laissé à l'administrateur système, qui doit chercher à optimiser les performances des accès les plus fréquents aux données.

<voir livre Maîtriser les BD>

Figure IV.7 — Différents types de placements possibles

Les diverses possibilités de placement illustrées ci-dessus sont définies au niveau du schéma par la clause LOCATION MODE. Il existe en fait quatre modes possibles, le mode SYSTEM spécifiant qu'un algorithme définit par le système est choisi. On retrouve les modes décrits ci-dessus, DIRECT par adresse calculée par le programme, CALC par clé et VIA par lien. Le fichier choisi peut être celui du propriétaire ou un autre dans le cas VIA, ce qui différencie le placement à proximité et le placement par homothétie. Dans tous les cas, le fichier choisi est précisé par la clause WITHIN. La syntaxe de la clause de placement est la suivante :

```
LOCATION MODE IS
{     SYSTEM
|     DIRECT <NOM-DE-PARAMETRE>
|     CALC USING <NOM-DE-DONNEE>
      [ DUPLICATES ARE [NOT] ALLOWED ]
|     VIA <NOM-DE-LIEN> SET }

WITHIN { <NOM-DE-FICHIER> | AREA OF OWNER }
```

2.8 Exemple de schéma

Afin d'illustrer le langage de définition de schéma CODASYL, la figure IV.8 propose un schéma possible pour la base de données dont le graphe des types a été présenté figure IV.4. La base a été placée dans trois fichiers (F-BUVEURS, F-PRODUCTEURS et F-COMMANDES). Les articles ABUS et VINS sont placés à proximité des propriétaires, respectivement BUVEURS et PRODUCTEURS, eux-mêmes placés par hachage. Les articles COMMANDES sont placés dans le fichier F-COMMANDES par homothétie avec le fichier F-BUVEURS. Des choix sont faits pour les ordres d'insertion, les contraintes de rétention et les modes de sélection des liens. Ces choix devront être respectés lors de l'écriture des programmes de manipulation.

<Voir livre Maîtriser les BD>

Figure IV.8 — Exemple de schéma CODASYL

3. LE LANGAGE DE MANIPULATION COBOL-CODASYL

Le langage de manipulation de données du CODASYL est fortement lié à COBOL, bien que généralisé et utilisable depuis d'autres langages de 3e génération tel Fortran.

3.1 Sous-schéma COBOL

Un schéma externe en CODASYL est appelé **sous-schéma**. La notion de sous-schéma est en général plus restrictive que celle de schéma externe. Il s'agit d'un sous-ensemble exact du schéma sans restructuration possible. Au niveau d'un sous-schéma, l'administrateur ne peut qu'omettre des types d'articles, des fichiers (area), des liens et des données déclarés dans le schéma. Certains systèmes permettent cependant de définir des données virtuelles, non stockées dans la base, mais calculées par le SGBD à partir de données de la base.

> **Notion IV.7 : Sous-schéma (*Sub-schema*)**
>
> Sous-ensemble du schéma vu par un programme d'application, spécifiant la vision externe de la base par le programme.

Outre la possibilité de ne définir qu'une partie des données, des articles, des liens et des fichiers, d'autres variations importantes peuvent exister entre le schéma et un sous-schéma. En particulier :

- l'ordre des atomes dans un article peut être changé,
- les caractéristiques (types) d'un atome peuvent être changées,
- les clauses SET SELECTION peuvent être redéfinies,
- les noms de types d'articles, d'attributs et de liens peuvent être changés.

Un sous-schéma COBOL se compose en principe de trois divisions : une division de titre (title division) qui nomme le sous-schéma et le relie au schéma, une division de transformation (mapping division) qui permet de définir les synonymes et une division des structures (structure division) où sont définis les fichiers (appelés REALM en COBOL car le mot AREA est utilisé à d'autres fins), les articles et les liens vus par le programme. Afin d'illustrer le concept de sous-schéma, la figure IV.9 propose un sous-schéma de la base vinicole que pourrait définir un administrateur pour des programmes s'intéressant seulement aux articles buveurs, commandes et abus (à l'exception du numéro de buveur).

<Voir livre Maîtriser les BD>

Figure IV.9 — Exemple de sous-schéma COBOL

3.2 La navigation CODASYL

Une fois qu'un schéma et des sous-schémas ont été définis, des programmes d'applications peuvent être écrits. Ceux-ci invoquent le système de gestion de bases de données à l'aide des verbes du langage de manipulation qui sont inclus dans un programme COBOL, ou dans un autre langage (C, FORTRAN, PL1 sont en particulier supportés). Les verbes de manipulation peuvent être classés en quatre types :

- la recherche d'articles (FIND),
- les échanges d'articles (GET, STORE),
- la mise à jour (ERASE, CONNECT, DISCONNECT, MODIFY),
- le contrôle des fichiers (READY, FINISH).

Les échanges d'articles entre le SGBD et un programme d'application s'effectuent par une zone de travail tampon appelée USER WORKING AREA, dans laquelle

les articles fournis par le SGBD sont chargés (GET), et dans laquelle ceux fournis par le programme sont placés avant d'être rangés dans la base (STORE). Chaque atome ou article décrit dans le sous-schéma a une place fixée dans la zone de travail, affectée lors du chargement du programme. Chaque objet peut être référencé par le programme en utilisant son nom défini dans le sous-schéma.

La recherche d'articles ne provoque pas d'échange de données entre le programme et le SGBD. Seuls des pointeurs associés au programme et à des collections d'articles, appelés **curseurs**, sont déplacés par les ordres de recherche (FIND).

> **Notion IV.8 : Curseur (*Currency*)**
>
> Pointeur courant contenant la clé base de données du dernier article manipulé d'une collection d'articles, et permettant au programme de se déplacer dans la base.

Il existe plusieurs curseurs associés à un programme en exécution, chacun permettant de mémoriser l'adresse du dernier article manipulé dans une collection d'articles (en général, un type) :

- un curseur indique l'article courant du programme, c'est-à-dire en principe le dernier article lu, écrit ou simplement cherché par le programme ;
- un curseur indique l'article courant de chaque type d'article référencé ;
- un curseur indique l'article courant de chaque type de lien référencé ;
- un curseur indique enfin l'article courant de chaque type de fichier référencé.

Le nombre de curseurs associés à un programme se comptabilise en fonction des types du sous-schéma. Il est obtenu par la formule : (1 + Nombre de types d'articles + Nombre de type de liens + Nombres de fichiers). Les curseurs sont déplacés grâce aux divers types de FIND que nous allons étudier. Seul l'article pointé par le curseur du programme peut être lu par le programme en zone de travail. Ainsi, un programme se déplace dans la base en déplaçant son curseur : on dit qu'il navigue [Bachman73].

3.3 La recherche d'articles

La recherche d'articles consiste donc à déplacer les curseurs dans la base. Elle s'effectue à l'aide de l'instruction FIND. L'exécution d'une telle instruction déplace toujours le curseur du programme, mais il est possible d'empêcher sélectivement le déplacement des autres curseurs. S'il n'a pas été spécifié qu'un curseur ne doit être déplacé, celui-ci est repositionné dès qu'un article de la collection à laquelle il est associé (type d'articles, liens, fichiers) est manipulé (lu, écrit ou traversé).

Le format général de l'instruction de recherche est :

```
FIND <EXPRESSION-DE-SÉLECTION> RETAINING CURRENCY FOR
{MULTIPLE
|     REALM
|     RECORD
|     SETS
|     <NOM DE LIEN>+ }
```

L'expression de sélection spécifie le critère de recherche. La rétention du déplacement de tous les curseurs, à l'exception de celui du programme, s'effectue par l'option RETAINING CURRENCY FOR MULTIPLE. Les autres choix de rétention de curseurs peuvent être combinés. Ci-dessous, nous examinons les différents types de FIND résultant des diverses expressions de sélections possibles.

3.3.1 La recherche sur clé

Comme l'indique le paragraphe IV.2, un article possède toujours une clé base de données (DATA BASE KEY) et peut posséder une clé de hachage (avec ou sans double) s'il a été placé en mode calculé. On obtient ainsi trois possibilités de recherche sur clés dont voici les syntaxes :

- Recherche connaissant la clé base de données (adresse invariante)

  FIND <nom-d'article> **DBKEY IS** <nom-de-paramètre>

- Recherche connaissant la clé calculée unique (clé de hachage)

  FIND ANY <nom-d'article>

Dans ce cas, la valeur de la clé de recherche doit préalablement être chargée par le programme en zone de travail.

- Recherche connaissant la clé calculée avec doubles

  FIND DUPLICATE <nom-d'article>

Plusieurs FIND DUPLICATE permettront de retrouver les doubles successivement.

3.3.2 La recherche dans un fichier

Cette recherche est avant tout séquentielle ou en accès direct. Il est ainsi possible de sélectionner le premier article (option FIRST), le dernier (option LAST), l'article suivant (option NEXT) ou précédent (option PRIOR) l'article courant, ainsi que le i^{e} article d'un fichier. Le numéro de l'article à sélectionner peut être spécifié par un paramètre du programme. Le format de l'instruction de recherche dans un fichier est :

FIND { **FIRST** | **LAST** | **NEXT** | **PRIOR** | <i> | <paramètre> }

<nom-d'article> **WITHIN** <nom-de-fichier>

3.3.3 La recherche dans une occurrence de lien

De même que dans un fichier, il est possible de sélectionner le premier article, le dernier, l'article suivant ou précédent l'article courant et le i^e article de l'occurrence courante d'un lien. Le format de l'instruction est identique à celui de la recherche en fichier :

FIND **{ FIRST | LAST | NEXT | PRIOR |** <i> | <paramètre> }

<nom-d'article> **WITHIN** <nom-de-lien>

Il est également possible de rechercher un article à partir d'une valeur de donnée dans l'occurrence courante d'un lien. La donnée dont la valeur est utilisée pour ce type de recherche associative dans une occurrence de lien est citée en argument du mot clé USING. Cette valeur doit bien sûr être préalablement chargée en zone de travail. Selon que l'on recherche la première (ou l'unique) occurrence d'article ou la suivante, on emploie :

FIND <nom-d'article> **WITHIN** <nom-de-lien>

USING <nom-de-donnée>$^+$

FIND DULICATE WITHIN <nom-de-lien>

USING <nom-de-donnée>$^+$

Un lien peut être parcouru depuis un article membre vers le propriétaire, donc en sens inverse de l'arc qui le représente dans le diagramme de Bachman. Ainsi, il est possible de sélectionner le propriétaire de l'occurrence courante d'un lien par la commande :

FIND OWNER WITHIN <nom-de-lien>

Une telle instruction permet en général de passer depuis un membre d'un lien à un propriétaire, donc de remonter les arcs du graphe des types d'une base de données CODASYL.

Il est aussi possible de tester des conditions concernant l'appartenance de l'article courant d'un programme à un lien. La condition est vraie si l'article courant du programme est propriétaire (option OWNER), membre (option MEMBER) ou plus généralement participant (option TENANT) à l'intérieur d'une occurrence du lien cité. La forme de la commande est :

IF [NOT] <nom-de-lien> { **OWNER | MEMBER | TENANT** }

EXECUTE <instructions>

Il est enfin possible de tester si une occurrence de lien sélectionnée par un article propriétaire ne possède aucun membre en utilisant la commande :

IF <nom-de-lien> **IS [NOT] EMPTY EXECUTE** <instructions>

3.3.4 Le positionnement du curseur de programme

A chaque recherche, le curseur du programme est positionné. Par suite, les FIND provoquent des changements successifs de la position du curseur du programme : on dit que le programme navigue dans la base CODASYL [Bachman73]. Après des navigations successives, il peut être utile de revenir se positionner sur l'article courant d'un type d'article d'un fichier ou d'un lien. Cela peut être fait à l'aide de la commande :

FIND CURRENT [<nom-d'article>]

[**WITHIN** { <nom-de-fichier> | <nom-de-lien> }]

Cette commande a donc pour effet de forcer le curseur du programme à celui du nom-d'article spécifié, ou à celui du fichier ou du lien spécifié, éventuellement selon un type d'article précisé.

3.4 Les échanges d'articles

L'article pointé par le curseur d'un programme peut être amené en zone de travail du programme pour traitement. Seules certaines données de cet article peuvent être lues. Cette opération s'effectue à l'aide de la commande :

GET { [<type-d'article>] | {<nom-de-donnée>} * }

Les arguments peuvent être soit le type d'article qui doit alors être le même que celui de l'article pointé par le curseur du programme, soit une liste des données du type de l'article courant que l'on souhaite amener en zone de travail. Si aucun nom de donnée n'est précisé, toutes les données sont lues.

Le rangement d'un article dans la base s'effectue, après chargement de l'article en zone de travail, par exécution de la commande STORE. En principe, tous les curseurs sont modifiés par la commande STORE : tous pointent après exécution sur le nouvel article stocké. Il est possible de retenir le déplacement de certains curseurs en utilisant l'option RETAINING, de manière identique à FIND. Le format de la commande STORE est le suivant :

STORE <nom d'article> **RETAINING CURRENCY FOR**

{ **MULTIPLE**

| **REALM**

| **RECORD**

| **SETS**

| <nom de lien>$^{+}$ }

3.5 Les mises à jour d'articles

Elles incluent à la fois les suppressions et les modifications de données. Alors que la suppression est simple, la modification soulève des problèmes au niveau des liens qui peuvent ou non être modifiés.

3.5.1 Suppression d'articles

Avant de supprimer un article, il faut bien sûr le rechercher. Il devient alors le courant du programme. La suppression de l'article courant d'un programme s'effectue par la commande :

ERASE [ALL] [<nom-d'article>].

Si cet article est propriétaire d'occurrences de lien, tous ses descendants sont également supprimés seulement dans le cas où le mot clé ALL est précisé. Une suppression est ainsi cascadée à tous les articles dépendants, et cela de proche en proche. Le nom d'article permet optionnellement au système de vérifier le type de l'article courant à détruire.

3.5.2 Modification d'articles

La modification de l'article courant d'un programme s'effectue en principe après avoir retrouvé l'article à modifier comme le courant du programme. Les données à modifier doivent être placées en zone article dans le programme (zone de travail). Seules certaines données sont modifiées, en principe celles spécifiées par la commande ou toutes celles dont la valeur en zone article est différente de celle dans la base. Les liens spécifiés sont changés en accord avec les clauses de sélection de liens (SET SELECTION) précisées dans le schéma. Le format de la commande est le suivant :

MODIFY { [<nom-d'article>] | [<nom-de-donnée>] }

[INCLUDING { ALL | ONLY <nom-de-lien>$^{+}$ } **MEMBERSHIP**]

Les données à modifier peuvent être précisées ou non ; dans le dernier cas, le système modifie toutes celles qui sont changées dans l'article en zone de travail. On doit aussi préciser les modifications à apporter aux liens dont l'article est membre ; plusieurs options sont possibles. Il est possible de demander la réévaluation, en accord avec la clause du schéma SET SELECTION de toutes (INCLUDING ALL) ou de certaines (INCLUDING liste de noms de liens) appartenances à des occurrences de liens. Il est aussi possible de ne modifier aucune donnée, mais seulement les appartenances aux liens (option MODIFY… ONLY …).

3.5.3 Insertion et suppression dans une occurrence de lien

L'insertion d'un article dans une occurrence de lien peut être effectuée par le SGBD ou par le programme, selon le choix lors de la définition du schéma. Deux commandes permettent d'insérer et d'enlever l'article courant d'un programme d'une occurrence de lien : CONNECT et DISCONNECT. Leur utilisation nécessite que le mode d'insertion dans le lien ait été défini comme manuel (MANUAL) ou alors que l'appartenance au lien soit optionnelle (OPTIONAL). La syntaxe de ces commandes est la suivante :

CONNNECT [<nom-d'article>] **TO** <nom-de-lien>.

DISCONNECT [<nom-d'article>] **FROM** <nom-de-lien>.

Le nom d'article permet de vérifier le type de l'article courant.

3.6 Le contrôle des fichiers

Avant de travailler sur un fichier, un programme doit l'ouvrir en spécifiant le mode de partage souhaité. Le fichier peut être ouvert en mode exclusif (seul l'utilisateur peut alors accéder) ou en mode protégé (les accès sont alors contrôlés au niveau des articles), à fin de recherche (RETRIEVAL) ou de mise à jour (UPDATE). Cette ouverture s'effectue à l'aide de la commande :

READY { <nom-de-fichier> [**USAGE-MODE IS**

{ **EXCLUSIVE | PROTECTED } { RETRIEVAL | UPDATE** }] } $^{+}$

En fin de travail sur un fichier, il est nécessaire de le fermer à l'aide de la commande :

FINISH { <nom-de-fichier> } $^{+}$

3.7 Quelques exemples de transaction

Voici différents types d'accès réalisés par des transactions simples. Ces transactions utilisent le sous-schéma CLIENT de la figure VI.9. La signification des transactions apparaît en légende de la figure correspondante.

<Voir livre Maîtriser les BD>

Figure IV.10 — Accès sur clé calculée unique

<Voir livre Maîtriser les BD>

Figure IV.11 — Accès séquentiel à un fichier et à des occurrences de lien

<Voir livre Maîtriser les BD>

Figure IV.12 — Accès associatif dans une occurrence de lien

<Voir livre Maîtriser les BD>

Figure IV.13 — Accès au propriétaire et utilisation d'une condition

<Voir livre Maîtriser les BD>

Figure IV.14 — Suppression d'un article et de ses descendants

<Voir livre Maîtriser les BD>

Figure IV.15 — Modification d'un article

<Voir livre Maîtriser les BD>

4. LE MODÈLE HIÉRARCHIQUE

Les bases de données modélisent des informations du monde réel. Puisque le monde réel nous apparaît souvent au travers de hiérarchies, il est normal qu'un des modèles les plus répandus soit le modèle hiérarchique. Quelques systèmes sont encore basés sur ce modèle [MRI74, IBM78]. Vous trouverez une présentation détaillée du modèle hiérarchique dans [Tsichritzis76]. Nous résumons ici les aspects essentiels.

4.1 Les concepts du modèle

Le modèle hiérarchique peut être vu comme un cas particulier du modèle réseau, l'ensemble des liens entre types d'articles devant former des graphes hiérarchiques. Cependant, les articles ne peuvent avoir de données répétitives. De plus, le modèle introduit en général des concepts spécifiques afin de modéliser les objets. Un **champ** est exactement l'équivalent d'un atome du modèle réseau. Il s'agit d'une donnée élémentaire à valeur simple.

> **Notion IV.9 : Champ (*Field*)**
>
> Plus petite unité de données possédant un nom.

Un **segment** correspond à un article sans groupe répétitif. Une occurrence de segment est en général de taille fixe. Les champs d'un segment sont tous au même niveau, si bien qu'une occurrence de segment est parfois qualifiée d'article plat. Un segment peut avoir un champ discriminant appelé **clé**. La valeur de la clé permet alors de déterminer une occurrence unique dans le segment.

> **Notion IV.10 : Segment (*Segment*)**
>
> Collection de champs rangés consécutivement dans la base, portant un nom et dont une occurrence constitue l'unité d'échange entre la base de données et les applications.

Les segments sont reliés par des liens de 1 vers N qui à un segment père font correspondre N segments fils (N est un entier positif quelconque), aussi bien au

niveau des types qu'au niveau des occurrences. Ainsi, un type de segment possède en général plusieurs types de segments descendants. De même, une occurrence de segment est reliée à plusieurs occurrences de chacun des segments descendants. Pour représenter une descendance de segments reliés par des associations père-fils, on construit des **arbres de segments**.

> **Notion IV.11 : Arbre de segments (*Segment tree*)**
>
> Collection de segments reliés par des associations père-fils, organisée sous la forme d'une hiérarchie.

Les types de segments sont donc organisés en arbre. Un type racine possède N1 types fils, qui à leur tour possèdent chacun N2 types fils, et ainsi de suite jusqu'aux segments feuilles. Il est aussi possible de considérer une occurrence d'un arbre de segments : une occurrence d'un segment racine possède plusieurs occurrences de segments fils. Parmi celles-ci, certaines sont d'un premier type, d'autres d'un second, etc. A leur tour, les occurrences des fils peuvent avoir des fils, et ainsi de suite jusqu'aux occurrences des feuilles.

Finalement, une **base de données hiérarchique** peut être considérée comme un ensemble d'arbres, encore appelé forêt, dont les nœuds sont des segments. La définition s'applique aussi bien au niveau des types qu'au niveau des occurrences. Les arbres sont en principe indépendants. Chaque arbre possède un segment racine unique, des segments internes et des segments feuilles. Le niveau d'un segment caractérise sa distance à la racine.

> **Notion IV.12 : Base de données hiérarchique (*Hierarchical data base*)**
>
> Base de données constituée par une forêt de segments.

Une représentation par un graphe d'une base de données hiérarchique découle de la définition. Les nœuds du graphe sont les segments alors que les arcs représentent les associations père-fils ; ces arcs sont orientés du père vers le fils. Il est possible de considérer un graphe des types ou un graphe d'occurrences (voir figure IV.16). Les deux graphes sont bien sûr des forêts, c'est-à-dire des ensembles de hiérarchies sans lien entre elles. Un graphe des types n'est pas différent d'un diagramme de Bachman d'une base de données réseau ; cependant, de chaque segment est issu au plus un lien allant vers son fils. Les liens ne sont pas nommés.

<Voir livre Maîtriser les BD>

Figure IV.16 — Graphe hiérarchique de type et d'occurrence

A titre d'exemple, nous avons défini une base de données hiérarchique à partir de la base vinicole. Il n'est évidemment pas possible de représenter un réseau de liens tel que défini figure IV.4. Afin de ne pas perdre d'informations, on est conduit à dupliquer certaines données, par exemple le cru du vin dans les segments ABUS et le nom du buveur dans les segments COMMANDES (Champ CLIENT). Comme les segments ne sont pas hiérarchiques, le type d'article VINS

doit être éclaté en deux segments CRUS et MILLESIMES. La figure IV.17 schématise une représentation hiérarchique de la base vinicole.

<Voir livre Maîtriser les BD>

Figure IV.17 — Exemple de base hiérarchique

Certains modèles hiérarchiques autorisent les liens entre arbres pour permettre de modéliser des associations de 1 vers N sans avoir à dupliquer des segments. Tout segment d'un arbre peut alors pointer vers un segment d'un autre arbre. Cela peut être limité à un seul pointeur par segment : on parle de lien frère, pour distinguer ce lien du lien vers le fils. On aboutit alors à des modèles hiérarchiques étendus dont les possibilités se rapprochent de celles du modèle réseau [IBM78].

4.2 Introduction au langage DL1

DL1 est un langage de manipulation de bases de données hiérarchiques proposé par IBM dans son produit IMS [IBM78]. Nous allons ci-dessous en donner quelques principes afin d'illustrer la manipulation de bases hiérarchiques. Le langage DL1 est un langage très complexe qui permet de naviguer dans une base hiérarchique à partir de programmes écrits en PL1, COBOL ou FORTRAN. Vous trouverez diverses présentations de ce langage dans [Tsichritzis76, Cardenas85].

Les recherches s'effectuent en naviguant dans les arbres d'occurrences. Un ordre de parcours des arbres de la racine vers les fils, et de gauche à droite, est imposé. Plus précisément, cet ordre est défini comme suit :

1. Visiter le segment considéré s'il n'a pas déjà été visité.

2. Sinon, visiter le fils le plus à gauche non précédemment visité s'il en existe un.

3. Sinon, si tous les descendants du segment considéré ont été visités, remonter à son père et aller à 1.

Cet ordre de parcours des arbres est illustré figure IV.18 sur une forêt d'occurrences de l'un des arbres des types de segments d'une base hiérarchique. Une telle forêt est supposée avoir une racine virtuelle de tous les arbres pour permettre le passage d'un arbre à un autre, représentée par une ligne sur le schéma. L'ordre de parcours est utilisé pour les recherches d'occurrences de segments satisfaisant un critère, mais aussi pour les insertions de nouvelles occurrences.

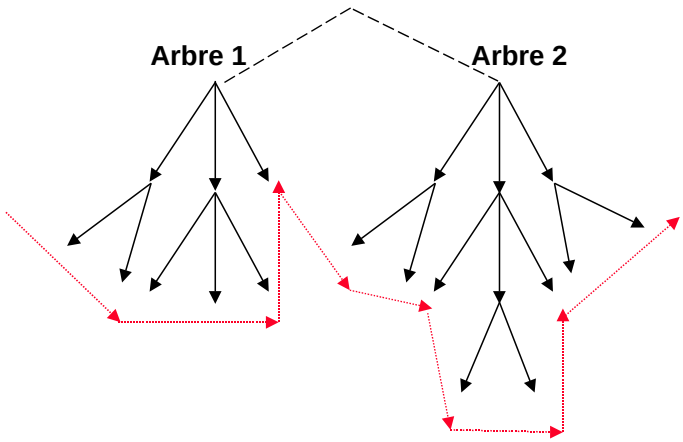

Figure IV.18 — Ordre de parcours des arbres en DL1

DL1 est un langage de manipulation navigationnel, en ce sens que des curseurs permettent de mémoriser un positionnement sur la base. Ces curseurs, appelés PCB, sont stockés dans des structures de données passées en arguments des appels DL1 effectués comme des appels de procédure depuis les programmes utilisateurs. Les PCB permettent en particulier de mémoriser des positionnements sur la base de données par des **clés base de données** (adresses d'occurrences de segments). Une clé base de données est souvent la concaténation des clés des segments traversés depuis la racine jusqu'au segment de niveau le plus bas sélectionné. Les PCB permettent aussi de récupérer un code réponse STATUS du système après chaque appel à une commande de recherche ou de mise à jour. La figure IV.19 illustre la structure simplifiée d'un PCB. Outre le nom de la base et le code réponse, on notera la présence de la clé base de données indiquant le positionnement courant du PCB.

```
01 PCB /* BLOC DE CONTROLE DE PROGRAMME */
02 NOM DE BASE /* BASE DE DONNEES REFERENCEES */
02 NIVEAU /* NIVEAU MAXIMUM CONSIDERE */
02 STATUS /* CODE REPONSE AUX REQUETES */
02 LONGUEUR CLE /*LONGUEUR DU CHAMP SUIVANT */
02 CLE BASE DE DONNES/*POSITIONNEMENT COURANT SUR LA BASE */
```

Figure IV.19 — Curseurs de programme en DL1

Une qualification de chemin peut être associée à un appel DL1. Elle se compose d'une suite de qualifications de segments (Search Segment Argument); une qualification de segments est une structure dont une vue simplifiée est représentée figure IV.20. Il s'agit en fait d'une expression logique parenthésée ou conjonctive (ET de OU) de critères de comparaisons portant sur un segment. Outre le nom du segment concerné, chaque prédicat élémentaire contient un code commande permettant de spécifier des options telles que recherche ou insertion demandée, un nom de champ, un comparateur et une valeur. Les prédicats élémentaires sont connectés par le connecteur logique. Ils se suivent dans un SSA. Des parenthèses peuvent être utilisées pour marquer des priorités.

```
01    SSA /*(QUALIFICATION DE SEGMENTS)*/
02    NOM-DE-SEGMENT
02    CODE-COMMANDE /*( DIVERSES OPTIONS POSSIBLES)*/
02    NOM-DE-CHAMP
02    OPERATEUR-DE-COMPARAISON /*(<, ≤, >, ≥, =, ≠)*/
02    VALEUR
02    CONNECTEUR /*(AND, OR) */
 …
```

Figure IV.20 — Qualification de segment en DL1

Un ensemble de qualifications de segments suivant la hiérarchie des segments constitue une **qualification de chemin**. Une telle qualification spécifie un chemin depuis la racine jusqu'au segment cherché dans un arbre du graphe des types en permettant de sélectionner certaines occurrences de segment à chaque niveau. Certains niveaux peuvent être absents, ce qui implique un balayage séquentiel du segment de ce niveau. Le code commande permet en particulier de spécifier si l'occurrence de segment sélectionnée doit ou non être lue en zone de travail lors de l'utilisation du SSA dans une commande de recherche au SGBD. Lors d'une insertion d'une hiérarchie de segments, ce code permet de préciser si le segment du niveau doit être inséré. A partir de ces éléments (PCB et qualification de chemin), différents types d'appels au SGBD peuvent être effectués. Les plus importants sont explicités ci-dessous.

GET UNIQUE (GU) permet de rechercher directement la première occurrence de segment satisfaisant la qualification, par exemple pour une recherche sur clé. Le PCB passé en argument est utilisé pour mémoriser la clé base de données du segment trouvé ; le contenu des segments retrouvés à chaque niveau est chargé en mémoire pour les segments dont le SSA demande la lecture dans le code commande.

GET NEXT (GN) permet de rechercher l'occurrence de segment suivant celle mémorisée dans le PCB (en général la dernière trouvée) satisfaisant la qualification si elle existe. Le parcours du graphe s'effectue selon l'ordre indiqué ci-dessus en balayant séquentiellement chaque occurrence de segment, sans tenir compte des liens de filiation.

GET NEXT WITHIN PARENT (GNP) a la même fonction que GET NEXT, mais il autorise la recherche seulement à l'intérieur des descendants du segment courant. Il permet donc de parcourir un lien depuis un parent.

INSERT (ISRT) permet d'insérer un nouveau segment en-dessous du parent sélectionné par la qualification.

DELETE (DLET) permet de supprimer l'occurrence du segment courant. Celle-ci a dû être précédemment sélectionnée par une instruction spéciale du type GET HOLD UNIQUE (GHU) ou GET HOLD NEXT (GHN) ou GET HOLD NEXT WITHIN PARENT (GHNP) afin d'assurer la non sélection simultanée par plusieurs utilisateurs. La suppression détruit l'occurrence de segment

précédemment sélectionnée ainsi que tous ses descendants s'il en existe. Il s'agit donc d'une suppression en cascade.

REPLACE (REPL) permet de modifier l'occurrence du segment courant. Celle-ci a dû être précédemment lue par une instruction spéciale du type GET HOLD, comme pour la suppression. La clé d'un segment n'est pas modifiable par la commande REPLACE.

4.3 Quelques exemples de transactions

Voici quelques types d'accès par des transactions simples. La syntaxe utilisée est très loin de la syntaxe pénible de DL1 [IBM78]. Il s'agit en fait d'une abstraction de cette syntaxe compatible avec les commandes décrites ci-dessus.

<voir livre Maîtriser les BD>

Figure IV.21 — Accès sur clé

<voir livre Maîtriser les BD>

Figure IV.22 — Balayage des descendants d'une occurrence de segment

<voir livre Maîtriser les BD>

Figure IV.23 — Mise à jour d'une occurrence de segment

5. CONCLUSION

Dans ce chapitre, nous avons présenté les principaux modèles d'accès, c'est-à-dire ceux directement issus de la modélisation d'organisation d'articles stockés dans des fichiers et reliés entre eux par des pointeurs. Ces modèles sont encore très utilisés : une part significative des grandes bases de données sont sans doute encore hiérarchiques ou organisées en réseaux. Ces modèles sont aussi très performants car très proches du niveau physique. Le modèle réseau permet des organisations de liens plus générales que le modèle hiérarchique, bien que celui-ci ait souvent été étendu par des liens inter-hiérarchies.

Compte tenu de l'accroissement des possibilités des machines, les modèles d'accès sont aujourd'hui inadaptés en tant que modèles conceptuels ou externes de données. Ils assurent en effet une faible indépendance des programmes aux données. Ils peuvent rester très compétitifs au niveau interne. Les critiques proviennent sans doute aussi de la complexité des langages de manipulation historiquement associés à ces modèles, basés sur la navigation. Cependant, il est tout à fait possible de définir des langages de plus haut niveau fondés sur la logique des prédicats pour un modèle hiérarchique ou réseau.

Finalement, la question s'est posée de savoir si le modèle entité-association est un meilleur candidat pour modéliser les données au niveau conceptuel que le modèle réseau. La réponse est positive. En effet, un modèle se voulant un cadre conceptuel pour définir une large classe de base de données structurées et un médiateur entre des représentations de données stockées et des vues usagers, devrait probablement avoir au moins quatre caractéristiques [Codd79] :

1. permettre une représentation sous forme de tableaux de données ;

2. permettre une interrogation ensembliste afin d'autoriser des requêtes sans préciser les ordres élémentaires de navigation dans la base ;

3. permettre une interprétation des objets en termes de formules de la logique des prédicats afin de faciliter les inférences de connaissances ;

4. supporter une représentation graphique simple afin de pouvoir visualiser les objets et leurs associations pour concevoir la base.

Ajoutons également que la simplicité est une caractéristique essentielle d'un modèle. C'est là un avantage important du modèle relationnel que nous allons étudier dans la partie suivante de cet ouvrage.

6. BIBLIOGRAPHIE

[Bachman64] Bachman C., Williams S., "A General Purpose Programming System for Random Access Memories", *Fall Joint Computer Conference*, Vol. 26, AFIPS Press, pp. 411-422.

La première présentation du système IDS, réalisé à General Electric au début des années 60.

[Bachman69] Bachman C., "Data Structure Diagrams", *Journal of ACM, SIGBDP* Vol. 1, N° 2, mars 1969, pp. 4-10.

*Cet article introduit les diagrammes de Bachman. Ceux-ci permettent de modéliser les types d'articles par des boîtes et les liens (*sets*) par des flèches depuis le propriétaire vers le membre. Ils modélisent une base de données comme un graphe dont les noeuds sont les types d'articles et les arcs les associations de 1 vers n articles. Ce type de modélisation est toujours très utilisé, notamment dans l'outil de conception BACHMAN, du nom de l'inventeur.*

[Bachman73] Bachman C., "The Programmer as Navigator", *Communication of the ACM*, Vol. 16, N° 1, novembre 1971.

Cet article correspond à la présentation de Charlie Bachman lorsqu'il a reçu le Turing Award en 1973. Bachman compare le programmeur à un navigateur qui suit les chemins d'accès aux bases de données à l'aide de curseurs, afin d'atteindre les articles désirés.

[Cardenas85] Cardenas A., *Data Base Management Systems*, 2nd Edition, Allyn and Bacon, Boston, Ma, 1985.

Ce livre présente plus particulièrement les techniques de base des modèles réseau et hiérarchique. Il décrit de nombreux systèmes dont TOTAL, IDS II, SYSTEM 2000 et IMS.

[Codasyl71] Codasyl Database Task Group, *DBTG April 71 Report*, ACM Ed., New York, 1971.

Les spécifications de référence des langages de description et de manipulation du CODASYL. La présentation effectuée dans ce chapitre est très proche de ces propositions.

[Codasyl78] Codasyl Database Task Group, *Codasyl Data Description Language Journal of Development*, Codasyl Ed., New York, 1978.

La nouvelle version du langage de définition de données du CODASYL. Cette version est probablement moins implémentée que la précédente.

[Codasyl81] Codasyl Database Task Group, *Codasyl Data Description Language Journal of Development — Draft Report*, Codasyl Ed., New-York, 1981.

Encore une nouvelle version du langage de définition et de manipulation de données du CODASYL. Cette version n'a jamais été adoptée.

[IBM78] IBM Corporation, "Information Management System/ Virtual Storage General Information", *IBM Form Number GH20-1260*, SH20-9025-27.

La description générale du système IMS II d'IBM.

[MRI74] MRI Systems Corporation, *System 2000 Reference Manual*, Document UMN-1, 1974.

Le manuel de référence de la première version de System 2000.

[Taylor76] Taylor R.W., Frank R.L., "Codasyl Data Base Management Systems", *ACM Computing Surveys*, Vol. 8, N° 1, mars 1976.

Un panorama très complet résumant les caractéristiques et les évolutions du modèle réseau.

[Tsichritzis76] Tsichritzis D.C., Lochovsky F.H., "Hierarchical Database Management — A Survey", *ACM Computing Surveys*, Vol. 8, N° 1, mars 1976.

Un panorama très complet résumant les caractéristiques et les évolutions du modèle hiérarchique.