SYSTEME PORTABLE DE MANIPULATION DE BASES DE DONNEES HETEROGENES

Yves DELVAUX - Jean-Luc HAINAUT

Institut d'Informatique
Facultés Universitaires de Namur
rue Grandgagnage, 21
B-5000 NAMUR
BELGIQUE

RESUME.
L'article présente les principes et l'architecture d'une interface d'exploitation de bases de données existantes gérées par des Systèmes de Gestion de Bases de Données (SGBD) hétérogènes. L'interface repose sur l'existence d'un modèle de données commun aux SGBD. Un langage de manipulation de données permet tant au programmeur qu'à l'utilisateur occasionnel de travailler de manière interactive sur une ou plusieurs bases de données. Le système réalisant l'interface est indépendant des SGBD et de la machine matérielle et logicielle sur laquelle il fonctionne. Il est en outre aisément extensible.

MOTS-CLES.
Interface interactive, coexistence de SGBD hétérogènes, systèmes portables, modèle d'accès, langage de manipulation de données.

1 INTRODUCTION

Le projet IDML (Interactive Data Manipulation Language) a pour objectif la conception et la réalisation d'un système interactif d'exploitation de bases de données. Ainsi posé, le problème n'offrirait guère d'originalité s'il n'était assorti des contraintes suivantes:

- L'accès aux bases de données doit être offert tant à l'usager occasionnel qu'au programmeur d'application.
- Le système doit prendre en charge toute base de données existante et non pas exiger des bases de données qui lui soient adaptées.
- Il doit permettre la coexistence de bases de données hétérogènes, c'est-à-dire gérées par des SGBD (Systèmes de Gestion de Bases de Données) ou SGF (Systèmes de Gestion de Fichier) différents.
- Le système doit être transportable vis-à-vis des machines et des SGBD.
- Il doit enfin être extensible.

La première tâche de conception a été la mise au point d'un modèle général des données qui offre à l'utilisateur une description des bases de données qui soit simple, indépendante des LDD (Langages de Description de Données), et suffisamment précise pour permettre une exploitation efficace de ces bases de données.

La deuxième tâche fut la définition du LMD (Langage de Manipulation de Données) qui permet l'accès aux données vues au travers du modèle général. Ce langage devrait offrir des possibilités d'accès sous une forme aussi simple et naturelle que possible, tout en offrant des mécanismes de programmation puissants.

Enfin, l'architecture du système IDML a été élaborée de manière à mettre en oeuvre modèle et langage de manipulation tout en respectant les objectifs d'indépendance, de simplicité d'utilisation et d'extensibilité.

L'article est organisé en trois parties correspondant à chacune de ces phases.

2 LE MODELE DES DONNEES

L'étude des structures de données offertes par les SGBD/SGF montre rapidement qu'au travers des apparences les plus dissemblables, ceux-ci ne mettent en oeuvre qu'un nombre très limité de concepts, et qu'ils se distinguent essentiellement l'un de l'autre par les assemblages de ces concepts qu'ils permettent. Nous établirons un relevé de ces concepts tel que l'on puisse décrire les types de structures de données d'un SGBD par un jeu de contraintes d'assemblage.

Préalablement à la construction d'un tel modèle, il convient de préciser le niveau de celui-ci. Il serait tentant d'offrir d'une base de données opérationnelle une description conceptuelle qui pourrait s'exprimer dans l'un des modèles actuellement répandus : modèle(s) de CODD, modèles Entité/Association, modèles binaires, etc... Cependant, nous nous exposerions alors à d'énormes difficultés de traduction des expressions de manipulation de données liées à ce modèle en séquences d'appels au SGBD. Ces difficultés sont de deux ordres. Le premier concerne l'efficacité de la traduction, et l'on sait que ce problème n'est pas encore réellement résolu. Le second tient à l'objectif même du projet : prendre en charge des bases de données opérationnelles; la conceptualisation d'une telle base masquerait un nombre considérable de détails techniques dont la connaissance est cependant indispensable pour manipuler les données,

particulièrement en ce qui concerne la mise à jour. Atteindre ce niveau est illusoire à l'heure actuelle. Il nous faut donc composer et viser un niveau de perception qui pour être suffisamment épuré n'en est pas moins proche des concepts perçus par le SGBD.

Ce niveau a été mis en évidence depuis plusieurs années dans le cadre d'une démarche d'analyse [14],[3],[13]; notre objectif étant quelque peu différent (description d'une base de donnée existante), nous ferons appel à un modèle spécifique : le modèle des accès logiques [6] dont nous décrirons sommairement les concepts essentiels. Ceux-ci sont organisés en trois classes : les objets, les mécanismes d'accès et les primitives.

2.1 Les objets

L'élément de base est l'article, unité d'information enregistrée qui peut faire l'objet d'une demande d'accès, de création et de suppression. Tous les articles sont distincts et chacun appartient à un type d'article qui en définit les propriétés générales.
La valeur d'item est une donnée manipulable dans un programme et qui appartient à un type, appelé Item. Un item est décomposable en d'autres items si ses valeurs sont constituées de fragments significatifs. A un article peuvent être associées des valeurs d'item. Les propriétés de l'association entre un type d'article et un item font de ce dernier un item obligatoire ou facultatif, simple ou répétitif, identifiant ou non.
Un fichier est une collection dynamique d'articles; un article appartient toujours à un fichier.
Une base de données est la collection des articles d'un ensemble de fichiers; elle contient toujours un article particulier (article SYSTEME) qui est son point d'entrée et portera le nom de cette base de données, ou plus exactement de son schéma externe. Certains SGBD donnent effectivement une existence matérielle à cet article (IDS, DMS 11, CODASYL).
Afin de simplifier la manipulation des fichiers pour l'utilisateur non spécialiste (cette notion ne peut pas lui être cachée) on associera à tout type d'article un item (fictif) de nom FICHIER dont la valeur est le nom du fichier de l'article. Ceci rend très homogène l'accès aux articles tant en sélection qu'en mise à jour.

2.2 Les mécanismes d'accès

L'accès est pour un programme la mise à disposition d'un objet de la base de données. La possibilité de cette opération est représentée par l'existence d'un mécanisme d'accès parmi les suivants :
- l'accès à une base de données, représenté par l'accès à son article SYSTEME,
- l'accès à un fichier qui peut être simulé grâce à l'item FICHIER,
- l'accès aux articles d'une base de données, d'un fichier,
- l'accès aux valeurs d'items d'un article,
- l'accès aux articles auxquels est associée une valeur déterminée d'un item ou groupe d'items (qui est appelé clé d'accès de ce type d'article),
- l'accès à des articles à partir d'un article; ce mécanisme appelé chemin d'accès mérite une définition plus précise.

Un chemin d'accès est un mécanisme qui associe à un article (origine) 0, 1 ou plusieurs articles (cibles) d'une manière telle qu'il soit possible d'accéder à partir de l'article origine, successivement aux articles cibles. Tout chemin appartient à un type défini essentiellement par :
- les types possibles de l'article origine,
- les types possibles des articles cibles,
- sa connectivité (N-N, 1-N, N-1, 1-1, de l'origine vers la cible),

- l'existence d'un type de chemin inverse,
- l'ordre éventuel d'accès aux cibles.

Ce modèle peut aisément être formalisé par un modèle relationnel binaire. Voir à ce sujet les références [3] et [10].

2.3 Les primitives

Elles correspondent aux opérations élémentaires qu'il est possible d'effectuer sur les objets d'une base de données. Elles peuvent être rangées en trois classes : les primitives d'accès, qui mettent en oeuvre les mécanismes d'accès, les primitives de modification qui font évoluer l'état de la base de données, et les primitives de contrôle d'environnement qui fonctionnellement permettent la création de super-primitives, l'établissement de points de reprise et la maîtrise de la concurrence (notion de transaction bd dans certains SGBD).

2.4 Les modèles partiels

Le modèle complet d'une base de données peut être relativement complexe. Cependant des modèles partiels très simples peuvent en être déduits. Tel serait le cas d'une utilisation en consultation pure où de nombreux concepts pourront être ignorés.

2.5 Représentativité du modèle

Le modèle a été utilisé pour décrire les principaux SGBD/SGF existants (e.a, CODASYL, IMS, ADABAS, SOCRATE, COBOL). Les concepts de ces SGBD qui n'ont pu être pris en charge sont extrêmement réduits (voir notamment[8]).

3 LE LANGAGE IDML

Le langage de manipulation des données s'est vu assigner deux objectifs en apparence contradictoires : la simplicité d'usage et d'expression, et d'autre part la puissance et l'efficacité.

Nous avons cherché la simplicité dans plusieurs directions :
- les données de toute base de données sont perçues selon les concepts du modèle d'accès, complet ou partiel et non ceux du SGBD,
- les mécanismes de programmation sont réduits mais extrêmement généraux (par ex. la boucle FOR-END),
- la sélection d'articles s'exprime sous des formes concises, intuitives et puissantes.
- certains raccourcis d'écriture conduisent à des expressions non algorithmiques dans de nombreux cas.

Quant à la puissance et à l'efficacité, elles sont offertes au programmeur au travers notamment :
- d'un langage de type algorithmique doté de mécanismes tels que la procédure (récursive), les manipulations de valeurs numériques et de caractères, de conversion de types, etc...
- de l'exploitation (facultative) de la connaissance de certaines mécanismes d'accès (par exemple clé d'accès),
- de la manipulation aisée de collections temporaires d'articles.

Dans le cadre limité de cet article, nous n'aborderons que les principes des variables IDML, de la boucle IDML et des modes d'exécution d'un programme.

3.1 Les variables IDML

Une procédure IDML, même réduite à une instruction usera généralement de variables, déclarées soit explicitement (comme en Pascal ou en Cobol) soit implicitement (comme en Basic ou Fortran), ce qui simplifie leur usage par un non spécialiste.
Outre les types standard, qui correspondent aux items (numériques, caractères, décomposables), IDML offre un type particulier, la variable d'article, dont la valeur est une référence à un article d'une base de données.

Une variable peut être définie comme étant simple (une valeur), en tableau (nombre fixe de valeurs) ou en liste (nombre variable de valeurs). En particulier, une variable d'article en liste constitue une représentation élégante de "collections temporaires d'articles" de taille quelconque.
La désignation d'une valeur d'item (ou collection de) attachée à un article utilisera la notation pointée : soient EMP1 (simple) et EMPN (liste) deux variables d'article;

EMP1.NOM désigne la valeur de NOM associée à l'article référencé par EMP1,

EMPN.NOM désigne la collection des valeurs de NOM associées aux articles référencés par EMPN.

3.2 La boucle IDML

La boucle "for" est une extension aux objets externes au programme de la boucle énumérative des langages de programmation classiques. Elle constitue l'unique moyen d'accès aux bases de données. Sa forme et son interprétation sont les suivantes :

for VC := ‹DEO› ‹sequence› end	pour chaque objet de l'ensemble décrit par ‹DEO›, exécuter les instructions de ‹sequence›; la variable VC désigne l'objet courant durant cette exécution.

Lorsque ‹DEO› désigne un ensemble de valeurs numériques, caractères ou décomposables, la boucle "for" est semblable à celle des langages classiques . ‹DEO› peut cependant désigner des articles d'une base de données.
L'expression générale de ‹DEO› est alors la suivante :

‹AC› ‹COND› from ‹AO› via ‹CHA›

et s'interprète comme suit :
"les articles désignés par ‹AC› (nom de type d'article ou variable d'article) qui vérifient la condition ‹COND› et qui sont cibles du chemin ‹CHA› dont l'origine est l'article désigné par la variable ‹AO›".

Soit DEP une variable désignant un article DEPARTEMENT (Voir schéma en annexe). La boucle suivante a pour effet d'afficher au terminal le NOM et le SALAIRE des EMPLOYES gagnant plus de 5.000 F et occupés dans ce département:

```
for E := EMPLOYE (: SALAIRE > 5000) from DEP via OCCUPE
        display E.NOM , E.SALAIRE
end
```

Une forme equivalente de cette désignation serait d'exprimer le parcours du chemin comme une condition de sélection :

```
for E := EMPLOYE ((: SALAIRE > 5000) & (OCCUPE : DEPARTEMENT (= DEP)))
```

Cet exemple est intéressant car il utilise les trois formes fondamentales de la condition de sélection d'articles :

- condition sur les valeurs d'item associées : (: SALAIRE > 5.000)
- condition d'association à d'autres articles par chemin : (OCCUPE : DEPARTEMENT <C>) où <C> est une condition de sélection d'articles DEPARTEMENT
- la condition d'appartenance à un ensemble : (=D) qui est mise pour (∈ {D})

Précisons sans développer que les valeurs d'item font l'objet de conditions obéissant à ces trois formes; par ex. > 5000 est mis pour (∈ {5001..infini})

L'accès séquentiel aux articles d'un type se ramène au cas précédent grâce à la convention suivante : tout type d'article accessible séquentiellement est cible d'un type de chemin implicite (sans nom) dont l'origine est l'article système. D'autre part l'accès à une base de données ("ouverture" de cette base) est représenté par un accès à son article système. L'accès séquentiel aux articles EMPLOYE s'écrira donc simplement :

```
for ROOT := BDPERS
        ...
        for E := EMPLOYE from ROOT
                ...
        end
        ...
end.
```

Seuls les types de chemin implicites étant dénués de nom, la clause from ROOT peut ici être omise.

L'accès par clé d'accès est tout aussi simple : la boucle suivante, vide, range dans E la référence de l'article EMPLOYE dont NUME = NUMX.

```
for E := EMPLOYE (: NUME = NUMX)
end
```

La boucle d'accès permet donc d'exprimer tous les types d'accès nécessaires : (séquentiel, séquentiel filtré, par clé d'accès, par chemin d'accès, par chemin d'accès filtré) sous une forme unique et particulièrement naturelle.
N.B. On trouvera semblables mécanismes dans des systèmes comme SOCRATE [1], DATABASIC [2] et SPHINX [5].

3.3 Mode d'exécution d'un programme

Le mode standard d'utilisation d'IDML consiste à rédiger (par un éditeur de texte) un programme puis d'en demander l'exécution, à l'image des systèmes BASIC interactifs. Cependant, toujours à l'image des BASIC, il est possible d'effectuer un travail pas à pas sur une base de données. La convention est la suivante : toute séquence d'instructions terminée par "!" est exécutée et non traitée par l'éditeur; en outre toute boucle

non clôturée l'est automatiquement sauf pour l'article système. Exemple :

```
* for R := BDPERS !                          - Entrée dans la base BDPERS
* for E := EMPLOYE (:NOM = "Mercier") -  Les rèfèr. aux articles EMPLOYE
                    $LISTE :=+ E !            de NOM "Mercier" sont accumulèes
                                              dans la variable en liste $LISTE.
* display size ($LISTE) !                    - taille de la liste ?
  14
* for E:= $LISTE                             - imprimer les valeurs de NUME
       display E.NUME !                       de ces articles
  114
  720
    . . .
```

Ce mode permet une exploitation non prédéfinie qui exigerait en programmation classique un algorithme généralement complexe.

4 LE SYSTEME IDML

L'objectif du système IDML est le traitement des programmes IDML à leurs différents stades. Il doit par conséquent assurer le dialogue avec l'utilisateur, gérer l'introduction, la correction et la conservation des programmes, assurer et contrôler l'exécution et la mise au point de ceux-ci, et donc, gérer les communications avec les SGBD/SGF. Il doit également réaliser la traduction des schémas des SGBD en schémas simplifiés selon le modèle d'accès.
Le système devait dès le départ être doté de caractéristiques d'indépendance ou de portabilité vis-à-vis de la machine et des SGBD, d'extensibilité, et enfin de généralité et de simplicité de concepts.

Après une présentation sommaire de l'architecture du système, nous l'évaluerons vis-à-vis de ces caractéristiques.

4.1 Architecture du système IDML.

L'architecture repose sur les principes suivants :

- toute communication avec l'extérieur est perçue comme une transaction avec une base de données au sens du modèle d'accès. Tel est évidemment le cas des bases de données authentiques, mais aussi celui du terminal, des fichiers de travail, des sauvegardes des programmes sources et objets et de la base des schémas. Ces communications sont gérées par des interfaces d'accès dont les points d'entrée correspondent aux primitives du modèle d'accès.
- Toute opération de manipulation de données qui dépend de la machine et du système d'exploitation est réalisée par un module spécialisé. Outre les interfaces d'accès déjà mentionnées, on trouvera dans le système un module arithmétique, un module de manipulation de chaînes de caractère, etc...
- L'interprétation d'un programme IDML conduirait à des performances dégradées; par contre, il n'est pas question de réaliser une compilation en un langage classique en raison du caractère interactif du système. L'exécution est par conséquent décomposée en deux phases : une traduction dans un code intermédiaire, puis une interprétation de celui-ci.

Ajoutons enfin que le système lui-même, à l'exception de certains modules spécialisés a été écrit en FORTRAN minimum ou plus exactement en RATFOR [11] (FORTRAN structuré à précompiler).

Le système IDML est constitué de deux sous-systèmes : le système de traitement des programmes et celui de traitement des schémas.

4.2 Le sous-système de traitement des programmes

Les composants fondamentaux sont : un éditeur de texte, un compilateur, un relieur, et un interprète. Ils sont décrits dans le schéma reporté en annexe.

L'éditeur, inspiré du BASIC, permet de créer des procédures IDML, de les corriger, de les stocker en un fichier.

Le compilateur traduit une procédure IDML en une procédure en code interne (qui peut être stockée en un fichier). Cette traduction nécessite la connaissance du schéma des accès logiques des bases exploitées; celui-ci est accessible dans la base de données des schémas IDML. Dans le code interne, les variables sont référencées via des descripteurs qui précisent type, longueur, structure de décomposition et adresse absolue des valeurs. Ajouté à l'adressage relatif des branchements, ceci rend le code interne aisément relogeable.

Le relieur assemble, si nécessaire, les différentes procédures pour constituer un programme en code interne exécutable.

L'interprète, ou machine virtuelle IDML, exécute le programme en code interne. Il fait appel à des processeurs spécialisés tels que le module arithmétique, le module de manipulation de chaînes ainsi que le processeur de bases de données qui regroupe les différentes interfaces d'accès aux bases de données, au terminal, aux divers types de fichiers

4.3 Le sous-système de traitement de schémas

La fonction de ce système est la production de deux composants du sous-système précédent, à savoir la base de données des schémas IDML et les interfaces d'accès. Ces deux composants ont en fait une portée beaucoup plus générale que celle du système IDML. En effet, les interfaces offrent au programmeur d'application COBOL (ou autres) un très haut niveau d'indépendance par rapport au SGBD et SGF, tandis que la base de données des schémas pourra servir à la production de documents à destination de ces programmeurs ou même à la génération automatique de programmes d'exploitation de bases de données (cfr. vol. V de [5]).
Ces deux composants sont produits par un programme qui, à partir du texte rédigé dans le LDD du SGBD/SGF, range dans la base de données des schémas la description du modèle d'accès sous-jacent à ce texte, puis construit l'interface d'accès à la base de données; cette construction, très systématique, consiste à associer à chaque primitive la séquence d'instructions COBOL + LMD du SGBD qui la réalise.

Ce programme est propre à un SGBD/SGF. Il est constitué d'un analyseur, d'un chargeur (BD de schémas) et d'un générateur (d'interfaces).

4.4 Respect des objectifs

L'indépendance du système IDML est assurée par l'usage d'un noyau du FORTRAN IV qui est le langage le plus répandu, le plus figé et techniquement le plus efficace (voir à ce sujet l'ouvrage [12]), par la généralisation des interfaces d'accès, dont la génération est automatisée, et par les modules spécialisés en nombre réduit.

L'extensibilité est favorisée d'une part par l'existence de la machine virtuelle IDML qui n'est en rien liée au langage IDML; d'autres compilateurs pourraient ainsi etre construits. D'autre part, la

décomposition de cette machine en processeurs spécialisés permet une extension ou une modification fonctionnelle de celle-ci.

Quant à la généralité, elle est liée essentiellement à la notion d'interface d'accès et à la structure du code intermédiaire, qui dépasse le cadre de cet article.

5 ETAT DU PROJET, CONCLUSIONS ET PERSPECTIVES

Nous disposons à l'heure actuelle d'un noyau du système IDML constitué de la machine virtuelle IDML, du compilateur et de l'éditeur. Le langage IDML accepté par le compilateur n'inclut pas encore l'appel de procédure, , le traitement des variables en liste et les ordres de mise-à-jour de bases de données. Cependant, même réduit, le système offre une facilité incomparable de consultation. En ce qui concerne le traitement de schémas, la prise en charge de bases de données COBOL (c'est-à-dire ensemble de fichiers gérés en commun) est en voie d'achèvement. La génération d'interfaces CODASYL devrait être disponible dans le courant de 1982; pour l'instant, ces interfaces sont écrites à la main.

La principale conclusion serait à notre avis qu'une tentative de faire coexister véritablement des bases de données hétérogènes existantes et de les mettre à la disposition de tous les utilisateurs, spécialistes ou non, semble avoir réussi. Par ailleurs, le système réalisé jouit de propriétés dont la plus importante est sa grande indépendance par rapport à son environnement.

Il convient cependant de mettre en évidence deux aspects du système :
- à la lourdeur du SGBD s'ajoute celle du système IDML lui-même; le volume de ses programmes n'est pas négligeable pas plus que sa (relative) consommation CPU;
- si l'intégration sans restriction de bases de données semble réaliste pour des systèmes du type réseau (CODASYL, DMS II, TOTAL, ADABAS) ou pseudo-relationnel orienté accès (fichiers COBOL, ADABAS, SESAM), les systèmes du type arborescent (IMS) posent des problèmes non triviaux, en raison des restrictions qu'ils imposent à une exploitation navigationnelle.

Le noyau actuel doit encore être étendu par la réalisation des fonctions prévues et non encore disponibles. Cependant, il constitue déjà le point de départ de nouvelles recherches. L'une de celles-ci consiste à étudier concrètement l'extension aux bases de données réparties; à partir de fichiers distribués dans un réseau de micro-ordinateurs, on étudiera différentes solutions de transmission : programme IDML, programme en code interne, primitive du modèle d'accès.

6 REFERENCES

[1] ABRIAL, BAS, BEAUME, HENNERON, MORIN, VIGLIANO, "Projet Socrate", IMAG, Grenoble, 1970.

[2] BACHMAN, BREWER, "A Direct Access System", Honeywell Computer Jour. vol. 8, n 1, 1974.

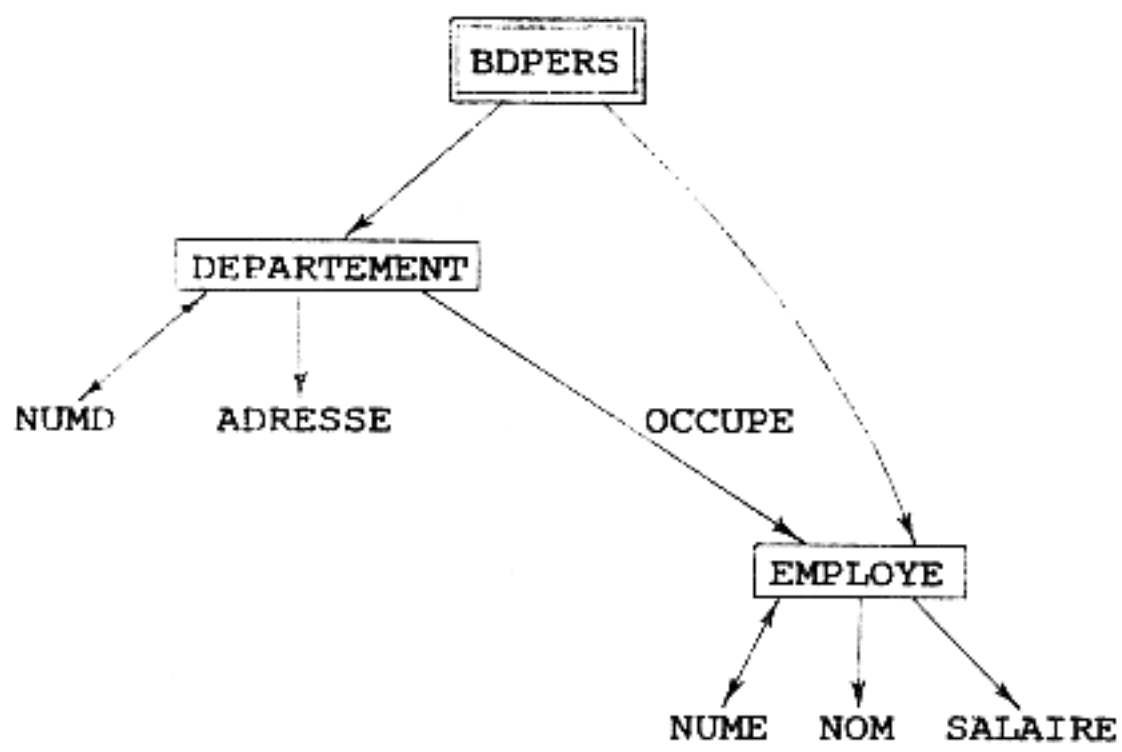
[3] CABANES, "Rapport Indépendance dans les SGBD", Institut d'Informatique d'Entreprise, CNAM, Paris [Mars 1977].

[4] DELVAUX, HAINAUT, Projet IDML, Documents internes, Institut d'Informatique Namur.

[5] Equipe Bases de Données, "Systèmes de conception et d'exploitation de

bases de données", volumes I à V, Rapport final du projet CIPS I.2/15, 1978, Institut d'Informatique de Namur.
[6] HAINAUT, "Un modèle de description de fichiers : le modèle des accès logiques", Institut d'Informatique, Namur 1980.
[7] HAINAUT, "Analyse des accès proposés par le DBTG CODASYL 1971", Institut d'Informatique, Namur, 1980.
[8] HAINAUT, Cours de Bases de Données, deuxième partie. Institut d'Informatique de Namur.
[9] HAINAUT, "IDML : Système d'interaction avec des bases de données CODASYL", 8eme Ecole d'Eté de l'AFCET, Namur, juillet 1978.
[10] HAINAUT, "Theoretical and practical tools for data base design", Proc. 7th conf. on VLDB, 1981 - à paraître [IEEE]
[11] KERNIGHAN, "RATFOR - A preprocessor for a Rational FORTRAN" in Software Practice and Experience, vol. 5, 1975.
[12] KERNIGHAN, PLAUGER, "Software tools", Addison-Wesley Publish, 1976.
[13] TARDIEU, NANCI, PASCOT, "Conception d'un système d'information", les Editions d'Organisation/Gaëtan Morin, 1979.
[14] --- ,"Data structure models for information systems", proc. Internat. Workshop of Namur, 1974, Presses Universitaires de Namur.

ANNEXE 1 : Exemple de modèle d'accès.

Soit une base de données de nom BDPERS contenant deux types d'article, DEPARTEMENT (auxquel sont associés les items NUMD et ADRESSE) et EMPLOYE (auquel sont associés les items NUME, NOM, SALAIRE). Il existe un type de chemin d'accès 1-N de DEPARTEMENT vers EMPLOYE; d'autre part, on admet l'existence, à partir de l'article Système (appelé BDPERS), d'un chemin implicite vers DEPARTEMENT et d'un autre vers EMPLOYE ; leur type n'a pas de nom. NUMD et NUME sont des clés d'accès pour leur type d'article.

Une représentation graphique intuitive serait la suivante :