La Micro mémoire relationnelle (MIMER) : un outil pour la construction de SGBD relationnels, projet MICROBE

Fernando Fernandez

THESE

présentée à

l' Université Scientifique et Medicale de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR

par

Fernando FERNANDEZ

LA *MI*CRO *ME*MOIRE *R*ELATIONNELLE *(MIMER)* : UN OUTIL POUR LA CONSTRUCTION DE SGBD RELATIONNELS. PROJET MICROBE.

Thèse soutenue le 12 novembre 1981 devant la Commission d'Examen :

Monsieur	C. DELOBEL	: Président
Messieurs	M. ADIBA F. BANCILHON J.-C. CHUPIN	Examinateurs

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Monsieur Gabriel CAU : Président
Monsieur Joseph KLEIN : Vice-Président

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	AMBLARD Pierre	Clinique de dermatologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	I.S.N.
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale A
	BEAUDOING André	Clinique de pédiatrie et puériculture
	BELORIZKY Elie	Physique
	BARNARD Alain	Mathématiques pures
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques pures
	BEZES Henri	Clinique chirurgicale et traumatologie
	BLAMBERT Maurice	Mathématiques pures
	BOLLIET Louis	Informatique (I.U.T. B)
	BONNET Jean-Louis	Clinique ophtalmologie
	BONNET-EYMARD Joseph	Clinique hépato-gastro-entérologie
Mme	BONNIER Marie-Jeanne	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BOUTET DE MONVEL Louis	Mathématiques pures
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologique
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmacodynamie

.../...

MM.	LLIBOUTRY Louis	Géophysique
	LOISEAUX Jean-Marie	Sciences nucléaires
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mlle	LUTZ Elisabeth	Mathématiques pures
MM.	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Clinique cardiologique
	MAYNARD Roger	Physique du solide
	MAZARE Yves	Clinique Médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEGRE Robert	Mécanique
	NOZIERES Philippe	Spectrométrie physique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET Jean	Séméiologie médicale (neurologie)
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REVOL Michel	Urologie
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-Chirurgie
	SARRAZIN Roger	Clinique chirurgicale B
	SEIGNEURIN Raymond	Microbiologie et hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique (I.U.T. I)
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
Mme	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique biophysique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale

.../...

PROFESSEURS ASSOCIES

MM.	CRABBE Pierre	CERMO
	SUNIER Jules	Physique

PROFESSEURS SANS CHAIRE

Mlle	AGNIUS-DELORS Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Gilbert	Géographie
	BENZAKEN Claude	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique (I.U.T. I)
	BUISSON René	Physique (I.U.T. I)
	BUTEL Jean	Orthopédie
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique (I.U.T. I)
	DELOBEL Claude	M.I.A.G.
	DEPASSEL Roger	Mécanique des fluides
	GAUTRON René	Chimie
	GIDON Paul	Géologie et minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biochimie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et médecine préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KRAKOWIACK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique (I.U.T. I)
	LUU DUC Cuong	Chimie organique - pharmacie
	MICHOULIER Jean	Physique (I.U.T. I)
Mme	MINIER Colette	Physique (I.U.T. I)

.../...

MM.	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	M.I.A.G.
	REBECQ Jacques	Biologie (CUS)
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	STIEGLITZ Paul	Anesthésiologie
	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	ARMAND Yves	Chimie (I.U.T. I)
	BACHELOT Yvan	Endocrinologie
	BARGE Michel	Neuro-chirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamie
MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (I.U.T. B) (Personne étrangère habilitée à être directeur de thèse)
	BERNARD Pierre	Gynécologie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COLIN DE VERDIERE Yves	Mathématiques pures
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie

.../...

MM.	CYROT Michel	Physique du solide
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FAURE Gilbert	Urologie
	GAUTIER Robert	Chirurgie générale
	GIDON Maurice	Géologie
	GROS Yves	Physique (I.U.T. I)
	GUIGNIER Michel	Thérapeutique
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	JALBERT Pierre	Histologie
	JUNIEN-LAVILLAVROY Claude	O.R.L.
	KOLODIE Lucien	Hématologie
	LE NOC Pierre	Bactériologie-virologie
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MALLION Jean-Michel	Médecine du travail
	MARECHAL Jean	Mécanique (I.U.T. I)
	MARTIN-BOUYER Michel	Chimie (CUS)
	MASSOT Christian	Médecine interne
	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique (I.U.T. I)
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (I.U.T. B) (Personnalité étrangère habilitée à être directeur de thèse)
	PEFFEN René	Métallurgie (I.U.T. I)
	PERRIER Guy	Géophysique-glaciologie
	PHELIP Xavier	Rhumatologie
	RACHALL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD Pierre	Pédiatrie
	RAPHAEL Bernard	Stomatologie
Mme	RENAUDET Jacqueline	Bactériologie (pharmacie)
MM.	ROBERT Jean-Bernard	Chimie-physique
	ROMIER Guy	Mathématiques (I.U.T. B) (Personnalité étrangère habilitée à être directeur de thèse)
	SAKAROVITCH Michel	Mathématiques appliquées

.../...

MM.	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques pures
	CHARACHON Robert	Clinique ot-rhino-laryngologique
	CHATEAU Robert	Clinique de neurologie
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	COUDERC Pierre	Anatomie pathologique
	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumophtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DODU Jacques	Mécanique appliquée (I.U.T. I)
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	FONTAINE Jean-Marc	Mathématiques pures
	GAGNAIRE Didier	Chimie physique
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique générale
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
Mme	LAJZEROWICZ Janine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LE ROY Philippe	Mécanique (.I.U.T. I)

.../...

MM.	SCHAERER René	Cancérologie
Mme	SEIGLE-MURANDI Françoise	Crytogamie
MM.	STOEBNER Pierre	Anatomie pathologie
	STUTZ Pierre	Mécanique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	DEVINE Roderick	Spectro Physique
	KANEKO Akira	Mathématiques pures
	JOHNSON Thomas	Mathématiques appliquées
	RAY Tuhina	Physique

MAITRE DE CONFERENCES DELEGUE

M.	ROCHAT Jacques	Hygiène et hydrologie (pharmacie)

Fait à Saint Martin d'Hères, novembre 1977

Je tiens à remercier :

Mr. Claude DELOREL Directeur du laboratoire IMAG pour la manière dont il m'a aidé et conseillé. Qu'il me soit permis de lui témoigner ici toute ma reconnaissance pour m'avoir accueilli dans son équipe et me faire l'honneur de présider ce jury.

Mr. Michel ADIBA professeur à l'université de Grenoble, qui a dirigé cette thèse et m'a fourni de nombreuses suggestions et critiques, en apportant ainsi une aide inapreciable.

Mr. Jean Claude CHUPIN responsable de la division d'architecture de systèmes de la compagnie CII-HB et Mr. François BANCILHON qui ont bien voulu faire partie de ce jury.

Mrs. Gia Toan NGUYEN et Guy SERGEANT responsables du projet MICROBE, qui ont eu, eux aussi, la patience de lire et de critiquer les manuscrits de cette thèse.

Un grand Merci à tous mes collègues et amis de l'équipe MICROBE pour leur travail et leur amical soutien pendant toute la préparation de cette thèse.

En particulier, la trilogie féminine de MICROBE: Mlle. Fatima AZROU, Mme. Andrée CHAPEL et la déesse MIMER; les deux premières qui ont élargi mes connaissances de la langue française (et de la cuisine) et la dernière qui a prêté son nom à l'outil présenté ici.

Puesto que no solo de ciencia se hace investigación, qué se sientan profundamente implicados en este agradecimiento : mis padres, mis hermanos, mi tía y todas mis amistades.

TABLE DES MATIERES

0 INTRODUCTION

Trois niveaux d'abstraction, dans la représentation d'une Base de Données, sont couramment admis:

i) Le niveau interne où un schéma physique décrit la façon dont les données sont stockées en mémoire secondaire.

ii) Le niveau conceptuel où un schéma conceptuel représente les objets du monde réel au moyen d'un modèle de données. Trois modèles principaux sont apparus successivement: les modèles hiérarchique, réseau et relationnel.

iii) Le niveau externe où les schémas externes donnent sur une Base de Données, une "vue logique" appropriée à chaque application.

Nous avons adopté le modèle relationnel de données <COD70> qui marque depuis 1970 une étape décisive et nouvelle dans la conception et la mise en oeuvre des Systèmes de Gestion de Base de Données (SGBD). En particulier, le modèle relationnel, à la différence des modèles précedents, introduit un formalisme qui permet d'identifier et de traiter mathématiquement les différentes entités d'une Base de Données.

Si D1,D2,...,Dn représentent des ensembles de valeurs (appelés domaines), une relation R est définie comme un sous-ensemble du produit cartésien de ces domaines. R est généralement représentée sous forme d'un tableau à n colonnes, dont chacune est appelée attribut. Chaque ligne

de R, appelée tuple est identifiée de manière unique par une clé qui correspond à la valeur d'un (ou plusieurs) de ses attributs.

Parmi les objectifs (cf §2.2.1) de cette thèse, figure au premier plan la réalisation d'un outil de base pour la mise en oeuvre des systèmes relationnels. Le projet MICROBE visant à l'implantation d'un SGBD relationnel sur micro-ordinateurs, a donné un premier cadre d'application à notre travail et a permis en même temps de valider les objectifs fixés.

Dans MICROBE la gestion des données repose sur un logiciel permettant de définir et de manipuler aisément des relations, des attributs et des tuples. Nous désignons ce logiciel sous le nom de MIcro MEmoire Relationnelle (MIMER).

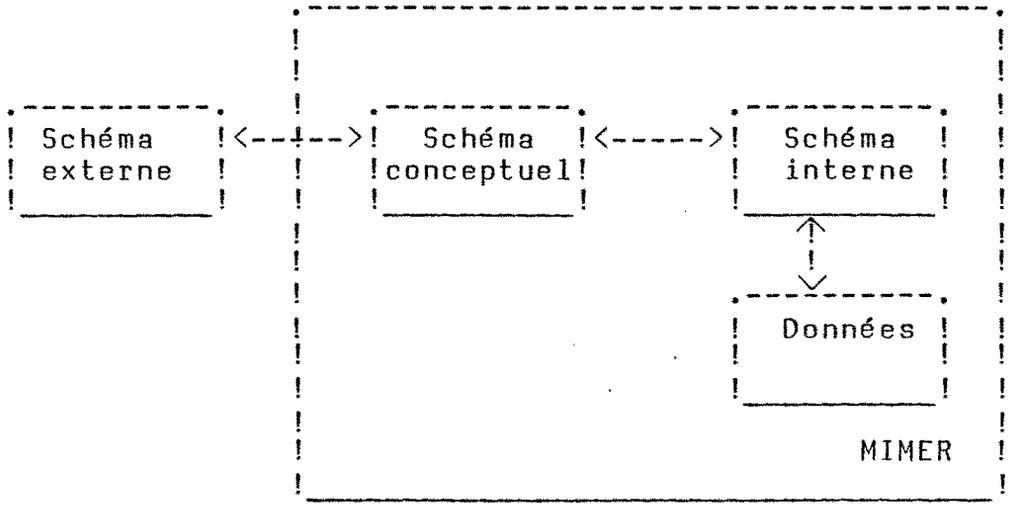


Figure 0.1

Ainsi par rapport aux trois niveaux d'abstraction

présentés ci-dessus, MIMER permet la gestion du schéma conceptuel, du schéma interne et des données de la Base: Figure 0.1.

De façon plus précise, MIMER permet de "voir" des données stockées sous forme de relations et non de fichiers (Figure 0.2). La gestion des relations est garantie par un langage de bas niveau dont les primitives offrent un certain nombre de facilités telles que: créer ou détruire une relation ou un index sur une relation, insérer ou supprimer un tuple, créer ou détruire un attribut sur une relation et balayer ou grouper des relations.

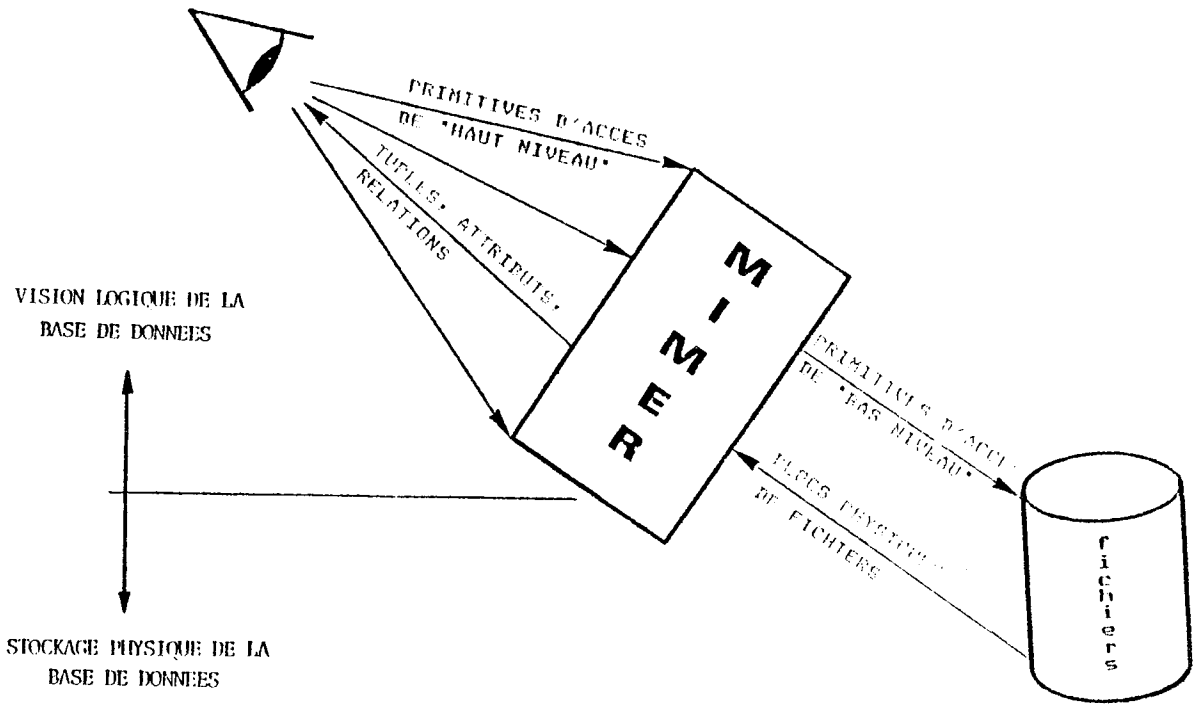


Figure 0.2 Vision relationnelle d'un fichier

Le plan de la thèse est le suivant: une description de

MICROBE est présentée au chapitre 1, tandis qu'au chapitre 2, nous donnons un court aperçu sur l'état de l'art en matière de mémoires relationnelles. Nous y présentons aussi brièvement notre solution. Celle-ci sera largement décrite aux chapitres 3 et 4. Le chapitre 5 est consacré à l'étude des méthodes d'accès aux données. Nous y mettons l'accent sur la méthode des B*-arbres.

Enfin, au chapitre 6 nous tirons des conclusions et proposons quelques axes de recherche qui pourront être poursuivis dans le futur.

1 PRESENTATION DU PROJET MICROBE

1.1 Introduction

1.1.1 Environnement

Un ensemble de bâtiments du campus universitaire grenoblois est relié depuis Janvier 1981 par un réseau local de type DANUBE <NAF79>, <COR81> dont la conception et la réalisation ont été assurées par un projet mené au sein du laboratoire de Mathématiques appliquées de Grenoble (IMAG). Ce réseau est composé d'un ensemble de stations interconnectées par un medium de comunication utilisant la technique de diffusion (type ETHERNET). Une étude du marché du micro-ordinateur a conduit à choisir pour matériel des calculateurs PLESSEY MICRO1 et MICRO2, construits autour d'un microprocesseur de type LSI11.

Le réseau se compose actuellement de six stations MICRO1/MICRO2 comportant chacune 1 Unité Centrale, 64K, 128K ou 256K octets de mémoire centrale, une mémoire secondaire en disque (10 Moctets = 5 Moctets disque fixe + 5 Moctets disque mobile) ou en disquette (500K octets) et des unités d'entrée/sortie. Ces stations sont utilisées sous le système d'exploitation multitâches RSX11M <RSX11>.

Deux autres stations différentes ont été rajoutées au réseau: un mini-ordinateur SOLAR 16-65 et l'ordinateur HB68 du Centre Scientifique de Calcul de Grenoble. Le nombre de

stations MICRO1 /MICRO2 doit rapidement atteindre 8 et on envisage d'ajouter la connection au MINI6 du Centre Scientifique CII-HB.

Toutes les stations connectées sont pourvues d'un logiciel de communication du niveau transport <ISO79> permettant d'assurer des échanges contrôlés d'informations.

1.1.2 Objectifs généraux des travaux

S'appuyant d'une part sur les services du système RSX11M et d'autre part sur l'interface usager fournie par la station de transport <NAF79>, le projet MICROBE consiste en la conception et réalisation d'un Système de Gestion de Bases de Données Répartie sur les stations du réseau local (SGBDR) <FER80>, <FER81b>.

Plusieurs prototypes ont déjà été décrits et implémentés <SUL78>, <TIN78> sans toutefois mettre en oeuvre des algorithmes distribués, tant pour la manipulation des informations réparties que pour la gestion des mécanismes d'exécution répartie et de contrôle de rêquetes simultanées.

Ce projet fait suite aux études menées à Grenoble dans le cadre des systèmes et des Bases de Données Réparties <ADE80>, <DAN77>, <ADI80a>, <ADI80b>.

Des études préliminaires <SER80>, <FER79>, <NGU79>,

<CEL80> ont montré qu'il était possible de tirer parti des contraintes physiques d'environnement réparti, telles que :

-distribution de données,

-processeurs à couplage faibles,

-diffusion de messages,

pour mettre en oeuvre des algorithmes évolués de:

-synchronisation d'exécutions parallèles,

-décomposition de requêtes,

-optimisation de l'exécution répartie,

dans un système distribué où les processeurs ne sont pas liés par un ordre hiérarchique figé, mais ont tous des prérogatives identiques <LAN78>.

Schématiquement le projet MICROBE définit un système dont la structure générale peut se représenter comme suit (Figure 1.1):

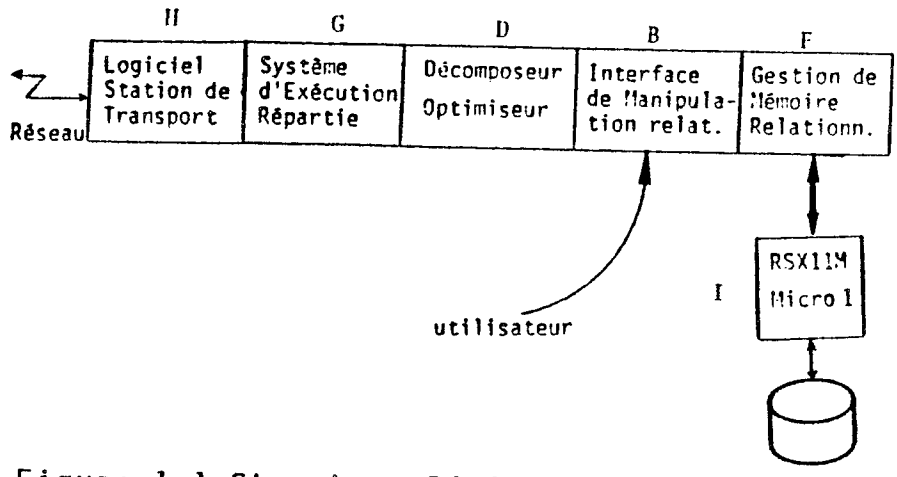


Figure 1.1 Structure Générale de MICROBE

Chaque station MICRO1 locale comporte:

-Une INTERFACE DE MANIPULATION RELATIONNELLE (B) composée d'un traducteur de langage relationnel en une

"arborescence algébrique". Une arborescence algébrique est le support formel de toute requête qu'un usager soumet à la Base de Données. En conséquence, quand ce terme sera utilisé il sera sous-entendu qu'il s'agit d'une requête.

-Un DECOMPOSEUR (D), chargé de localiser les sous-arborescences de façon dynamique, c'est-à-dire en cours d'exécution des requêtes. Pour cela, l'arborescence initiale est diffusée à tous les sites concernés, chargés d'initialiser l'exécution de requêtes partielles. L'obtention des résultats partiels successifs permet d'activer les opérations intermédiaires de chaque requête, reliées aux actions locales d'

-Un SYSTEME D'EXECUTION REPARTIE (G), accédant aux autres composants réseau du SGBDR par l'intermédiaire de la station de transport (H),

-Une MEMOIRE RELATIONNELLE (F) assurant l'interface entre la vision logique relationnelle et le stockage physique des données.

1.1.3 Plan de travaux

Le projet MICROBE a été réalisé en deux phases successives:

ler phase) Etude et mise en oeuvre d'un SGBD centralisé

de type relationnel independant du réseau local,

2ième phase) Etude et mise en oeuvre d'un SGBD réparti sur un réseau local.

Nous allons reprendre dans les paragraphes suivants chacun des composants du système en précisant leur rôle dans l'architecture fonctionnelle décrite ci-après.

1.2 Architecture fonctionnelle

1.2.1 MICROBE centralisé

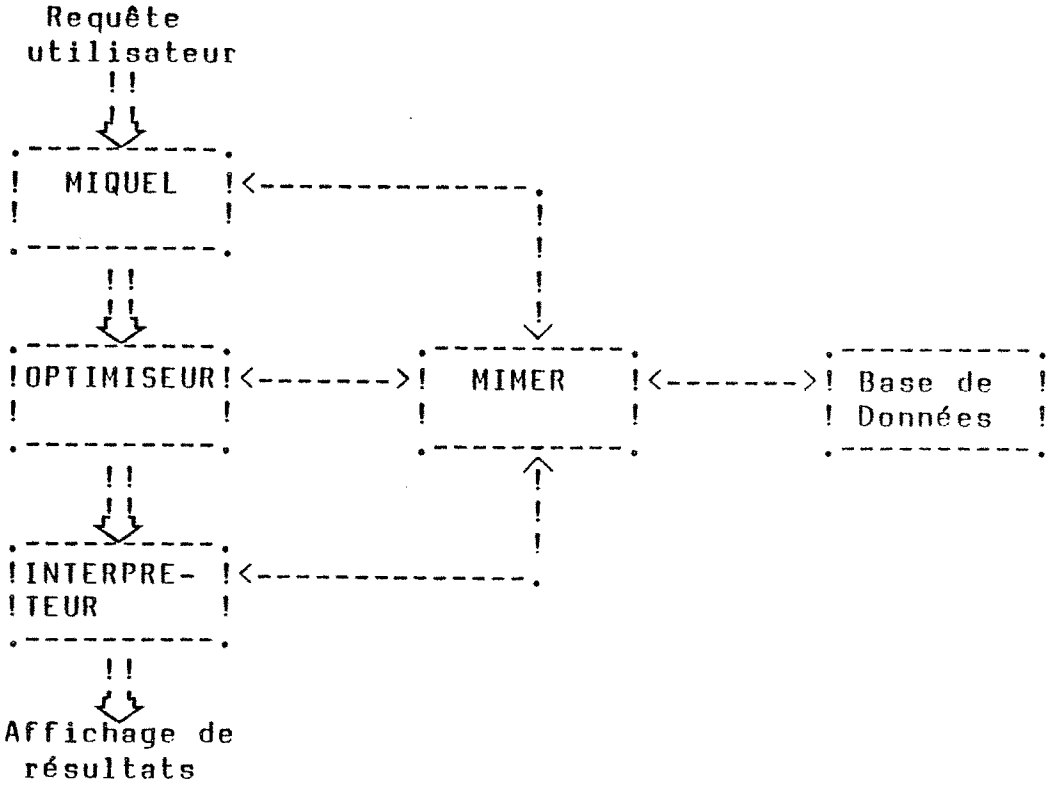


Figure 1.2 Architecture fonctionnelle de MICROBE centralisé

Le SGBD centralisé MICROBE est mis en oeuvre au moyen d'une architecture modulaire (Figure 1.2) présentée en

détail dans <FER80>, <FER81a>, <FER81c>, <FRT81>, <LEE81> et <GAL81>.

Un langage de haut niveau (MIQUEL) <FRT80>, <FRT81> permet de soumettre des requêtes d'interrogation/ modification et de définition de données sur la Base de Données. Nous verrons dans §1.3.1 d'autres manières d'exprimer des requêtes. Une requête MIQUEL est analysée syntaxiquement et sémantiquement, puis transformée en une arborescence binaire d'opérateurs relationnels (Join, Select, etc.). Cette arborescence est alors modifiée à l'aide des règles algébriques qui garantissent une exécution "optimale" de la requête. De plus dans cette étape, un chemin d'accès "optimal" aux données est associé à la requête. Enfin, l'arborescence est interpretée et les résultats sont rendus à l'usager.

Les différents modules de MICROBE accédent à la Base de Données par l'intermédiaire de la mémoire relationnelle MIMER.

Dans l'Annexe 6 nous présentons un exemple de session MICROBE centralisé.

1.2.2 MICROBE réparti

Le Système de Gestion de Bases de Données Réparties est formé d'un ensemble de stations banalisées, connectées au réseau par l'intermédiaire d'une station de transport <NAF79>.

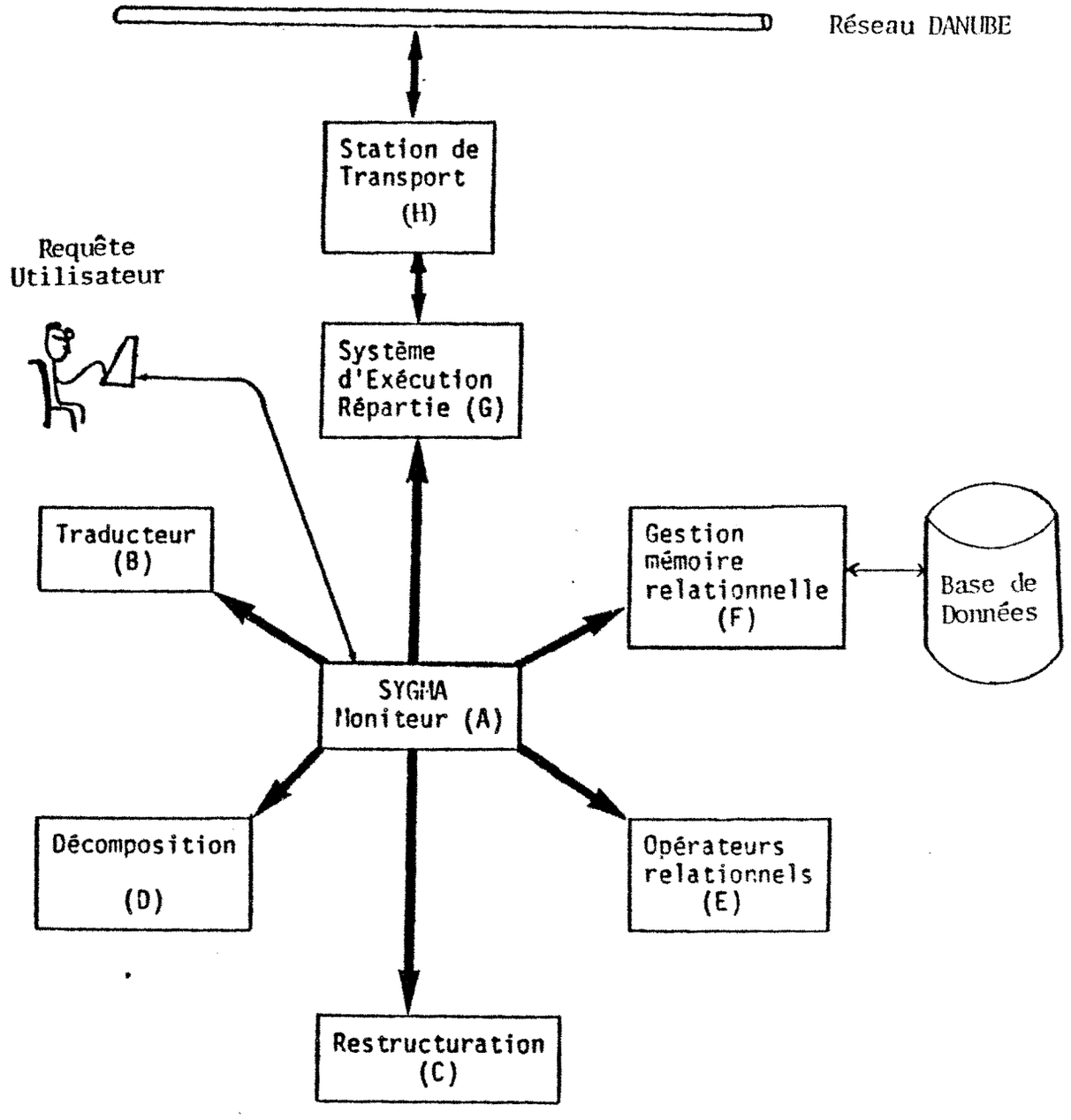


Figure 1.3 Architecture fonctionnelle de MICROBE réparti

Chaque site est bâti autour d'un Système Général de Manipulation d'Arborescences (SYGMA), conformément au schéma ci-dessus (Figure 1.3) et comporte:

-Un moniteur réentrant (A) assurant la synchronisation des échanges de données relationnelles, d'ordres d'exécution et de messages de contrôle entre les modules de chaque serveur local.

-Un traducteur (B) du langage de manipulation de relations en arborescences relationnelles algébriques (cf §1.3) <FRT80>.

-Un module de restructuration (C) de ces arborescences qui les transforme selon des règles prédéfinies.

-Un algorithme (D) de décomposition dynamique et de localisation des noeuds de ces arborescences <NGU79>, chargé de déterminer au cours de l'exécution de la requête les sites d'exécution de chaque opération élémentaire, selon des critères appropriés (cf §1.4).

-Un ensemble de routines (E) de mise en oeuvre des opérateurs relationnels algébriques (SELECT, PROJECT, JOIN, etc.), offrant, pour chacun, une gamme d'algorithmes différents.

-Un module (F) de gestion de la mémoire relationnelle MIMER, offrant des primitives de manipulations de relations de haut niveau.

-Enfin, un système d'exécution répartie (G) (SER) <SER80> est couplé à SYGMA et chargé d'assurer la communication

entre les divers moniteurs locaux, de transmettre et de synchroniser les actions locales correspondant à chaque module actif, etc. (cf §1.5).

Une requête relationnelle soumise par un utilisateur est ainsi transmise au traducteur qui la transforme en une arborescence d'opérateurs algébriques. Cette arborescence est ensuite restructurée selon un ensemble de règles mémorisées dans la Base de Données, puis décomposée en un ensemble d'arborescences partielles. Celles qui sont entièrement localisées sont transmises aux sites appropiés pour être exécutées. Cette exécution fait appel aux algorithmes qui implémentent les divers opérateurs et donne lieu à la production de résultats partiels.

L'initialisation et la terminaison de l'exécution de chaque arborescence partielle donnent lieu à l'envoi de messages de synchronisation aux moniteurs dépositaires d'une requête partielle afin de permettre la poursuite des opérations de localisation et des transferts de données associés.

On décrit dans les paragraphes suivants les fonctions essentielles du traducteur du langage de manipulation de relations (cf §1.3), de la décomposition, du traitement de requêtes (cf §1.4) et du système d'exécution répartie (cf §1.5).

1.3 Utilisation de MICROBE

1.3.1. Introduction

Un utilisateur dispose de deux langages qui lui permettront de définir des données relationnelles et de poser des requêtes d'interrogation/modification sur une Base de Données.

Il existe :

i) Un langage de bas niveau dont les commandes sont représentées par les primitives de la mémoire relationnelle MIMER qui seront étudiées au chapitre 4. Ces primitives s'appellent comme des sous-programmes d'un langage de programmation hôte.

ii) Un langage relationnel de haut niveau MIQUEL <FRT81> que nous étudions ci-dessous.

1.3.2 Le langage MIQUEL

On s'intéressera essentiellement dans cette partie à:

-La présentation du langage MIQUEL et à

-la traduction de celui-ci en arborescences dont les noeuds seront choisis parmi les opérateurs de l'algèbre relationnelle.

Le plus connu des langages relationnels de haut niveau est le langage SQL mis en oeuvre dans SYSTEM-R <CHA76> (recémment comercialisé sous le nom SQL/DS <COD81>) dont une version réduite MINISEQUEL a été réalisée sur mini-ordinateur (Université de San Diego <ANB78>).

Le langage MIQUEL <FRT80> est inspiré de ce dernier pour deux raisons:

-C'est un langage de haut niveau basé sur quelques mots clés facilement assimilables même par un utilisateur inexpérimenté,

-sa structure de blocs récursifs permet néanmoins la construction de requêtes complexes.

Nous allons maintenant décrire les différents commandes du langage avant de nous intéresser à leur traduction en arborescence (cf §1.3.2.2).

1.3.2.1 Les commandes du langage

Les commandes du langage MIQUEL sont classées en quatre groupes:

i) Définition de données relationnelles,

ii) Manipulation de données,

iii) Consultation de caractéristiques de la Base de Données,

iv) Association entre une Base de Données et une session.

Nous allons dans la suite présenter brièvement ces

commandes; la syntaxe détaillée du langage sera donnée dans l'Annexe 1.

Le langage de définition de données est constitué des commandes suivantes:

.CREATE-DATABASE : Création d'une nouvelle Base de Données,

.SCRATCH-DATABASE: Destruction d'une Base de Données existante,

.CREATE-RELATION: Création d'une nouvelle relation,

.DELETE-RELATION: Suppression de l'une de relations de la Base de Données,

.CREATE-ATTRIBUT: Création d'un nouvel attribut sur l'une de relations de la Base de Données,

.DELETE-ATTRIBUT: Suppression d'un attribut existant sur l'une de relations de la Base de Données,

.CREATE-INDEX: Création dynamique d'un index selon un ensemble d'attributs d'une relation de la Base de Données.

Le langage de manipulation de données est constitué des commandes suivantes:

.INSERT-INTO: Insertion d'un nouveau tuple dans une relation,

.DELETE-FROM: Suppression de tuples d'une relation,

.UPDATE: Mise à jour de tuples d'une relation,

.SELECT: A partir d'une relation, on en produit une autre qui vérifie un certain nombre de prédicats.

Cette dernière relation peut être stockée dans la Base de Données ou simplement affichée. Il est possible aussi, d'appliquer des fonctions de calcul sur la relation produite par exemple: Moyenne arithmétique, Somme, etc. Dans §1.3.2.3 nous présenterons un exemple d'utilisation de cette commande.

L'utilisateur dispose d'un certain nombre de commandes permettant d'obtenir des informations sur la Base de Données et sur l'ensemble des relations:

.LIST-RELATIONS: Affichage de la liste de toutes les relations de la Base de Données et de leurs caractéristiques: nom externe, type, cardinalité, degré, longueur de tuple, type de stockage,

.CHARACTERISTICS-RELATION: Affichage des caractéristiques de l'une de relations de la Base de Données,

.LIST-ATTRIBUTS-FROM: Affichage de quelques (ou de tous les) attributs d'une relation et de leurs caractéristiques: nom externe, type, longueur et position dans le tuple,

.LIST-INDEX: Affichage des index construits sur une relation et de leurs caractéristiques: attributs faisant partie de la clé, type de l'index et ordre.

Trois commandes permettent le contrôle d'une session:

.OPEN-DATABASE: Mise en disponibilité d'une Base de

Données existante,

.CLOSE-DATABASE: Fermeture de la Base de Données sur laquelle on est en train de travailler,

.BYE: Terminaison d'une session MICROBE.

Enfin, deux remarques peuvent être faites:

-La plupart de commandes possèdent un code mnémonique abregé qui facilite leur emploi,

-Les commandes OPEN-DATABASE, SCRATCH-DATABASE, CREATE-DATABASE et INSERT-INTO sont interactives et donnent lieu à l'interrogation de l'usager pour l'obtention d'informations supplémentaires pendant leur exécution. Dans le cas de la commande CREATE-DATABASE par exemple, l'usager sera interrogé sur: le nom de la Base et de son Administrateur, le disque qui doit allouer la Base, le code d'exploitation et la taille de la Base en nombre de pages.

1.3.2.2 Traduction en arborescences

La représentation formelle d'une requête est une arborescence dont les noeuds sont des opérateurs de l'algèbre relationnelle (Selection, Projection, Jointure, Union, Intersection, etc.). Le premier problème qui se pose est comment à partir d'une expression en langage MIQUEL obtenir sa représentation. Nous allons étudier ici le cas de la traduction de la commande SELECT puisque celle-ci est spécialement délicate. Voyons, d'abord sa structure dans le cas le plus simple (la syntaxe complète est donnée dans

l'Annexe 1):

SELECT <liste-d'attributs>
FROM <liste-de-relations>
WHERE <expression> ;

<expression> est une liste de prédicats pouvant mettre en jeu d'autres commandes SELECT.

Les requêtes constituées par des commandes SELECT sont analysées récursivement, chaque clause WHERE étant traitée en une passe. Dans un deuxième temps, les connecteurs de clauses (AND et OR) sont utilisés pour relier entre elles les SELECT imbriqués, et donner lieu éventuellement à la génération d'opérations liant les opérandes de celles-ci <FRT80>.

1.3.2.3 Exemples de requêtes MIQUEL

Soit une Base de Données contenant les deux relations suivantes:

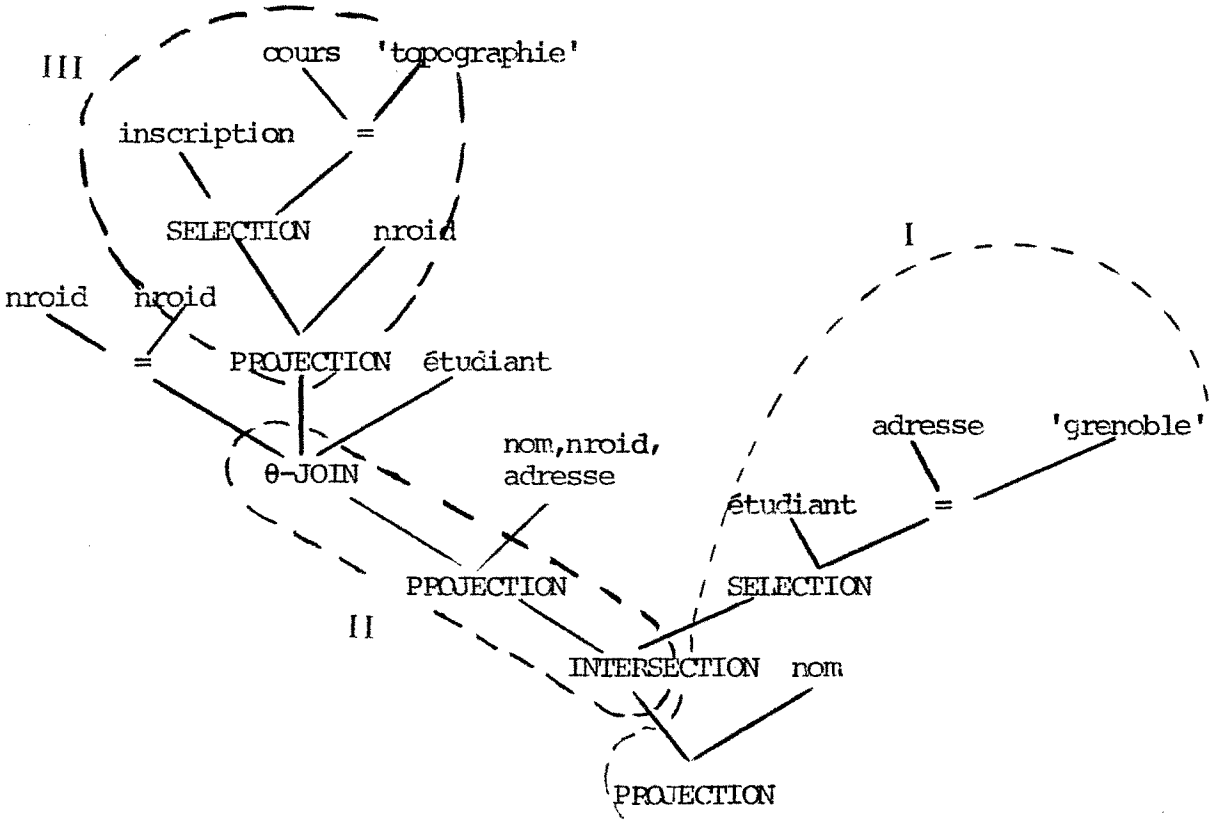
ETUDIANT(NROIDENTIFICATION, NOM, ADRESSE)
INSCRIPTION (NROIDENTIFICATION, COURS)

La première relation exprime pour chaque étudiant son numéro d'identification, son nom et son adresse. La deuxième relation indique les cours auxquels un étudiant identifié par un numéro est inscrit.

Considérons la question: "Obtenir la liste des étudiants habitant la ville de Grenoble et qui sont inscrits au cours de topographie". Elle conduit à la requête MIQUEL suivante:

```
Select NOM
From   ETUDIANT
Where  (ADRESSE = "GRENOBLE")
           and
           (NROIDENTIFICATION =
                   Select NROIDENTIFICATION
                   From   INSCRIPTION
                   Where  COURS = "TOPOGRAPHIE") ;
```

La production d'une arborescence à partir d'une requête MIQUEL ne fait pas l'objet du présent travail en conséquence, nous n'en discuterons pas ici; le lecteur interessé pourra se référer à <FRT80>. L'arborescence correspondante à la requête ci-dessus est schématisée ainsi:



I correspond au premier bloc SELECT,

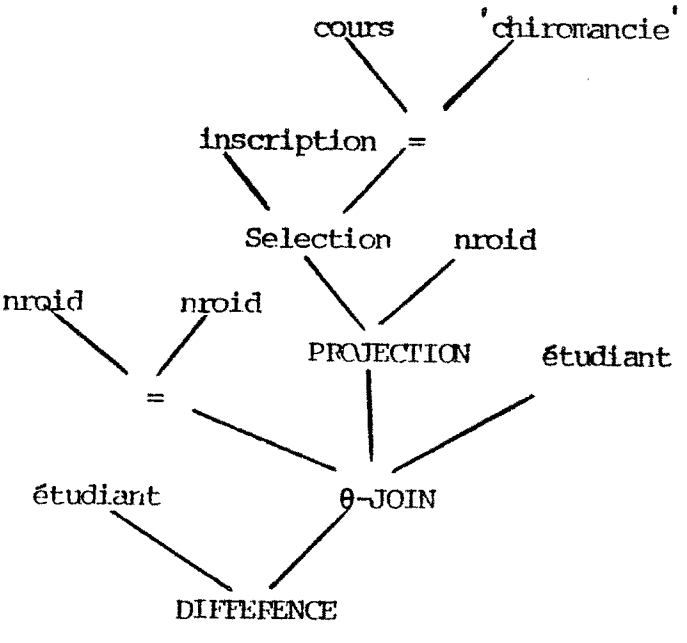
III correspond au deuxième bloc SELECT,

II correspond à la liaison entre les deux blocs due à NROIDENTIFICATION = Select ...

Un autre exemple d'une requête exprimée en langage MIQUEL est:

```
Delete-from ETUDIANT
Where       NROIDENTIFICATION =
                        Select NROIDENTIFICATION
                        From   INSCRIPTION
                        Where  COURS = "CHIROMANCIE" ;
```

Dans ce cas concret, l'action de supprimer des tuples peut être réalisée par l'opérateur de différence appliqué entre la relation ETUDIANT et une relation intermédiaire calculée au préalable qui contient les tuples à supprimer. L'arborescence correspondante est:



1.4 Décomposition et traitement de requêtes

1.4.1 Critères de décomposition

On trouve sur chaque serveur local du système MICROBE un Système Général de Manipulation d'Arborescences (SYGMA), chargé d'exécuter l'arborescence algébrique fournie par le compilateur du langage externe, ou une arborescence "équivalente", de façon "optimale".

Les différents objectifs d'optimisation retenus sont:

-Le temps de réponse d'une requête,
-La charge globale du système réparti,
-Le temps de réponse à toutes les requêtes posées à un instant donné.

Le premier objectif est suffisant dans le cas d'un système mono-utilisateur, ce qui est la première étape de réalisation de MICROBE. La minimisation du temps total de réponse (pour chaque et pour toutes les requêtes) deviendra essentielle pour les développements ultérieurs.

De nombreux facteurs interviennent dans la définition d'une fonction d'optimisation. Citons parmi les principaux:

-Les facteurs propres au réseau (taille des paquets transmis, délais de transmission),

-Les facteurs propres aux machines utilisées (vitesses d'accès à la mémoire centrale, temps moyen d'accès à un bloc disque, taille d'un bloc disque, charge instantanée de la machine),

-Les facteurs propres aux utilisateurs (fréquence des mises à jour, temps de réponse toléré, sécurité, integrité),

-Les facteurs propres aux données (volume des relations, existence des index primaires et/ou secondaires, sélectivité des attributs, structure des données en mémoire secondaire, méthodes d'accès utilisées),

-Les facteurs propres aux opérateurs relationnels. A chaque opérateur correspond plusieurs algorithmes qui dépendent du parallélisme interne (pipe-lining), des ressources mémoire nécessaires, de la priorité inter-opérateurs, de la localisation, etc...

Concrètement, on utilise dans MICROBE les paramètres suivants:

-La structure de l'arborescence à exécuter,
-La distribution de données (duplications, regroupements),
-La localisation des opérateurs,
-Le choix entre les divers algorithmes offerts par opérateur.

Dans la plupart des modèles d'optimisation existants à ce jour <YAO79>, <SDD79>, <EPS78>, <CHU79> les temps de traitements locaux son négligeables. Ces modèles sont inopérants en MICROBE car, dans notre environnement ces temps sont du même ordre de grandeur que les délais de transmission (réseau local à 1 Megabits de débit).

Il est donc nécessaire de revoir la démarche usuelle consistant à ne traiter les problèmes d'optimisation que localement, par l'intermédiaire des transferts de données.

A cet effet MICROBE possède:

-Un ensemble d'algorithmes de traitement pour chaque opérateur relationnel de base (avec ou sans pipe-lining, avec 0, 1 ou 2 semi-joins pour l'opérateur de join de deux relations, etc),

-Un module de restructuration des arborescences fournies par le compilateur du langage externe <GAL81>, qui tire parti des proprietés des opérateurs algébriques (commutativité, distributivité), de réduction d'expressions conditionnelles, etc,

-Un module de décomposition dynamique d'arborescences DOPAGE mettant en oeuvre une technique de localisation progressive des opérations associées aux noeuds, et de transfert à seuil pour les données échangées entre les serveurs locaux <NGU79>, <FER79>, <AZR81>.

SYGMA est chargé de synchroniser l'activité de ces différents composants, en utilisant chaque fois que cela est possible la technique de pipe-lining pour exécuter les opérations élémentaires. Un processus distinct est en effet dédié à chacune: cela permet de travailler de façon asynchrone en minimisant le stockage de données intermédiaires, et de faire simultanément du transfert d'informations et du traitement.

Par ailleurs, SYGMA peut provoquer dynamiquement une restructuration de l'arborescence restant à exécuter à un instant donné, si les résultats partiels successifs sont trop éloignés des caractéristiques fournies (cardinalité, sélectivité, etc) par les catalogues et des valeurs successives du seuil de transfert, fournies par l'algorithme de décomposition dynamique DOPAGE.

1.4.2 Exemple

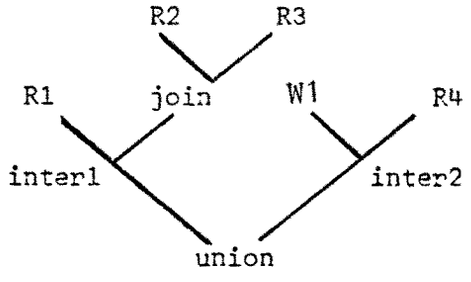
Nous illustrerons le fonctionnement de DOPAGE à l'aide d'un exemple tiré de <FER80>.

Supposons que la requête à exécuter sur la base répartie suivante

Relation	Site	Volume en octets
R1 (A,B,C)	S1	1500
R2 (A,F,G)	S2	1000
R3 (F,I,J)	S3	1500
R4 (I,K,L)	S3	1000

(où les sites serveurs S1, S2, S3 peuvent exécuter tous les opérateurs de l'algèbre relationnelle)

corresponde à l'arborescence:



où W1 est un noeud dupliqué de join (R2,R3). Le système devra tenir compte de cette duplication pour éviter le double calcul de l'opération join.

L'arborescence ci-dessus représente la requête algébrique:

Q: union (intersection (R1, join (R2,R3)),
intersection (R4, join (R2,R3)))

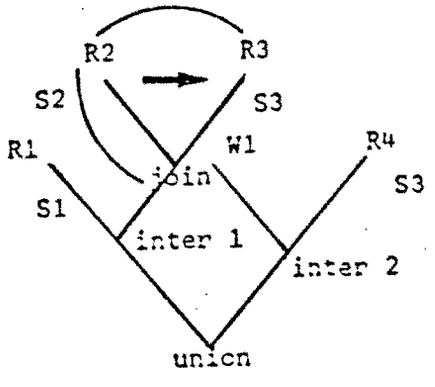


Figure 1.4

Par simplicité, on suppose de même que les opérateurs ne sont exécutés qu'à la disponibilité complète de leurs opérandes (c'est-à-dire sans pipe-lining).

Nous utilisons ici, l'algorithme DOPAGE <NGU79>, <FER79>, <AZR81> qui ne tient pas compte de données statistiques, les décisions de localisation d'opérateurs et de transfert d'informations sont prises à l'aide d'une fonction dite fonction à seuil. Celle-ci donne une idée du comportement de la requête à un instant donné, fonction de la connaissance effective des résultats partiels.

Dans notre exemple et par simplicité nous utilisons la fonction à seuil suivante:

$$T_0 = \frac{\sum_{i=1,n} |R_i|}{n}$$ où n est le nombre de relations de base mises en jeu dans la requête et |Ri| est le volume de la relation Ri.

$$T_{j+1} = |T_j + |\text{volume dernier transfert}|| / 2 \quad \text{pour } j \geq 0$$

Etape1

-Le site récepteur de la requête détermine les sous-arborescences monosite maximales. Dans le cas présent, puisqu'il n'y a aucun opérateur unaire (par exemple: project (R1,A,B)), cette étape consiste à localiser le nombre maximum d'opérations binaires. Seul le noeud join (R2,R3) peut l'être. Puisque le cardinal de R2 est tel que |R2| < |R3| la première décision prise sera:

-Transfert de R2 sur le site de R3, soit S3

-Exécuter join (R2,R3) sur le site de R3 (Figure 1.4).

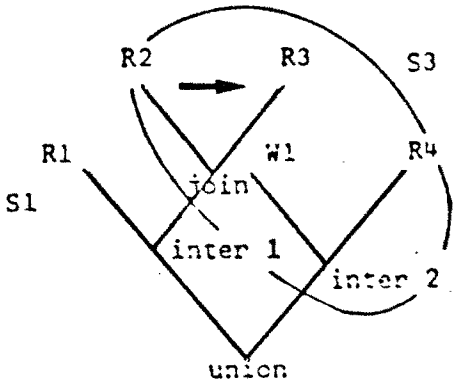


Figure 1.5

De même, W1 sera localisé sur S3. Comme W1 et R4 sont localisés tous deux sur S3, la deuxième décision prise est:

-Exécuter inter2 (W1,R4) sur S3

La troisième décision est de mettre à jour la valeur du seuil de transfert, c'est-à-dire:

$$T_1 = \frac{T_0 + |R2|}{2} = 1125$$

Nous supposons que la valeur initiale du seuil de transfert T_0 est la moyenne des cardinalités des relations R1 à R4, c'est-à-dire $T_0 = 1250$.

Etape2

-L'arborescence ainsi partiellement localisée est ensuite diffusée sur tous les sites avec les messages exprimant les trois décisions

précédentes. Chaque site récepteur exécute les ordres qui lui sont assignés, c'est-à-dire:

-Pour S2, transférer R2 à S3,

-Pour S3, attente de R2 et exécution du join(R2,R3).

Etape3

-Lorsque l'opération join (R2,R3) est terminée, en supposant |join (R2,R3)| = 100, le cardinal de ce résultat est diffusé sur tous les sites. Puisque |join (R2,R3)| < |R1|, le site S3 prend immédiatement les décisions:

-Transfert du résultat de join (R2,R3) sur le site de R1, c'est-à-dire S1,

-Mise à jour du seuil, c'est-à-dire:

$$T_2 = \frac{T_1 + 100}{2} = 612$$

Ces décisions sont diffusées sur S1,...,S4.

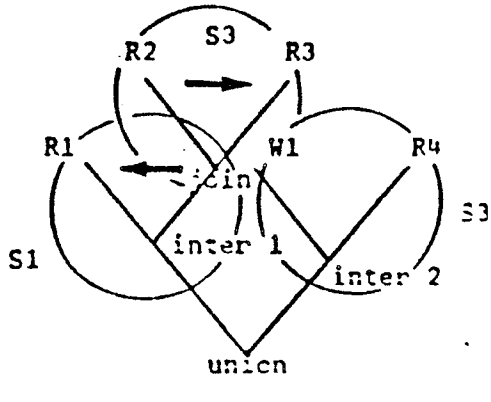


Figure 1.6

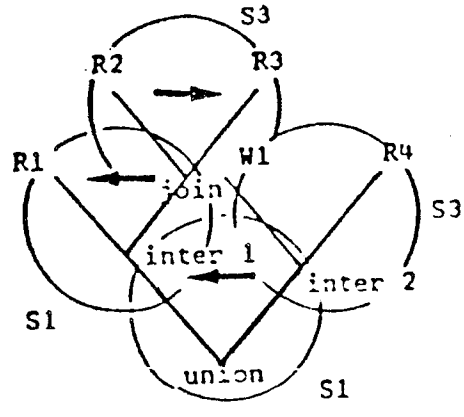


Figure 1.7

Le site S3 qui était en attente de W1 exécute parallèlement l'opération inter2 (W1,R4) (Figure 1.6).

Etape4

-lorsque le site S1 a reçu le résultat de l'opération join (R2,R3), l'opération inter1 (R1, join (R2,R3)) est exécutée.

Etape5

-Lorsque l'opération inter2 (W1, R4) est terminée, en supposant que |inter2 (W1,R4)| < 75 (c'est-à-dire |inter2 (W1,R4)| < T_2), S3 prend les décisions:

-Transfert du résultat de inter2 sur S1,

-Mise à jour du seuil, c'est-à-dire:

$$T_3 = \frac{T_2 + 75}{2} = 343$$

Ces deux décisions sont diffusées (Figure 1.7).

Etape6

-Lorsque l'opération inter1 (R1,join(R2,R3)) est terminée, le site S1 se met en attente d'un résultat de inter2. Lorsque celui-ci est effectivement reçu, S1 exécute l'opération d'union et transfère le résultat final à l'utilisateur.

1.5 Système d'Exécution Répartie (SER)

1.5.1 Objectifs

SER fait suite et s'inspire des travaux récents dans le domaine des systèmes d'exécution répartie réalisés dans le cadre du projet SIRIUS, notamment SIGOR <SER78> et MERE <ADE78>.

Appliqué ici dans le cadre d'un système de Bases de Données réparties, SER a néanmois été conçu dans un esprit de généralité, pouvant s'appliquer à une large gamme d'applications. En effet:

-Sur le plan des échanges entre processeurs interconnectés, SER s'appuie sur un service de transport classique (de type niveau 4 de l'Open System Architecture de l'ISO <ISO79>);

-Il n'interfère avec les systèmes locaux que par la connaissance d'un catalogue de commandes ou programmes locaux pouvant intervenir dans l'application répartie.

Les services fournis par SER sont essentiellement:

-activation de programmes à distance,
-enchaînement de ces programmes suivant les modalités d'exécution répartie spécifiées par MICROBE dans un Plan d'Exécution Répartie (PEX),

-transfert d'information entre programmes d'un PEX,
-expression sous forme logique et évaluation des conditions d'activation d'un programme, fonctions éventuellement des résultats d'autres programmes,
-utilisation à l'interieur d'un même PEX des possibilités de parallélisme offertes par la machine multiprocesseurs que constitue l'ensemble des processeurs interconnectés,
-détection et signalisation des pannes et des fins d'exécution normales ou anormales des programmes.

1.5.2 Concepts de base

Le fonctionnement de SER <BEN81> est caractérisé par un contrôle d'exécution entièrement répartie qui s'appuie sur trois entités de base: Action Locale (AL), Variable de Synchronisation (VS) et Fichiers de Messages Temporaires (FMT). Ces trois entités sont utilisées non seulement de manière interne à SER, mais également par MICROBE pour la formulation du Programme d'Exécution Répartie (PEX).

1.5.2.1 Action Locale

L'action locale est l'unité logique d'exécution de MICROBE. Elle correspond physiquement à l'exécution d'un programme sur un processeur. Ce peut être, par exemple, une requête de consultation sur la Base locale ou bien un tri-fusion.

Une action locale en cours d'exécution ne réalise aucune opération explicite de synchronisation ou de dialogue avec d'autres AL se déroulant en parallèle. Les seules actions qu'une AL peut avoir sur l'environnement d'autres AL d'un PEX sont la mise à jour de VS et la consommation ou la production de FMT.

1.5.2.2 Variable de Synchronisation

Les variables de synchronisation permettent de gérer les enchaînements d'actions locales. En effet, à toute demande d'action locale, figurant dans un PEX, MICROBE peut associer des variables de synchronisation. Pour chaque AL, une condition d'activation, définissant une configuration déterminé des VS associées, permet de fixer le moment de son lancement. De même, une expression de terminaison permet sur fin d'exécution de définir des mises à jour de VS associées à d'autres AL.

On utilisera, par exemple, une variable de synchronisation pour conditionner le lancement d'une AL sur la fin d'une autre.

1.5.2.3 Fichiers de Messages Temporaires

SER gère, de manière automatique, une zone de stockage réseau qui est utilisée pour stocker, de manière temporaire, les données d'entrée et de sortie des AL sous forme de Fichiers de Messages Temporaires. Les FMT se présentent

comme des fichiers séquentiels à articles de taille variable. Ils sont produits sans possibilité de retour en arrière et détruits après consommation. SER assure le transfert des FMT de l'AL productrice vers l'AL consommatrice en mode pipe-line.

1.5.2.4 Fonctions du SER

SER constitue par lui-même un système réparti, c'est-à-dire qu'il existe sur chacun des stations connectée au réseau un SER local assurant l'activation et le contrôle des actions locales de cette station pour le compte de MICROBE (Figure 1.8).

Un PEX, fourni sur un processeur donné, est composé de commandes demandant l'exécution d'AL sur les différents processeurs. Chaque commande définit le contexte de l'AL en spécifiant les conditions d'activation et les expressions de terminaison ainsi que les FMT qui seront produits ou consommés par l'AL. Le soumetteur de PEX identifie de manière unique et localisée, dans chaque commande, l'AL, les VS et FMT utilisés.

Un SER local de type "client" assure la prise en compte de chaque commande de création de contexte d'AL et son envoi à un SER local de type "serveur" sur le processeur d'exécution. Une fois le contexte d'exécution mis en place, le SER client n'intervient plus si ce n'est pour des échanges de commandes et d'informations de supervision

(indication de panne d'un processeur, compte-rendu de fin d'exécution ou abort). Le contrôle d'exécution est alors décentralisé au niveau des SER serveurs.

L'enchaînement des divers AL est commandé par l'évaluation des conditions d'activation dont les mises à jour se font de manière totalement répartie, sans référence au processeur initial.

Chaque SER serveur assure pour chaque contexte d'AL:

-la préparation de son lancement, c'est-à-dire l'évaluation de sa condition d'activation,
-le lancement effectif en liason avec le système d'exploitation local,
-la mise à disposition des données (FMT) nécessaires à l'exécution,
-le contrôle local de l'exécution ainsi que la prise en compte de la fin normale ou anormale entraînant l'évaluation et les mises à jour éventuelles de VS distantes,
-la prise en compte des données produites (FMT) et leurs transferts éventuels (données d'entrée d'autres AL).

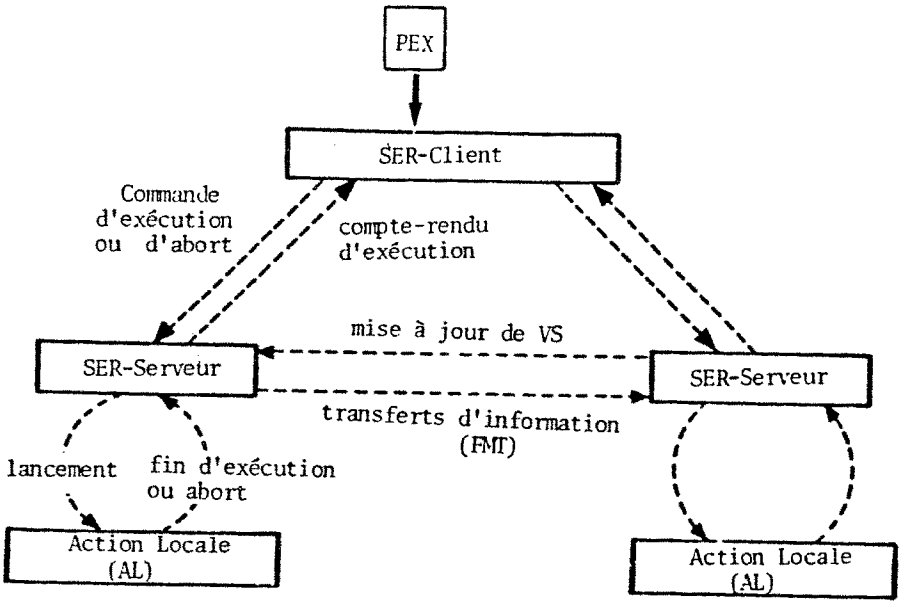


Figure 1.8 Structure d'exécution répartie de SER

1.6 La mémoire relationnelle

Cette partie étant le sujet de la présente thèse sera largement détaillée par la suite, nous nous contenterons ici donc, de signaler que: ce composant du système MICROBE permet de stocker et de récupérer toutes les informations de la Base de Données sur la mémoire sécondaire.

2 LES MEMOIRES RELATIONNELLES

2.1 L'état de l'art

SGBD RELATIONNEL	MATERIEL ET SYSTEME D'EXPLOITATION	METHODES D'ACCES
MRS(Micro-computer relational database System) <MRS79>	.DEC, LSI-11 UNIX 28k mots (1 mot=16 bits)	Initialement ISAM, postérieurement B-arbres
MINISEQUEL <ANB78>	.1 cpu 8080 ou LSI-11,28k mots (1 mot=16 bits) 2 Floppy disk spindles 1 floppy=500blc 1 blc=512bytes	.Sequential links, .B*-arbres .PM-tries
INGRES <ING76>	.DEC PDP 11/40 11/45, 11/70 UNIX	.à clé: -hashed -ISAM -compressed hash -compressed .sans clé
SYSTEM R <CHA76>	.IBM 370 VM	.B*-arbres .VSAM
La mémoire relationnelle LAMB <AND78>	.IRIS 80 SIRIS 8	.Espace virtuel paginé d'Uranus .VDAM
URANUS <NGU77>	.IRIS 80 SIRIS 8	.Espace virtuel paginé .VDAM
MICROBE <FER80>	.PLESSEY MICRO1 RSX-11M	.Séquentiel .B*-arbres .Hachage virtuel

Figure 2.1

Dans ce qui suit, nous allons décrire brièvement quelques mémoires relationnelles qui ont été conçues pour d'autres SGBD relationnels. Dans §2.2 nous mettrons en évidence les principales différences qui séparent notre approche de tels systèmes. Un tableau synoptique de quelques SGBD relationnels est présenté en Figure 2.1.

GAMMA-0 <BCD73>, XRM <LOR74>, RSS <CHA76> et LAMB <AND78> sont des systèmes de mémoires relationnelles. Les trois premiers ont été developpés par IBM et le dernier dans le cadre du projet POLYPHEME <ADI80a>. RSS et XRM ont fait partie de recherches dans le cadre du projet SYSTEM-R <CHA76> commercialisé récemment sous le nom SQL/DS <COD81>.

2.1.1 XRM (eXtended Relational Memory)

Il s'agit d'un système de mémoire relationnelle développé au centre scientifique IBM de Cambridge <LOR74> qui permet la mise en oeuvre de l'algèbre relationnelle, du calcul relationnel et éventuellement le développement d'applications plus sophistiquées.

XRM a été conçu pour être implémenté sur des gros ordinateurs (IBM 370). C'est ainsi que ce système a été adopté comme méthode d'accès relationnelle dans le premier prototype du SYSTEM-R. XRM est un système mono-utilisateur ne tenant pas compte des incohérences introduites lors d'une panne de la machine sur laquelle celui-ci est exécuté.

Chaque relation de la Base est répérée par un identificateur de relation, noté symboliquement idr, qui est une adresse interne à XRM; les attributs sont identifiés par leurs positions dans les relations auxquelles ils appartiennent. Les tuples sont identifiés par des identificateurs de tuples, notés symboliquement idt, qui sont aussi des adresses internes. Les idr ainsi que les idt sont visibles au programmeur d'une application et c'est lui qui doit les manipuler. Ceci rend difficile l'utilisation de XRM car, le niveau application se place à un niveau très bas.

Quatre types de relations sont gerées par XRM:

i) les relations classe: il s'agit de relations unaires où la longueur de chaque tuple est variable. Chacune de ces relations correspond à un attribut d'une (ou de plusieurs) relation(s) régulières(s),

ii) les relations régulières: Ce sont des relations n-aires dont les tuples ont une longueur fixe qui stockent n pointeurs vers des relations classe. La valeur d'un attribut est substituée à un pointeur qui est un idt d'un tuple d'une relation classe. Ce type de relations correspond aux relations de base créées par l'utilisateur du SGBD.

La Figure 2.2 représente deux relations régulières:

EMPLOYE (NOM, NRODEPT, ADRESSE) et

DPTO (NRODEPT, RESPONSABLE)

Les attributs NOM, NRODEPT, ADRESSE, et RESPONSABLE forment des relations de type classe. On notera que le responsable étant une personne, son domaine de valeurs est identique à celui des noms.

DPTO

idt		
i1	i5	i18
i2	i4	i20
i3	i6	i14

EMPLOYE

idt			
i7	i14	i6	i24
i8	i15	i4	i23
i9	i16	i6	i24
i10	i17	i5	i22
i11	i18	i6	i21
i12	i19	i4	i21
i13	i20	i5	i21

NRODEPT

idt	
i4	17
i5	21
i6	53

NOM

idt	
i14	COSME
i15	BRUN
i16	PERRIER
i17	FOURNAND
i18	FARRA
i19	CHANG
i20	MARTIN

ADRESSE

idt	
i21	PARIS
i22	ROME
i23	LYON
i24	GRENOBLE

Figure 2.2

Cette structure de pointeurs évite la répétition de valeurs à l'interieur d'un attribut

ainsi que la duplication d'un attribut appartenant simultanément à plusieurs relations.

. XRM optimise l'aspect non-duplication des informations, puisque des pointeurs sont substitués aux valeurs réelles; ceci se fait au détriment des performances d'accès, en effet l'accès à un tuple se traduit par une succession d'accès en mémoire secondaire pour récupérer chaque attribut de la relation, ce qui rend le coût d'accès prohibitif.

iii) les relations spéciales: sous ce nom sont classés deux groupes de relations, toutes les deux binaires:

.les relations de hachage: il s'agit de relations qui associent à chaque élément d'un attribut d'une relation, une valeur extraite d'une fonction de hachage, celle-ci permettant d'accéder à l'élément,

.les relations inverses: il s'agit d'une relation associée à un attribut permettant de lui accéder d'une façon ordonnée.

iv) la relation maître: celle-ci représente le catalogue du système. Chacun de ses tuples constitue un descripteur d'une relation de la Base de Données. Il contient pour chacune son type, son degré, la clé primaire, des informations de contrôle et des informations sur l'utilisateur.

Les relations sont physiquement stockées dans des blocs sur disque qui sont amenés à la demande, dans une zone de pagination en mémoire centrale.

XRM dispose d'un ensemble de primitives d'accès classées en trois groupes:

i) primitives concernant l'existence des relations:

.DEFINE crée une nouvelle relation. A cet effet la primitive insère un tuple dans la relation maître,

.DROP supprime une relation existante. A cet effet le tuple descripteur de la relation est supprimé de la relation maître,

ii) primitives pour le traitement des tuples:

.ADD et DELETE ajoute et supprime des tuples sur des relations existantes,

.FETCH permet d'accéder à un tuple dont l'idt est connu,

.TID retourne l'idt d'un tuple dont la clé est connue,

.UPDATE permet la modification d'attributs non-clés d'une relation,

iii) primitives pour le traitement de relations:

.OPEN/NEXT/CLOSE permettent de récupérer des tuples

(ou des idt de tuples) de relations au moyen de balayages,

.NUMBER retourne la cardinalité d'une relation,

.EMPTY vide une relation tout en conservant son descripteur dans la relation maître,

.INVERT associe une ou plusieurs relations inverses à un ou plusieurs attributs d'une relation régulière,

.RETRIEVE crée une liste d'idt des tuples d'une relation qui vérifient un certain nombre de prédicats. Chaque prédicat est évalué sur un attribut qui doit avoir nécessairement une relation inverse.

Ce système ne prévoit pas l'adjonction dynamique de nouveaux attributs à des relations existantes. Il n'est pas non plus possible d'effectuer l'opération inverse, c'est-à-dire la destruction d'attributs d'une relation existante. On verra (cf §2.2.1 et §4.1.3) que cette contrainte n'existe pas dans MIMER.

2.1.2 RSS (Relational Storage System)

Récemment, une version du SYSTEM-R <CHA76> a été commercialisée pour des gros systèmes (plus de 1 Moctets de mémoire centrale !). Il s'agit de SQL/DS <COD81>. SYSTEM-R comporte une mémoire relationnelle appelée RSS, celle-ci a été substituée à XRM. RSS comprend des mécanismes assurant l'accès concurrent par plusieurs utilisateurs, la cohérence des informations et la reprise du système en cas de panne.

Les relations sont stockées dans un espace virtuel qui est divisé en segments logiques, chacun d'eux est composé de pages physiques de taille 4K octets, celles-ci sont rangées en mémoire secondaire et amenées en mémoire centrale à la demande dans une zone de pagination.

Chaque tuple d'une relation est de longueur variable et identifié par un idt (la concaténation de l'adresse d'une page et d'un déplacement). Celui-ci n'est pas tout à fait l'adresse du tuple mais, d'une zone dans l'en-tête d'une page où l'on stocke toutes les adresses des tuples de la page (Figure 2.3). Ce mécanisme d'indirection permet de restructurer aisément les tuples d'une page. Ceci coûte, bien entendu, deux accès en mémoire pour chaque lecture/écriture d'un tuple.

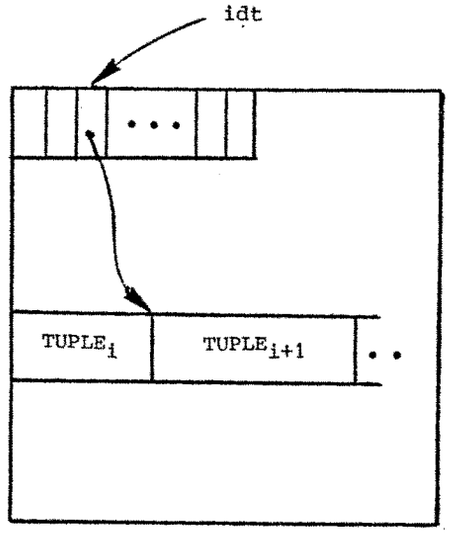


Figure 2.3 L'adressage d'un tuple dans RSS

Contrairement au système XRM, RSS stocke physiquement un tuple comme une suite contigue d'attributs, diminuant ainsi, le nombre d'accès lors de la manipulation d'un tuple.

Etant donné qu'un segment peut contenir plusieurs relations et qu'une page contient des tuples appartenant à plusieurs relations, un préfixe est ajouté à chaque tuple afin d'identifier à quelle relation il appartient. En plus, le préfixe contient d'autres informations telles que: des liens vers d'autres relations, le nombre de ces liens et le nombre d'attributs présents dans le tuple. RSS permet de créer des index sur des relations de façon à ce que celles-ci puissent être accédées selon un ordre défini par un attribut ou une combinaison de leurs attributs. De même, il est possible de relier les tuples d'une relation avec les tuples d'une autre au moyen des objets appelés liens. Pour cela des chaînes de tuples sont établies par l'intermédiaire des préfixes stockés dans les tuples. Ceci permet d'établir des chemins d'accès rapides entre deux relations; par exemple l'opération de join pourra être exécutée avec une méthode navigationnelle et retrouver très rapidement les tuples correspondants d'une relation dans l'autre.

Une interface RSI (Relational Storage Interface) sous forme de langage de primitives permet:

i) d'accéder aux différents segments,

ii) d'établir des transactions et des verrouillages sur les segments,

iii) d'accéder ou d'insérer des tuples dans les relations ainsi que d'établir des balayages sur celles-ci,

iv) de définir ou de détruire des relations,

v) de définir de nouveaux attributs dans les relations.

Enfin, RSS est très "souple" mais ne permet pas la destruction dynamique d'attributs existants dans les relations de la Base de Données, comme MIMER (cf §4.1.3).

2.1.3 Autres mémoires relationnelles

GAMMA-0 <BCD73> est à notre connaissance le premier système de mémoire relationnelle conçu; il n'a pas abouti à une implémentation. Celui-ci a été développé au centre scientifique IBM de San José et ses caractéristiques sont fortement semblables à celles du système XRM décrit dans le paragraphe précédent.

La mémoire relationnelle LAMB <AND78> a été conçue et réalisée pour être un composant de POLYPHEME <ADI80a>; malheureusement, elle n'a jamais été réellement intégrée dans le prototype de ce SGBDR. Son apport le plus original est de permettre la définitions de types de relations. Chaque relation doit appartenir à un type préexistant dans la Base de Données. Celui-ci est défini soit par LAMB elle-même (par exemple le type catalogue) soit, par l'utilisateur. Enfin, un type identifie une famille de relations qui ont des caractéristiques communes, à savoir: les mêmes attributs, clés, degrés et longueur de tuples.

Une autre approche, dans la conception de mémoires

relationnelles, est actuellement en étude dans les projets SABRE <GAR80>, VERSO <BAN80>, RAP <BRU80>, RARES <SMI79> et DIRECT <WIT78> qui sont en cours de réalisation et qui doivent aboutir à l'implémentation de prototypes sur des Machines Bases de Données. Il s'agit de mémoires relationnelles dont les primitives sont des opérateurs cablés qui sont intégrés dans le "hardware" de ces machines.

2.2 Notre solution: MIMER

2.2.1 Objectifs

A l'origine du projet MICROBE, est apparu la nécessité de construire un sous-système assurant une traduction d'un schéma conceptuel relationnel en un schéma physique (Figure 2.4). Ceci constitue initialement la motivation principale

Figure 2.4 MIMER: Un traducteur de schémas

de notre démarche. Ultérieurement, nos recherches se sont poursuivies beaucoup plus loin et nos objectifs sont devenus plus généraux; parmi ces derniers figurent principalement:

i) La construction d'un système relationnel sur micro-machine,

ii) L'étude et la mise en oeuvre d'un outil de traitement pour des applications comportant de faibles volumes

d'information (cas assez général de l'usager d'une micro-machine),

iii) La réalisation d'une structure de données adaptée aux usagers dont les besoins évoluent. Une structure qui puisse être donc, modifiée dynamiquement en fonction de besoins,

iv) Enfin, objectif à notre avis le plus important, la réalisation d'un outil de base autour duquel divers recherches puissent être construites. Ceci parce que nous avions constaté que lors du démarrage d'un nouveau projet ou d'une nouvelle application en Bases de Données il était nécessaire de recommencer à zéro du mois, en ce qui concerne l'implantation du logiciel.

Trois sous-objectifs se dégagent de ceci:

v) La transportabilité d'un tel système sur différentes machines, qui implique une grande independance par rapport à l'environnement sur lequel on travaille,

vi) L'extensibilité du sytème de façon à ce que de nouveaux éléments puissent y être introduits facilement et,

vii) La facilité d'utilisation du système.

Par la suite, nous étudierons comment la version opérationnelle mise en oeuvre répond aux objectifs ci-dessus:

i) Système relationnel sur micro-machine: MIMER a été mis

en oeuvre sur des micro-ordinateurs PLESSEY MICRO1/MICRO2 qui sont construits autour de micro-processeurs de type LSI11 ayant 64K Octets (128K ou 256K Octets) de mémoire centrale. De plus MIMER est par lui-même un système capable de manipuler des relations et des tuples conformes à la définition du modèle relationnel donné par CODD <COD70>,

ii) Faibles volumes d'information: La taille maximum d'une Base de Données est actuellement limitée par la surface d'un disque (2.5M Octets) ou d'une disquette (500K Octets),

iii) Besoins évolutifs: Cet objectif a été largement atteint du fait que les relations, les attributs et les tuples peuvent être détruits ou modifiés et que de nouveaux peuvent être insérés à n'importe quel instant de la vie de la Base de Données.

iv) Outil de base: MIMER est largement indépendant des machines PLESSEY car, il est programmé en langage PASCAL; ce qui veut dire qu'il peut être transporté sur d'autres machines avec un minimum de modifications. Celles-ci sont inhérentes aux différences de chaque compilateur PASCAL.

L'extension de MIMER ne pose pas de problèmes car, celui-ci est un ensemble de modules, chacun ayant une fonction spécifique qui évite les effets de bords. Des extensions seront possibles en utilisant les interfaces de niveau B et C du langage de primitives (cf §3.9.2).

Enfin, dans §2.2.2 nous continuons à répondre à cet objectif en montrant une liste non-exhaustive de quelques domaines dans lesquels MIMER pourrait constituer un outil de base.

2.2.2 Les differents domaines d'utilisation de MIMER

On se place dans une optique d'utilisation de MIMER comme couche de base pour la définition et la manipulation des données relationnelles. Rappelons que cette couche a servi à l'implantation du langage MIQUEL dans le projet MICROBE, mais que de par sa généralité elle peut également être utilisée directement par des programmes d'application écrits en PASCAL.

MIMER est un logiciel très général conçu pour être inséré dans des environnements divers. Ainsi, dans MICROBE c'est l'interpréteur <LEE81> qui, bien qu'utilisant un grand nombre de primitives, n'exploite pas toutes les possibilités de MIMER.

Par rapport à d'autres systèmes de mémoire relationnelle, dans MIMER on a cherché à raffiner quelques aspects, tels que:

i) Libérer l'interface application de la nomination de tuples des relations par leurs adresses internes. Ceci situe l'interface à un niveau plus haut. L'accès aux tuples est fait au moyen de balayages sur les relations (cf §4.1.4),

ii) Fournir un schéma conceptuel qui soit aisément modifiable. L'ajout et la destruction dynamique d'attributs et des relations est possible à n'importe quel instant de la vie de la Base de Données,

iii) Fournir des primitives bien conçues pour le balayage des relations. Ce sujet a été particulièrement bien étudié et c'est ainsi que ces balayages peuvent être (cf §4.1.4) conditionnés par un certain nombre de prédicats qui sont reliés entre eux par des opérateurs booléens. Cet aspect, en particulier, à notre avis d'une grande puissance n'a pas encore été exploité dans l'interprétation <LEE81> du langage MIQUEL.

Nous voyons MIMER comme un outil général de stockage et de consultation relationnel sur lequel il est possible de réaliser un certain nombre d'applications réelles et/ou de recherche, dont nous allons donner une liste non-exhaustive.

2.2.2.1 MIMER, un support d'implantation de langages de SGBD

MIQUEL (cf §1.3) est le seul langage de manipulation et de définition de données implanté sur MIMER à l'heure actuelle. Une requête exprimée dans un langage algébrique, tel que celui d'URANUS <NGU77> et de POLYPHEME <ADI80a>, peut être traduite en une arborescence algébrique, comme celles que nous avons étudiées dans §1.3.2.2, puis interprétée. Une implantation de ce langage pourrait être réalisée sur MIMER dans un très court délai car, ce type de langage n'est pas comme MIQUEL, loin de la structure

arborescente. La phase d'interprétation serait faite grâce à l'interpréteur d'arborescences déjà existant dans MICROBE.

De même, MIMER pourrait servir comme support d'exécution pour d'autres langages qui pourront être aussi traduits en arborescences ou interprétés directement sur MIMER; ces langages peuvent être:

i) de type prédicatifs, par exemple ALPHA <COD72>,

ii) de type graphique comme Query-By-Exemple <ZLO77>. Une réalisation de ce langage est en cours à l'IMAG pour une Base de Données documentaire <VEL81> ayant pour support le SGBD SOCRATE,

iii) basés sur l'emploi de types abstraits <SCH78>, <LOC79>, <SAL80>,

iv) de traitement de texte,

v) de type transactionnel, où chaque transaction est un module "programmé" dans un langage de haut niveau qui fait appel aux differentes primitives MIMER,

vi) naturels <COD74> ; il s'agit d'un domaine sur lequel on a relativement peu avancé, bien que les techniques modernes tendent à rapprocher les Bases de Données des utilisateurs non-informaticiens,

vi) mixtes; il s'agit de langages intermédiaires entre ceux de type prédicatifs et ceux de type algébrique, par exemple SQL, MIQUEL, etc.

3 DESCRIPTION DE MIMER

3.1 Introduction

MIMER est un logiciel qui possède quatre fonctions principales :

i) Assurer une "vision" relationnelle des fichiers constituant la Base de Données. C'est-à-dire qu'un utilisateur de MIMER ne verra pas ses données stockées sous forme de fichiers mais sous forme de relations et de tuples conformément au modèle relationnel défini par CODD <COD70>. Un langage de primitives permet la création, la consultation, la mise à jour et la destruction de relations, d'attributs et de tuples.

ii) Gérer la mémoire centrale. Les relations sont découpées et stockées en pages de taille fixe qui sont amenées en mémoire centrale à la demande, grâce à des mécanismes de mémoire virtuelle.

iii) Gérer les différentes relations de la Base de Données. Trois types de relations ont été définis: les relations de base qui sont créées par l'utilisateur, les relations internes qui stockent des informations sur toutes les relations de la Base de Données et, les relations inverses qui permettent d'accéder aux relations de base d'une façon ordonnée.

iv) Etablir de méthodes d'accès. Deux méthodes d'accès aux données ont été adoptées dans MIMER: la méthode séquentielle et les B*-arbres, une troisième est envisagée: le hachage virtuel (cf §5).

Trois modules constituent MIMER (Figure 3.1):

i) un ensemble non redondant (pour des raisons de facilité d'utilisation, il est plus fonctionnel que minimal) de primitives de manipulation et de définition de tuples et de relations,

ii) un ensemble de méthodes d'accès,

iii) un ensemble de données correspondant aux relations qui sont stockées selon une structure fixée par la méthode d'accès correspondante.

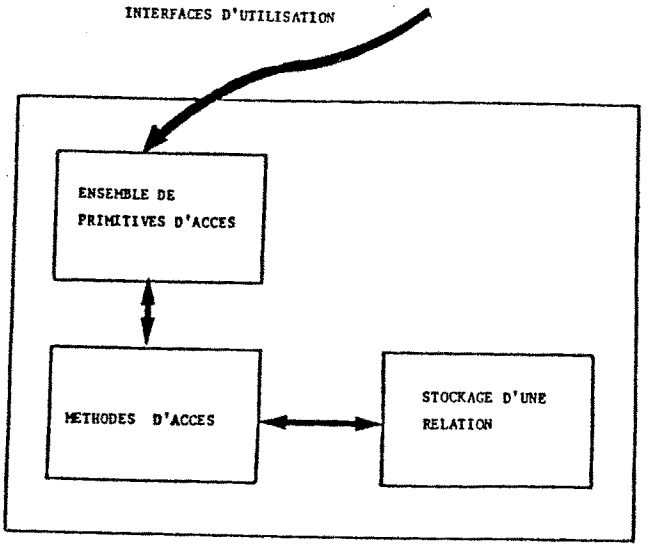


Figure 3.1 Architecture fonctionnelle de MIMER

La Figure 3.1 illustre le parcours d'une requête quelconque qui se traduit par un ensemble d'appels (fréquemment itérés) aux primitives de MIMER.

Le présent chapitre est consacrée à la présentation de la structure de MIMER et des différents objets qu'elle manipule.

3.2 Types de relations

Trois types de relations peuvent être gérés par MIMER:

i) les relations internes qui représentent les catalogues et les informations de contrôle de la Base de Données (cf §3.7 et §3.10).

ii) les relations de base qui stockent les données de l'utilisateur et qui sont créées par ce dernier,

iii) les relations inverses, qui sont des index sur les relations de base et créées à la demande de l'utilisateur.

Une relation inverse permet d'accéder rapidement à une relation de base, en établissant un adressage associatif sur la relation. De plus, elle donne une "vue" ordonnée (ascendante ou descendante) de la relation selon un critère d'ordre fixé par un (ou plusieurs) de ses attributs.

Logiquement, une relation inverse comporte 2 attributs:

.l'attribut CLE qui permet de distinguer un tuple

des autres tuples de la même relation. Si cette clé est unique, nous appelerons index primaire la relation inverse ainsi formée. De même, la clé peut être non-unique (possibilité de clés repétées) et nous appelerons alors la relation inverse: index secondaire (ou image dans d'autres systèmes).

.l'attribut POINTEUR qui stocke les idt des tuples correspondants dans la relation de base pointée.

Le lecteur pourra trouver un exemple de relations inverses au §4.1.7.

3.3 La liste d'espace disponible

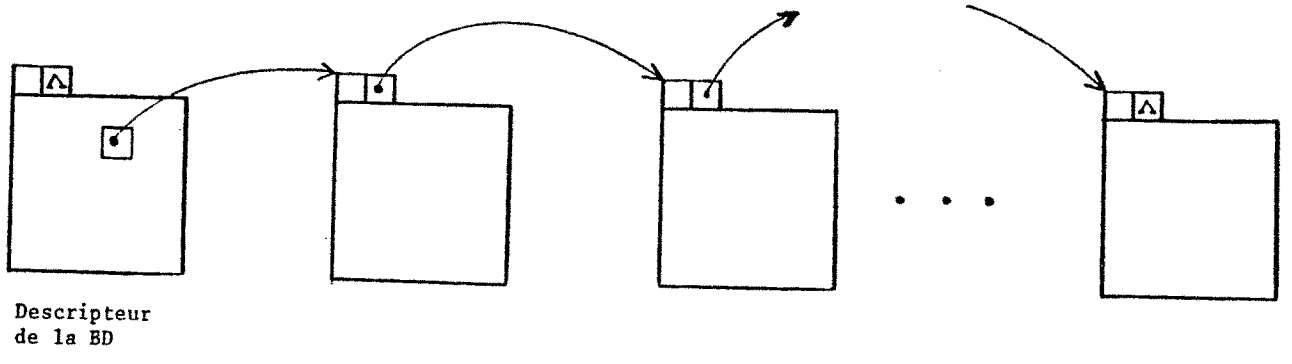


Figure 3.2 La liste d'espace disponible

L'ensemble de données est stocké physiquement sur un support magnétique permettant l'accès direct aux informations. Une Base de Données MIMER réserve un certain nombre de blocs sur ce support pour constituer un espace réel d'adressage. Chaque bloc est identifié par une adresse et correspond à un enregistrement physique d'un fichier qui

est accédé et recopié en mémoire centrale par la méthode d'accès direct aux fichiers, disponible dans le logiciel du micro-ordinateur.

L'espace disponible est constitué par la chaîne des blocs libres de la Base de Données (Figure 3.2). Celle-ci est gérée avec une politique LIFO ("Last In First Out"). A la création de la Base de Données tous les blocs appartiennent a cette liste. L'adresse du premier bloc est stockée dans le descripteur de la Base de Données (cf §3.4) et le dernier bloc contient l'adresse NIL.

Nota Bene: Dans notre implémentation le support magnétique correspond à la surface d'un disque de 2.5M Octets ou d'une disquette de 500K Octets. Un bloc a une taille de T octets; T est égal à 512, valeur maximum imposée par le logiciel de gestion de fichiers utilisé. Cependant T est un paramètre de génération qui peut être modifié (cf Annexe 2). C'est ainsi que la Base de Données pourra avoir approximativement jusqu'à 4882 blocs dans le cas d'un disque ou 976 blocs dans le cas d'une disquette.

3.4 Le descripteur de la Base de Données

Le premier bloc (bloc d'adresse 1) contient des informations concernant l'ensemble des données. Il comporte donc, sur la Base de Données, les informations suivantes :

i) son nom

ii) l'identification de son administrateur,

iii) l'identification du support magnétique où elle est stockée,

iv) un code d'exploitation,

v) sa taille en nombre de blocs,

vi) un pointeur vers le premier bloc de la liste d'espace disponible.

L'administrateur est le seul utilisateur, muni de tous les droits sur la Base de Données et le point ii) ci-dessus, est en fait un mot de passe qui l'identifie.

Nota Bene: Dans notre réalisation i), iii) et iv) représentent respectivement: le nom d'un fichier, le nom logique d'un périphérique et l'UIC (User Identification Code) selon le système d'exploitation RSX-11M <RSX11>.

3.5. Le stockage d'une relation

Une relation est découpée en pages logiques de taille fixe, puis à chaque page logique une transformation associe un bloc (que nous appelerons page physique), et un seul, en mémoire secondaire.

Chaque tuple d'une relation est distingué des autres au moyen d'un identificateur de tuple noté symboliquement idt qui est un couple formé de l'adresse de la page physique et

d'un déplacement (en nombre de tuples) par rapport au début de cette page. L'idt est en fait l'adresse interne du tuple.

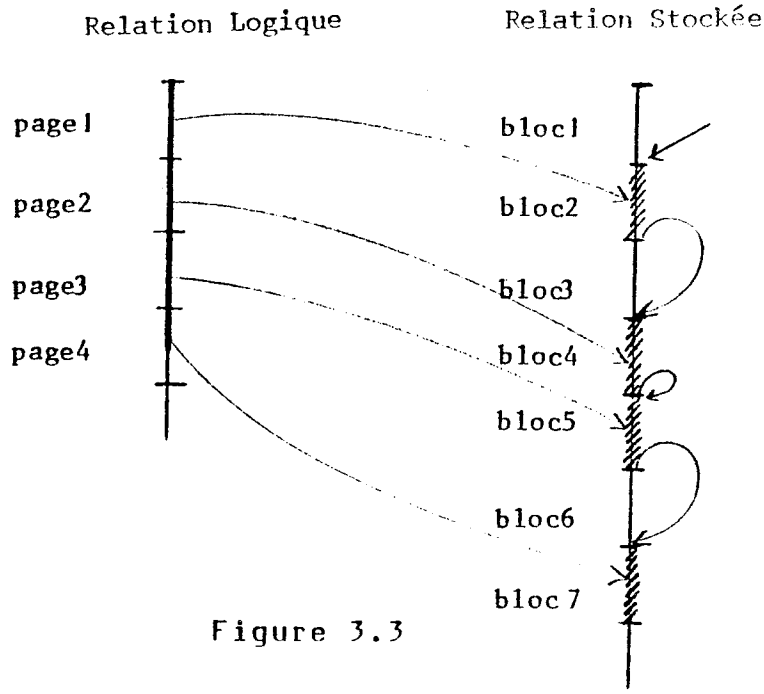


Figure 3.3

Les pages physiques ne sont pas forcément stockées séquentiellement mais la relation peut être reconstruite grâce à un chaînage (Figure 3.3). Un bloc est une page physique d'une relation ou un maillon de la liste d'espace disponible. Un bloc représentant une page d'une relation est relié (par l'intermédiaire de son en-tête) aux autres blocs de la relation.

Dans ce qui suit nous employerons, par abus de langage le terme page au sens de page physique.

3.6 Le format d'une page

Chaque page physique a le format suivant (Figure 3.4) :

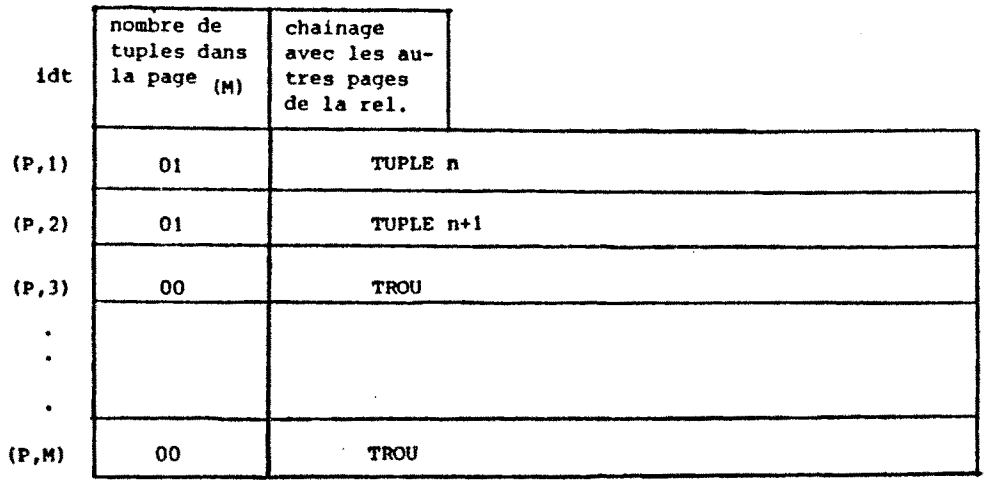


Figure 3.4 Format de la page d'adresse P d'une relation

l'EN-TETE de la page qui contient:

i) Le nombre maximun de tuples dans la page (sur 2 octets).

Il est égal à:

⌈(T-4)/<longueur-d'un-tuple-de-la-relation>⌉

ii) Le chaînage de la page avec la suivante, ou NIL cas de la dernière page de la relation (sur 2 octets).

les TUPLES :

Les tuples sont stockés les uns après les autres. Chacun est précédé d'un témoin d'existence de tuple (sur 2 octets). Ce témoin constitue un nouveau attribut de la relation, qui est créé automatiquement lors de la création de la relation. Celui-ci aura la valeur '00' s'il s'agit d'une place disponible ("un trou") et '10' si la place est occuppée effectivement par un tuple.

Un tuple ne peut pas être fragmenté entre plusieurs pages. Ceci, evidemment, limite la taille d'un tuple à la taille d'une page.

3.7 Les catalogues

La Base de Données est gérée par un ensemble de catalogues qui sont des relations internes.

Quatre catalogues (Figure 3.6) gérés directement par MIMER, décrivent toutes les relations de la base et établissent une liaison avec la mémoire secondaire:

i) La relation MAITRE : chacun de ses tuples est un descripteur de l'une des relations de la Base. Ce descripteur contient les informations suivantes: le nom externe, le type, la cardinalité, le degré, la longueur d'un tuple, le type de stockage, des liens

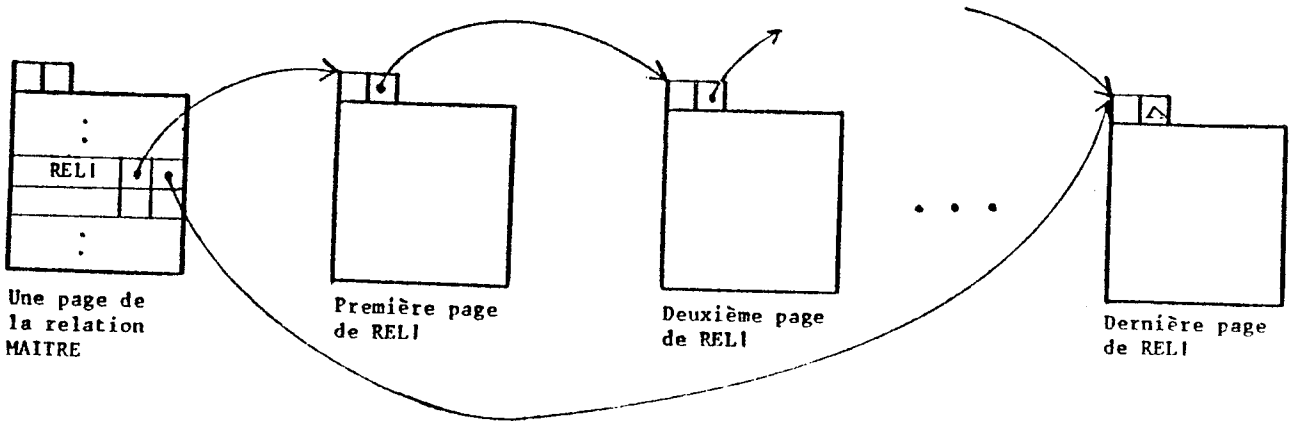


Figure 3.5

vers les relations INDEX et ATTRIBUTS, et l'adresse de la première et de la dernière page de la relation (Figure 3.5). De plus, dans le cas d'un SGBD distribué

on y stockera le site de localisation de chaque relation.

ii) La relation ATTRIBUTS : chaque tuple décrit un attribut d'une relation. L'insertion et la destruction dynamique d'attributs peuvent être réalisées grâce à un chaînage des attributs par relation. En outre, le nom, le type (Réel, Entier, Caractère), la longueur et la position d'un attribut dans une relation y sont stockés.

iii) La relation INDEX : chacun de ses tuples décrit une relation inverse (cf §3.2 et §5) créée sur une relation de base. C'est-à-dire, chaque tuple stocke: un pointeur vers le descripteur de l'inverse dans la relation MAITRE, un pointeur vers le premier attribut-clé dans la relation ATTRIBUTS-CLE et un indicatif signifiant le type d'inverse (Primaire ou Secondaire). De même un attribut supplémentaire permet de chaîner toutes les inverses existant sur une relation donnée.

iv) La relation ATTRIBUTS-CLE : elle sert à établir les attributs constituant les clés des relations inverses. De même, elle pointe vers la description des attributs correspondants dans la relation ATTRIBUTS.

	témoin	nom externe de la relation	type de relation	cardinalité	degré	longueur d'un tuple	type de stockage	pointeur vers la relation INDEX	pointeur vers la relation ATTRIBUT	pointeur première page relation	pointeur dernière page relation	site (cas distribué)
1												
2												
3	1	R	B	253	4	22	SSE	4	2	17	27	
4	1	R1	I		2	8		4	1			
5	1	R2	I		2	8		2	8			
6												
:												
n												

Exemple :

Soit la relation R (A, B, C, D)
ayant l'index primaire R1 avec clé A, B et
l'index secondaire R2 avec clé B, D

RELATION ATTRIBUTS

	témoin	nom externe attribut	type attribut	longueur attribut	position attribut dans le tuple	chainage attributs d'une relation
1	1	PTR	P	4	1	6
2	1	A	R	2	1	3
3	1	B	E	2	3	4
4	1	C	C	10	5	5
5	1	D	R	2	15	NUL
6	1	AB		4	5	NUL
:						
n						

RELATION INDEX

	témoin	pointeur vers la relation MAITRE	type index	chainage index d'une relation	pointeur vers ATTRIBUTS CLE
1					
2	1	5	I	NUL	4
3	0				
4	1	4	P	2	3
5					
6					
.					
p					

RELATION ATTRIBUTS-CLE

	témoin	chainage attributs de la clé	pointeur vers la relation ATTRIBUTS
1			
2			
3	1	6	2
4	1	5	3
5	1	NUL	5
6	1	NUL	3
.			
q			

Figure 3.6 Les catalogues

3.8 La mémoire virtuelle

Toutes les pages constituant la Base de Données sont amenées en mémoire centrale à la demande, dans une zone de pagination. Le nombre de pages résidentes en même temps en mémoire centrale est l'un de paramètres de génération de la Base de Données (cf Annexe 2).

Les pages sont remplacées dans la zone de pagination pour y placer des nouvelles avec une politique LRU ("Last Recently Used"). C'est-à-dire que si une page doit être amenée en mémoire centrale et qu'il n'y ait plus place, l'une des pages résidentes devra être libérée, on choisira donc, celle qui a été utilisée la moins récemment. Celle-ci sera récopiée sur sa place correspondante en mémoire secondaire, seulement si elle a été modifiée lors de son "séjour" en mémoire centrale.

3.9 Les primitives d'accès

Dans cette première version de MIMER, un ensemble de primitives permet de présenter aux utilisateurs (il s'agit d'utilisateurs ayant accès à des interfaces de bas niveau) une vision relationnelle de la mémoire. Dans des versions ultérieures, on pourra s'intéresser à l'élaboration de primitives qui garantissent, entre autres, la cohérence, l'accès concurrent et la protection de la Base de Données et de ses relations.

3.9.1 Introduction

MIMER est utilisable par l'intermédiaire d'un langage de primitives de bas niveau qui doivent être insérées dans un langage hôte de haut niveau.

Dans le cas concret de notre implémentation les primitives sont représentées au moyen de procédures et de fonctions PASCAL; toutes comportent un paramètre de sortie indiquant le type d'erreur commise éventuellement.

Une base peut être créée, ayant comme proprietaire un certain Administrateur, par la primitive CREER-BASE. DETRUIRE- BASE réalise l'opération inverse.

La création et la destruction de relations de la base sont assurées par les primitives CREER-RELATION et DETRUIRE-RELATION. Pour les attributs nous avons prévu CREER-ATTRIBUT et DETRUIRE-ATTRIBUT. Bien entendu, les attributs faisant partie de la clé d'un index ne pourront pas être détruits par ce moyen.

La primitive VIDER-RELATION détruit les données d'une relation sans pour autant détruire la définition de la relation.

INSERER-TUPLE ajoute un tuple à une relation tout en respectant le type de stockage défini par la primitive DEFINIR-STOCKAGE.

Il est aussi possible de créer des index primaires ou secondaires en utilisant CREER-RELATION-INVERSE. Celle-ci crée un B*-arbre (cf §5) qui est mis à jour lors des insertions ou des suppressions de tuples. Plusieurs inverses peuvent être définies sur une même relation.

GROUPER-RELATION offre la posibilité de trier et de grouper une relation selon un ordre défini sur un ou plusieurs de ses attributs.

A la différence des autres systèmes de mémoire relationnelle connus, nous avons mis l'accent sur la création de primitives qui permettent de traiter les tuples des relations de base, de façon à ce que leur identification (idt) soit invisible aux utilisateurs des interfaces qui se voient placés à un niveau beaucoup plus haut. L'utilisation du langage est aussi sensiblement allégée.

Une gamme de primitives dites de balayage permet l'accès séquentiel ou selectif aux tuples. Deux types de balayages sont présents dans MIMER: simple, c'est-à-dire que la relation est parcourue séquentiellement, telle qu'elle est stockée et, indirect, lorsque la relation est parcourue par l'intermédiaire d'un index. CREER-BALAYAGE crée un curseur de balayage dont la valeur est un pointeur sur les tuples d'une une relation; cette valeur est mémorisée dans la relation interne CURSEURS-ACTIFS (cf §3.10). CONDITION-BALAYAGE établie une condition de balayage, et TUPLE-SUIVANT fait avancer le curseur et

récupère un tuple. A l'inverse, la primitive COMP teste une condition sur un tuple et ne fait pas avancer le curseur. En cours d'un balayage des nouvelles conditions peuvent être créées établies ou toutes les conditions peuvent être annulées (ANNULER-CONDITIONS) sans changer la valeur du curseur. Un balayage recommence au premier tuple de la relation en employant la primitive RECOMMENCER-BALAYAGE.

En guise d'introduction à l'utilisation de MIMER, nous venons de décrire très sommairement les primitives de l'une de ses interfaces. Par la suite, Nous nous proposons d'analyser en détail la totalité de ces primitives.

3.9.2 Les interfaces d'utilisation

MIMER est manipulé par l'intermédiaire d'un langage de primitives de haut niveau. Ces primitives sont classées en trois niveaux, chacun constitue une interface différente qui correspond à un degré d'utilisation différent (Figure 3.7).

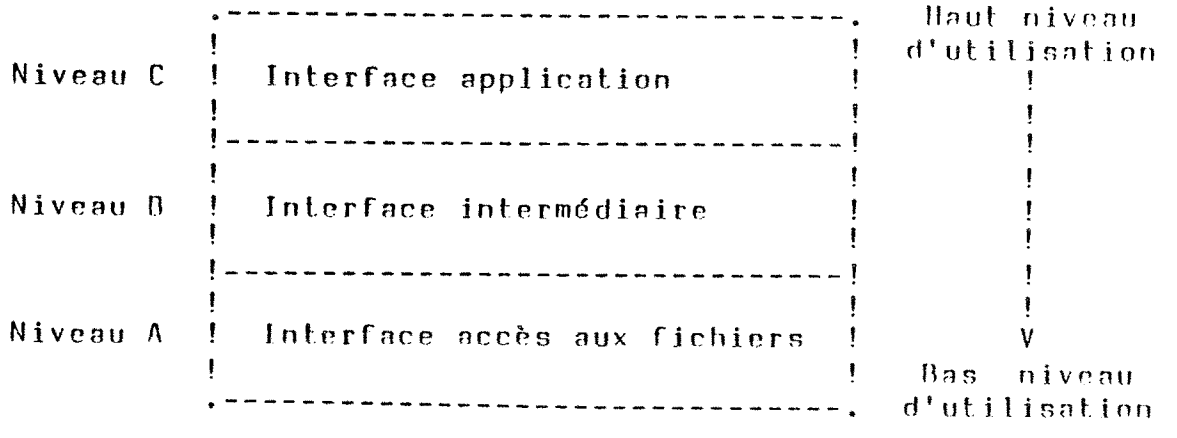


Figure 3.7

Selon le cadre d'implantation où se situe MIMER, on peut utiliser l'une des trois interfaces.

Si nous supposons qu'un SGBD est une machine relationnelle abstraite, l'interface de niveau C correspond au langage assembleur de celle-ci. C'est ainsi qu'un usager de MICROBE voulant aller au-delà des limites imposées par le langage MIQUEL aura le choix de programmer directement son application sur cette interface. Par exemple, l'interpréteur de MIQUEL a été construit sur cette interface.

Les primitives de niveau B servent à construire celles du niveau C. Elles réalisent des opérations d'accès aux tuples suivant leurs idt, de gestion de la mémoire virtuelle, de gestion de la liste d'espace disponible et de formattage de pages.

Dans un autre cadre d'utilisation de MIMER, s'il est nécessaire d'y introduire de nouvelles fonctions, on emploiera alors les interfaces B et C.

L'usage de l'interface A est exclusivement réservé à MIMER. Concrètement, elle réalise les opérations de création, d'ouverture et de fermeture du fichier allouant la Base de Données.

MIMER a été implementé en PASCAL OMSI Oregon Software Inc Version 1.1. Dans cette version de PASCAL chaque

session de mise à jour d'un fichier crée automatiquement une nouvelle copie du fichier. Certes, ceci convient à la gestion d'un certain type de fichiers, mais est inadéquat pour la gestion d'une Base de Données parce qu'une nouvelle copie de celle-ci serait créée à chaque session de mise à jour, même s'il ne s'agit que d'une modification ou d'une insertion d'un tuple.

Les primitives appartenant au niveau A ont été implémentées en langage assembleur. Ce choix a été guidé par le besoin d'éviter la création de nouvelles copies de la Base de Données. Cependant, nous considérons cette solution comme temporaire et nous espérons que des nouvelles versions de PASCAL nous permettrons d'éliminer cette contrainte, ce qui rendra MIMER complétement transportable sur d'autres machines.

On remarque que les primitives des interfaces A et B ne sont pas disponibles à l'utilisateur d'une application et qu'elles ne sont que des outils permettant l'écriture des primitives de l'interface C.

3.10 Les relations CURSEURS-ACTIFS et ARGUMENTS

Un curseur est l'idt d'un tuple de la relation interne nommée CURSEURS-ACTIFS. Ce tuple qui est en effet un descripteur d'un balayage est inséré dans la relation CURSEUR-ACTIFS à chaque création de balayage et, supprimé lors de l'exécution de la primitive FINIR-BALAYAGE.

Pour les balayages n'ayant qu'une seule condition, l'argument de recherche est mis dans l'attribut appelé argument de la relation CURSEURS-ACTIFS. Ceci permet d'accéder rapidement à ces valeurs. Les balayages à plusieurs conditions utilisent la relation ARGUMENTS pour le stockage des différents arguments. L'accès à cette relation est réussi grâce à une indirection établie depuis la relation CURSEURS-ACTIFS.

La relation CURSEURS-ACTIFS est donc, une relation interne qui permet le stockage d'informations portant sur :

.Les relations balayées,

.les conditions de balayage,

.les attributs, appelés attributs-argument, sur lesquels les conditions sont appliquées,

.l'argument de recherche.

La relation ARGUMENTS est une relation interne unaire qui contient les arguments de balayage dans le cas de conditions multiples ou d'une condition simple ayant un argument dont la longueur depasse la constante LNCHAI (cf Annexe 2). Le format exact de la relation CURSEURS-ACTIFS est:

LONGUEUR	ATTRIBUT
2	témoin d'existence du tuple
4	valeur du curseur (idt actuel du balayage)
8*MNBATT	liste d'attributs-argument : (pour la valeur de MNBATT voir

Annexe 2)

2	type du 1er. attribut-argument
2	position du 1er. attribut-argument
2	longueur du 1er. attribut-argument
2	opérateur de comparaison sur le 1er. attribut-argument
2	opérateur logique (ET, OU) liant les conditions
	. . .
2	type du MNBATT-ième attribut-argument
2	position du MNBATT-ième attribut-argument
2	longueur du MNBATT-ième attribut-argument
2	opérateur de comparaison sur le MNBATT-ième attribut-argument
2	opérateur logique (ET, OU) liant les conditions
4	nom interne de la relation balayée
2	longueur d'un tuple de la relation balayée
4	nom interne de la relation inverse (pour le balayage indirect)
4+2	pointeur de balayage de la relation inverse (pour le balayage indirect)
2	marque d'indirection
LNCHAI	argument de recherche (numérique ou chaîne de caractères). Pour la valeur de LNCHAI voir Annexe 2.

La liste d'attributs-arguments est ordonnée et termine avec le premier attribut ayant la valeur NUL.

La marque d'indirection peut avoir les valeurs suivantes :

.'00' : l'attribut-argument contient la valeur d'un argument numérique ou alphanumérique.

.'11' : l'attribut-argument contient un pointeur vers un tuple de la relation ARGUMENTS.

4 DESCRIPTION DES INTERFACES D'UTILISATION

Nous nous proposons d'étudier maintenant, d'une façon détaillée le langage de primitives appartenant à chacune de trois interfaces définies dans le chapitre précedent. L'interface d'application sera illustrée par des exemples dans §4.1.7.

La lecture de §3.9.2 est nécessaire à la compréhension du présent chapitre. L'utilisateur de l'une des interfaces devra utiliser par ailleurs: L'Annexe 2 qui présente les paramètres d'implémentation de MIMER et les Annexes 3 et 4 qui montrent la syntaxe de chaque primitive exprimée en langage PASCAL. Par facilité d'emploi, un code mnémonique substitue le nom de chaque primitive, l'Annexe 4 présente aussi, une liste exhaustive de ces codes.

Voyons tout d'abord, quelques définitions et conventions qui permettrons de mieux comprendre cette description:

Un nom externe est un identificateur alphanumérique qui désigne d'une façon unique une relation ou un attribut à l'interieur d'une relation. Dans notre réalisation il correspond à la valeur d'une variable de type EXNAME, telle qu'elle est définie dans l'Annexe 3.

Un nom interne est l'idt d'un tuple dans la relation MAITRE ou dans la relation ATTRIBUTS.

Nous avons introduit la notion de nom interne visible aux utilisateurs des interfaces car, cela permet d'accélerer l'accès aux tuples des catalogues.

Conventions:

-Les noms externes de relations seront notés R1,R2,...,Rn.

-Les noms internes de relations seront notés IR1,IR2,..,IRn.

-Les noms externes d'attributs seront notés A1,A2,...,An.

-Les noms internes d'attributs seront notés IA1,IA2,...,IAn.

Chaque primitive a des paramètres d'entrée et de sortie. En particulier, elles ont toutes un paramètre de sortie, noté CE (Code d'Erreur), qui représente les anomalies d'exécution. Une liste exhaustive de celles-ci est preséntée dans l'Annexe 6.

Enfin, nous recommandons au lecteur qui se place au niveau utilisateur d'une application de se dispenser de la lecture de §4.2 et §4.3. Il trouvera par contre dans §4.1 un manuel d'utilisation de MIMER.

4.1 L'interface utilisateur d'une application (Niveau C)

4.1.1 Primitives portant sur l'ensemble de la Base de Données

C1) CREER-BASE (<nom-externe-de-la-base>,
<administrateur-de-la-base>, <support-magnétique>,
<code-d'exploitation>, <taille-base>, CE);

-Paramètres en entrée : <nom-externe-de-la-base>,
<administrateur-de-la-base>, <support-magnétique>,
<code-d'exploitation>, <taille-base>
Paramètres en sortie : CE
-Le résultat de cette primitive est la création d'une Base de Données vide. Cela veut dire création de :
i) un fichier allouant la Base de Données,
ii) le descripteur de la Base de Données,
iii) la liste d'espace disponible et
iv) le système de catalogues.
-La Base de Données créée aura un nombre de pages égal à <taille-base> sur le périphérique identifié par <support-magnétique>. Elle sera appelée <nom-externe-de-la-base>, de même qu'elle sera propriété d'un utilisateur identifié par <code-d'exploitation> et <administrateur-de-la-base>.

C2) DETRUIRE-BASE (<nom-externe-de-la-base>,
<administrateur-de-la-base>, <support-magnétique>,
<code-d'exploitation>, CE) ;

-Paramètres en entrée : <nom-externe-de-la-base>, <administrateur-de-la-base>, <support-magnétique>, <code-d'exploitation>

Paramètres en sortie : CE

-La Base de Données appelée <nom-externe-de-la-base> stockée sur <support-magnétique> et appartenant à un utilisateur identifié par un <code-d'exploitation>, est détruite, à condition que le mot de passe <administrateur-de-la-base> coincide avec celui stocké dans la Base de Données.

-Cette primitive suppose que la Base de Données a été ouverte au préalable.

C3) OUVRIR-BASE (<nom-externe-de-la-base>, <nom-utilisateur>, <support-magnétique>, <code-d'exploitation> , CE) ;

-Paramètres en entrée : <nom-externe-de-la-base>, <utilisateur>, <support-magnétique>, <code-d'exploitation>

Paramètres en sortie : CE

-Celle-ci correspond à l'ouverture d'une Base de Données identifiée par <nom-externe-de-la-base> et stockée sur <support-magnétique>.

-<nom-utilisateur> et <code-d'exploitation> constituent l'identification de l'utilisateur qui veut se servir de la Base de Données.

C4) FERMER-BASE ;

-Celle-ci ferme une Base de Données qui a dû être ouverte au préalable.

C5) OBTENIR-CARACTERISTIQUES-BASE(<dimension>, <tableau>, CE);

-Paramètres en entrée: -
paramètres en sortie: <dimension>, <tableau>, CE.
-Il est possible d'obtenir, par l'intermédiaire de cette primitive, une liste de toutes les relations de la Base de Données et de leurs caractéristiques associées.
-La liste est transmise dans <tableau> et celui-ci est structuré ainsi:

	noms externes des relations	noms internes des relations	type	cardinalité	degré	longueur d'un tuple	type de stockage
1							
2							
. .							
<dimension>							

4.1.2 Primitives portant sur les relations de base

C6) CREER-RELATION (R1, IR1, <type-de-relation>, CE) ;

-Paramètres en entrée : R1, <type-de-relation>
Paramètres en sortie : IR1, CE
-Une relation vide est créée. A partir d'un nom externe R1, le système attribue un nom interne IR1 à la nouvelle relation. Si le nom R1 existe déjà dans la Base de Données, l'opération de création est refusée.
-Dans la pratique cette primitive correspond à l'insertion d'un tuple (descripteur de R1) dans la relation MAITRE.
-<type-de-relation> peut être B (Base), I (Inversée), T (inTerne); cf §3.2.

C7) DETRUIRE-RELATION (IR1, CE);

-Paramètres en entrée: IR1
paramètres en sortie: CE
-IR1 est le nom interne d'une relation de base ou inverse.
-Dans le cas où IR1 est une relation de base les index construits sur IR1 sont détruits.
NB: Il reste à étudier la primitive du point de vue protection.

C8) VIDER-RELATION (IR1, CE) ;

-Paramètres en entrée : IR1

Paramètres en sortie : CE

-Celle-ci libère l'espace occupé par les tuples de la relation IR1. Cependant, la description de la relation demeure dans les catalogues, toutefois avec une cardinalité égale à 0.

-Les index qui ont été construits sur IR1 sont détruits.

C9) DEFINIR-STOCKAGE (IR1, <type>, CE);

-Paramètres en entrée: IR1, <type>

paramètres en sortie: CE

-Cela permet d'associer un <type> de stockage à une relation de base. <type> peut avoir deux valeurs: Séquentiel (SSE) et Hachage Virtuel Linéaire (HVL). D'autres possibilités seront introduites ultérieurement.

C10) OBTENIR-CARACTERISTIQUES-RELATION (R1, IR1, <type>, <cardinalité>, <degré>, <longueur-d'un-tuple>, <type-de-stockage>, CE) ;

-Paramètres en entrée : R1

Paramètres en sortie : IR1, <type>, <cardinalité>, <degré>, <longueur-d'un-tuple>, <type-de-stockage>, CE

C11) GROUPER-RELATION (IR1, <ordonnancement>, <liste-d'attributs>, CE) ;

-Paramètres en entrée: IR1, <ordonnancement>,

<liste-d'attributs>

paramètres en sortie: CE

-La primitive INSERER-TUPLE exécutée sur une relation ayant SSE comme type de stockage insère des tuples séquentiellement et demande l'allocation de nouvelles pages, si nécessaire. La primitive DETRUIRE-TUPLE détruit des tuples et laisse des "trous" dans les pages; c'est-à-dire, qu'elle ne réorganise pas l'espace de la relation. Ces "trous" peuvent être supprimés par l'exécution de la primitive GROUPER-RELATION.

-Cela permet donc, de grouper une relation en la triant en même temps, selon les attributs IA1,IA2,..,IAn de la <liste-d'attributs>.

-Le groupement n'est pas maintenu; c'est ainsi qu'une relation restera complètement groupée jusqu'à ce que des nouvelles insertions ou destructions de tuples ou groupements (sur d'autres attributs) se produisent.

-Puisque ici, grouper est aussi synonyme de trier, <ordonnancement> peut être 'A' (tri Ascendant) ou 'D' (tri Descendant).

-Elle ne peut être appliquée que sur des relations ayant SSE comme type de stockage (cf primitive DEFINIR-STOCKAGE).

4.1.3 Primitives portant sur les attributs des relations

C12) CREER-ATTRIBUT (IR1, A1, IA1, <type-d'attribut>, <longueur-de-l'attribut>, CE);

-Paramètres en entrée: IR1, A1, <type-de-l'attribut>, <longueur-de-l'attribut>, CE

Paramètres en sortie: IA1, CE

-L'attribut A1 est ajouté à la relation de base IR1. La primitive donne en réponse le nom interne de A1, soit IA1.

-Si IR1 n'est pas vide, cette primitive fait une réorganisation de la relation.

-<type-de-l'attribut> peut être: E (Entier), R (Réel), C (Caractère).

-Enfin, chaque tuple aura un nouveau champ qui est initialisé à la valeur indéfinie (son équivalent selon le type qu'il comporte).

C13) DETRUIRE-ATTRIBUT (IR1, IA1, CE);

-Paramètres en entrée: IR1, IA1

paramètres en sortie: CE

-L'attribut IA1 est supprimé de la relation IR1.

-Si IR1 n'est pas vide, cette primitive fait aussi une réorganisation de la relation. C'est-à-dire la valeur de l'attribut dans chaque tuple est supprimé.

-Les attributs faisant partie de clés des index, ne peuvent pas être détruits par ce moyen.

C14) OBTENIR-CARACTERISTIQUES-ATTRIBUT (IR1, A1, IA1, <type-attribut>, <longueur-attribut>, <position-attribut>, CE) ;

-Paramètres en entrée : IR1, A1

Paramètres en sortie : IA1, <type-attribut>, <longueur-attribut>, <position-attribut>, CE

C15) OBTENIR-CARACTERISTIQUES-ATTRIBUTS (IR1, <tableau>, CE);

-Paramètres en entrée : IR1

Paramètres en sortie : <tableau>, CE

-Cette primitive donne en résultat un <tableau> dont le nombre de lignes est le degré de la relation IR1. Celui-ci représente une liste d'attributs de la relation IR1 et de leurs caractéristiques. Ce <tableau> est structuré ainsi :

	Nom externe de l'attribut	Nom interne de l'attribut	type de l'attribut	longueur de l'attribut	position de l'attribut dans le tuple
1					
2					
. . .					
<degré relation>					

4.1.4 Primitives de balayage des relations

Un balayage est un parcours établie sur une relation et qui permet d'accéder à ses tuples un par un. Durant ce balayage les tuples peuvent être modifiés, détruits ou simplement consultés. En plus, le parcours peut être sélectif en ce sens qu'on n'accède qu'à un certain nombre de tuples répondant à un ensemble de conditions. De cette

façon MIMER est vue comme une mémoire associative qui fait l'adressage de tuples par leurs valeurs, plutôt que par leurs positions dans les relations.

CURSEUR est le nom interne d'un tuple descripteur de balayage, dans la relation CURSEURS-ACTIFS (cf §3.10). Ce tuple stocke, entre autres, la valeur d'un pointeur de parcours de la relation balayée. Par la suite nous appelerons cette valeur: idt actuel du balayage et le tuple pointé par cette valeur: tuple actuel du balayage. Dans ce qui suit nous employerons largement cette terminologie.

L'idt actuel du balayage ne pourra être modifié qu'indirectement par l'intermédiaire des primitives de balayage.

L'ensemble de primitives décrites ici assurent la gestion de balayages et de tuples des relations balayées.

C16) CREER-BALAYAGE (IR1, CURSEUR, CE) ;

-Paramètres en entrée : IR1
Paramètres en sortie : CURSEUR, CE
-Cette primitive présente deux modes d'utilisation :
i) mode balayage simple : une relation est balayée séquentiellement telle qu'elle est stockée,
ii) mode balayage indirect : c'est le cas où l'on balaye une relation, dite mère, tout en suivant une

relation inverse qui pointe sur la première. Ceci permet d'accélérer la recherche des tuples et, de parcourir la relation mère selon un ordre établi par la relation inverse.

-IR1 peut être :

a) une relation interne,

b) une relation de Base,

c) une relation inverse.

Dans les cas a) et b) un balayage simple est créé, dans le cas c) un balayage indirect est créé sur la relation mère à IR1.

-Un tuple ayant CURSEUR comme idt est inséré dans la relation CURSEURS-ACTIFS.

-En bref, cette primitive permet de créer un balayage (éventuellement conditionné par l'emploi de la primitive CONDITION-BALAYAGE, comme on verra plus tard) sur une relation mère soit, d'une façon directe (balayage simple) soit, à l'aide d'une relation inverse qui pointe sur la relation mère (balayage indirect).

C17) CONDITION-DE-BALAYAGE (CURSEUR, <opérateur-booléen>, <attribut-argument>, <opérateur-de-comparaison>, <argument>, CE) ;

-Paramètres en entrée : CURSEUR, <opérateur-booléen>, <attribut-argument>, <opérateur-de-comparaison>, <argument>

Paramètres en sortie : CE

-Cette primitive permet d'établir une liste de conditions

de balayage sur une relation (nous parlerons dans ce cas là d'un balayage conditionnel). A chaque appel de la primitive, une nouvelle condition est ajoutée , celle-ci est reliée à la liste par l'intermédiaire d'un <opérateur-booléen> qui peut prendre les valeurs OU et ET. Evidemment, cet opérateur n'est pas pris en compte lors de l'établissement de la première condition.

-Plus tard, nous verrons que la liste de conditions peut être changée complètement. Il faut annuler les conditions actuelles, et en établir d'autres nouvelles (cf primitive ANNULER-CONDITIONS).

-<argument> est une valeur qui constitue un argument de recherche sur la relation balayée. Il n'est pas pris en compte si l'<opérateur-de-comparaison> est NL.

-<opérateur-de-comparaison> peut avoir les valeurs suivantes:

.EG (Egal),

.NE (Non Egal),

.SP (SuPerieur),

.SE (Superieur ou Egal),

.IF (InFerieur),

.IE (Inferieur ou Egal),

.NL (NuL).

-<attribut-argument> est le nom interne d'un attribut de la relation balayée sur lequel l'<argument> et l'<opérateur-de-comparaison> sont appliqués.

-La condition <attribut-argument>NL<argument> provoque un balayage inconditionnel. Le paramètre <argument> n'est

là que pour compatibilité avec la syntaxe de la condition. Il n'est pas pris en compte. De la même manière un balayage inconditionnel est exécuté quand on ne fait pas appel à la primitive CONDITION-DE-BALAYAGE.
-Une séquence, consistant en N appels à cette primitive créera un balayage conditionné par :

opb1 C1 opb2 C2 opb3 C3 ... Cn-1 opbn Cn

où Ci est une condition établie du ième appel et
opbi est OU ou ET
opb1 n'est jamais pris en compte.

MIMER évalue la condition de gauche à droite, donc en fait la condition résultante peut être vue ainsi :

((...((C1 opb2 C2) opb3 C3)...)Cn-1 opbn Cn)

L'utilisateur devra être spécialement soigneux lors de l'établissement d'une telle séquence car, elle determine l'ordre d'évaluation des conditions. En outre, cette séquence n'est pas toujours commutative.

C18) ANNULER-CONDITIONS-DE-BALAYAGE (CURSEUR, CE) ;

-Paramètres en entrée : CURSEUR
Paramètres en sortie : CE

-Celle-ci permet de revenir au balayage inconditionnel au cours d'un balayage conditionnel. Après cet appel le balayage continue mais, sans aucune condition; bien entendu, des nouvelles conditions pourront être

établies.

-On remarque que l'idt actuel du balayage n'est pas modifié.

C19) TUPLE-SUIVANT (CURSEUR, <tuple>, CE) ;

-Paramètres en entrée : CURSEUR

Paramètres en sortie : <tuple>, CE

-Celle-ci modifie l'idt actuel du balayage. Elle met le tuple actuel du balayage dans le tampon <tuple>.

-Avant l'appel à cette primitive, il est nécessaire de créer un balayage.

-Cette primitive doit être utilisée en combinaison avec la primitive FIN-RELATION qui est présentée ci-après. Une demande de tuple après la fin de la relation produira une erreur. Le code d'erreur correspondante est mis dans CE.

C20) FIN-RELATION (CURSEUR, CE) ;

-Paramètres en entrée : CURSEUR

Paramètres en sortie : FIN-RELATION, CE

-C'est une fonction booléenne delivrant la valeur vraie si la fin de la relation est atteinte et fausse sinon.

C21) RECOMMENCER-BALAYAGE (CURSEUR, CE) ;

-Paramètres en entrée : CURSEUR

Paramètres en sortie : CE

-Le balayage recommencera au début de la relation. Les conditions de balayage restent inchangées.

C22) DETRUIRE-TUPLE (CURSEUR, CE) ;

-Paramètres en entrée : CURSEUR
Paramètres en sortie : CE
-Le tuple actuel du balayage est détruit et un trou sera créé dans la relation correspondante. Ceci génère aussi, automatiquement une destruction des tuples dans les index pointant sur le tuple détruit.

C23) REMPLACER-ATTRIBUT (CURSEUR, IA1, <nouvelle-valeur>, CE);

-Paramètres en entrée : CURSEUR, IA1, <nouvelle-valeur>
Paramètres en sortie : CE
-IA1 est le nom interne de l'attribut du tuple actuel du balayage que l'on veut remplacer. <nouvelle-valeur> substitue la valeur précédemment stockée. IA1 ne peut faire partie d'aucune clé d'index.
-Le seul moyen de changer les valeurs d'attributs-clé est en faisant la séquence : DETRUIRE-TUPLE; INSERER-TUPLE.

C24) COMP (CURSEUR, <attribut-argument>, <opérateur-de-comparaison>, <argument>, CE) ;

-Paramètres en entrée : CURSEUR, <attribut-argument>, <opérateur-de-comparaison>, <argument>

Paramètres en sortie : COMP, CE

-Ici, <attribut-argument> <opérateur-de-comparaison> <argument> constitue une condition similaire à celle de la primitive CONDITION-BALAYAGE.

-COMP est une fonction booléenne qui delivre la valeur vraie si la condition est satisfaite par le tuple actuel, et fausse sinon.

-L'appel à COMP ne modifie pas l'idt actuel du balayage.

-Pour résumer, cette primitive, compare la valeur de <attribut-argument> avec la valeur stockée dans <argument>.

C25) FIN-BALAYAGE (CURSEUR, CE);

-Paramètres en entrée: CURSEUR

paramètres en sortie: CE

-Celle-ci permet d'annuler le balayage identifié par CURSEUR et de libérer la place correspondante dans la relation CURSEURS-ACTIFS.

4.1.5 Primitives portant sur les relations inverses

C26) CREER-RELATION-INVERSE(IR1, RX, IR2, <ordonnancement>, <type-d'index>, <liste-d'attributs>, CE);

-Paramètres en entrée: IR1, <ordonnancement>, <type-d'index>, RX, <liste-d'attributs>

paramètres en sortie: IR2, CE

-RX est le nom externe de la relation inverse que l'on

veut créer.

-L'<ordonnancement> peut être soit A (Ascendant), soit D (Descendant).

-IR1 est le nom de la relation sur laquelle on veut bâtir une relation inverse (IR2). La clé de IR2 est donnée par la concatenation de valeurs des attributs IA1,IA2,...,IAn appartenant à la <liste-d'attributs>.

-<type-d'index> peut prendre la valeur P dans le cas d'un index Primaire. L'insertion des tuples dont la clé existe déjà, sera rejetée. De même, une valeur de S indiquera qu'il s'agit d'un index Secondaire et, en conséquence des clés repetées seront admises.

C27) OBTENIR-CARACTERISTIQUES-INDEX (IR1, <nombre d'index>, <tableau>, CE);

-Paramètres en entrée: IR1

paramètres en sortie: <nombre d'index>, <tableau>, CE

-Cette primitive donne la liste des index existant sur la relation IR1 et de leurs caractéristiques.

-Le <tableau> résultant est structuré ainsi:

	Nom externe de l'index	nom interne de l'index	type d'index	type d' ordonnan-cement	nombre d'attributs clé	attributs clé (noms externes)		
1							..	
2							..	
:								
<nombre d'index>							..	

4.1.6 Autres primitives

C28) INITIAL ;

-Paramètres en entrée : -

Paramètres en sortie : -

-Cette primitive initialise le système de pagination. Elle doit être exécutée avant le premier appel à MIMER.

C29) INSERER-TUPLE (IR1, <tuple>, CE);

-Paramètres en entrée: IR1, <tuple>

paramètres en sortie: CE

-<tuple> est ajouté à la relation IR1, à condition que les valeurs des attributs correspondants aux clés d'index primaires n'existent pas déjà. C'est-à-dire l'unicité des clés des index primaires est préservée.

-Cette primitive insère systématiquement un tuple dans ses inverses.

-IR1 doit être une relation de base.

-Les insertions sont faites dans la dernière page de la relation ou dans une nouvelle si celle-là est déjà pleine. En conséquence le groupement initial n'est pas garanti.

4.1.7 Quelques exemples d'utilisation des primitives de Niveau C

Par des raisons de clarté et de simplicité, les

exemples sont présentés avec quelques variations, par rapport à la syntaxe des primitives décrites précédemment. Sauf mention contraire les exemples sont independants les uns des autres.

Soit la relation de base R1

idt	A	B	C
t1	α	7	a
t2	α	4	z
t3	β	4	a
t4	θ	4	x
t5	β	9	b
t6	θ	1	x

Soit la relation de base R4

idt	D	E
t7	32.4	b
t8	27.7	d
t9	40.5	a
t10	53.1	x

Un index Primaire R1BC

idt	X	Y
t11	1x	t6
t12	4a	t3
t13	4x	t4
t14	4z	t2
t15	7a	t1
t16	9b	t5

Un index secondaire R1C

idt	Z	T
t17	z	t2
t18	x	t4
t19	x	t6
t20	b	t5
t21	a	t3
t22	a	t1

EXEMPLE 0:

CREER-RELATION-INVERSE(IR1, 'R1BC', IR2, 'A', 'P', IB, IC, CE);

EXEMPLE 1:

CREER-RELATION-INVERSE(IR1, 'R1C', IR2, 'D', 'S', IC, CE);

EXEMPLE 2:

co détruire les tuples de la relation R1 ayant A=9 (balayage simple) co

CREER-BALAYAGE(IR1, CRS2, CE);

```
CD ( CRS2, 'OU', IA, 'EG', '0', CE ) ;
tantque not FR (CRS2, CE) faire
     TUPLE-SUIVANT (CRS2, TAMPON, CE ) ;
     DETRUIRE-TUPLE(CRS2, CE);
finfaire
FIN-BALAYAGE(CRS2, CE);
```

EXEMPLE 3:

co détruire tous les tuples de la relation R1 ayant B=4 et C=a (il n'y en aura qu'un, car BC est une clé primaire. Puisqu'il existe une inverse sur les attributs B et C, on y fera un balayage indirect co

```
CREER-BALAYAGE( IR1BC, CRS3, CE);
CREER-CONDITION (CRS3, 'OU', IX, 'EG', '4a', CE);
TUPLE-SUIVANT(CRS3, TAMPON, CE);
DETRUIRE-TUPLE(CRS3, CE);
FIN-BALAYAGE(CRS3, CE);
```

EXEMPLE 4:

co soient les relations R2 (D, E, F) et R3 (A, G) on veut faire le join de R2 et R3 avec E = A. Ceci equivaut dans le langage algebrique à join (R2,R3, E = A); co

```
CREER-BALAYAGE(IR2, CRS4, CE);
CREER-BALAYAGE(IR3, CRS, CE);
TUPLE-SUIVANT(CRS4, TUPR1, CE);
```

```
tantque not FIN-RELATION(CRS4, CE) faire
    ARG := extraire la valeur de l'attribut E du
    TUPR1;
    CD(CRS, 'OU', IA, 'EG', ARG, CE); co IA est
    le nom interne de l'attribut A de la relation
    R2 co
    TUPLE-SUIVANT(CRS, TUPR2, CE);
    tantque not FIN-RELATION(CRS, CE) faire
                écrire TUPR1, TUPR2
                TUPLE-SUIVANT(CRS, TUPR2, CE);
    finfaire
    TUPLE-SUIVANT(CRS4, TUPR1, CE);
    ANNULER-CONDITIONS(CRS, CE);
    RECOMMENCER-BALAYAGE(CRS, CE)
finfaire
FIN-BALAYAGE(CRS4, CE);
FIN-BALAYAGE(CRS, CE);
```

EXEMPLE 5:

```
co dans l'attribut D de R4 remplacer les valeurs
supérieures à 45.3 et inférieures à 67.4 par 50.0
co
CREER-BALAYAGE(IR4, CRS5, CE);
CREER-CONDITION(CRS5, 'OU', ID, 'SP','45.3', CE);
CREER-CONDITION(CRS5, 'ET', ID, 'IF','67.4', CE);
TUPLE-SUIVANT(CRS5, BIDON, CE);
tantque not FIN-RELATION (CRS5, CE) faire
    REMPLACER-ATTRIBUT(CRS5, ID, '50.0', CE);
```

```
        TUPLE-SUIVANT(CRS5, BIDON, CE)
    finfaire
    FIN-BALAYAGE(CRS5, CE);
```

EXEMPLE 6:

co lister les tuples de la relation R1 tels que (A='beta') ou (3<=B<=5). Ceci équivaut dans le langage algebrique à select(R1,(A='beta') ou (3<=B<=5)) co

```
CREER-BALAYAGE(IR1, CRS6, CE);
CONDITION-BALAYAGE(CRS6, 'OU', IB, 'SE', '3', CE);
CONDITION-BALAYAGE(CRS6, 'ET', IB, 'IE', '5', CE);
CONDITION-BALAYAGE(CRS6, 'OU', IA, 'EG', 'beta',
CE);
TUPLE-SUIVANT(CRS6, TAMPON, CE);
tantque not FIN-RELATION(CRS6, CE) faire
    écrire TAMPON
    TUPLE-SUIVANT(CRS6, CE)
finfaire
FIN-BALAYAGE(CRS6, CE);
```

On remarque que la séquence suivante, est incorrecte car elle représente (cf §4.1.4) ((A='beta')ou(3<=B))et(B<=5) :

```
CONDITION-BALAYAGE(CRS6,'OU', IA, 'EG', 'beta',
CE);
CONDITION-BALAYAGE(CRS6, 'OU', IB, 'SE', '3', CE);
CONDITION-BALAYAGE(CRS6, 'ET', IB, 'IE', '5', CE);
```

4.2 L'interface intermédiaire (niveau B)

Nous rappelons que les primitives ci-dessous ne sont pas disponibles à l'utilisateur d'une application. Elles ne sont que des primitives de service de l'interface C (cf §3.9.2).

4.2.1 Pour l'accès aux tuples

B1) LIRE-TUPLE (IDT, <longueur-tuple>, TUPLE, CE) ;

-Paramètres en entrée : IDT, <longueur-tuple>
paramètres en sortie : TUPLE, CE
-Le tuple dont l'identificateur de tuple est IDT est extrait de la page correspondante, puis il est transmis dans la zone TUPLE.

B2) ECRIRE-TUPLE (IDT, <longueur-tuple>, TUPLE, CE) ;

-Paramètres en entrée : IDT, <longueur-tuple>, TUPLE
paramètres en sortie : CE
-Le tuple stocké dans la zone TUPLE est inséré dans une page, à la position indiquée par l'identificateur de tuple IDT.

B3) TROUVER-PLACE-POUR-TUPLE (IR1, IDT, CE) ;

-Paramètres en entrée : IR1
paramètres en sortie : IDT, CE

-Celle-ci trouve "un trou" dans la dernière page de la relation IR1, s'il n'y en pas ou la relation n'a aucun tuple encore, une nouvelle page est donc, allouée à la relation.

-IDT est l'identificateur de tuple du trou.

-Cette primitive précède l'insertion d'un tuple dans une relation.

4.2.2 Pour la gestion de la mémoire virtuelle

B4) LIRE-PAGE (<numéro-page>, <numéro-buffer>, CE) ;

-Paramètres en entrée : <numéro-page>

paramètres en sortie : <numéro-buffer>, CE

-La page dont l'adresse est <numéro-page> est amenée en mémoire centrale, si elle n'est pas y présente.

-<numéro-buffer> est l'identification du buffer où la page a été allouée.

-Le remplacement de pages est géré par une politique LRU (cf §3.8)

4.2.3 Pour la gestion de la liste d'espace disponible

B5) DEMANDER-PAGE-LIBRE (<numéro-page>, CE) ;

-Paramètres en entrée : -

paramètres en sortie : <numéro-page>, CE

-Une page est déliée de la liste d'espace disponible et rendue à la primitive appelante. <numéro-page> est

l'adresse de cette page.

B6) LIBERER-PAGE (<numéro-page>, CE) ;

-Paramètres en entrée : <numéro-page>

paramètres en sortie : CE

-La page d'adresse <numéro-page> est insérée à la tête de la liste d'espace disponible.

4.2.4 Pour le formatage des pages

B7) FORMATER-PAGE (<numéro-page>, M, CE) ;

-paramètres en entrée : <numéro-page>, M

paramètres en sortie : CE

-M est un tuple descripteur d'une relation, c'est-à-dire un tuple de la relation MAITRE ou ayant le même format.

-La page d'adresse <numéro-page> est formatée selon les caractéristiques décrites par M. En outre, les témoins d'existence des tuples sont initialisés à '00'.

4.3 L'interface d'accès aux fichiers (Niveau A)

Nous rappelons que les primitives ci-dessous sont dépendantes de la machine sur laquelle MIMER tourne. Cette interface est la seule à modifier lors d'un transport sur d'autres machines.

4.3.1 Pour la création, ouverture et fermeture du fichier allouant la Base de Données

Dans notre implémentation <nom-fichier>, <support-magnétique> et <code-d'exploitation> correspondent respectivement au nom d'un fichier, au nom logique d'un périphérique d'accès direct et à l'UIC (User Identification Code) selon le système d'exploitation RSX-11M <RSX11>.

A1) CREER-FICHIER (<nom-fichier>, <support-magnétique>, <code-d'exploitation>, <taille-fichier>, CE);

-Paramètres en entrée : <nom-fichier>, <support-magnétique>, <code-d'exploitation>, <taille-fichier>
paramètres en sortie : CE
-Un fichier appelé <nom-fichier> est créé sous la propriété de <code-d'exploitation>. Un nombre de blocs égal à <taille-fichier> est alloué sur <support-magnétique>.

A2) DETRUIRE-FICHIER ;

-Paramètres en entrée : -
paramètres en sortie : -
-Celle-ci détruit un fichier qui avait été ouvert ou créé au préalable.

A3) OUVRIR-FICHIER (<nom-fichier>, <support-magnétique>,

<code-d'exploitation>, CE);

-Paramètres en entrée : <nom-fichier>,
<support-magnétique>, <code-d'exploitation>
paramètres en sortie : CE
-Le fichier <nom-fichier> appartenant à
<code-d'exploitation> est ouvert depuis
<support-magnétique>.

A4) FERMER-FICHIER ;

-Paramètres en entrée : -
paramètres en sortie : -
-Celle-ci ferme le fichier qui avait été ouvert au préalable.

4.3.2 Pour l'accès direct aux blocs

A5) LIRE-BLOC (BUFFER, CLEBLOC) ;

-Paramètres en entrée : CLEBLOC
paramètres en sortie : BUFFER
-Le bloc physique dont l'adresse dans la mémoire secondaire est CLEBLOC est lu et placé dans la zone BUFFER en mémoire centrale.

A6) ECRIRE-BLOC (BUFFER, CLEBLOC);

-Paramètres entrée : CLEBLOC, BUFFER

paramètres en sortie : -

-La page placée dans la zone BUFFER en mémoire centrale est recopiée en mémoire secondaire à l'adresse CLEBLOC.

5 LES METHODES D'ACCES DANS MIMER

5.1 Introduction

Deux méthodes d'accès on été étudiées et implantées dans MIMER: la méthode séquentielle (cf §5.2) et les B*-arbres (cf §5.3). C'est surtout cette dernière qui fera l'objet au long des paragraphes suivants, d'une description détaillée. En fin de chapitre (cf §5.4) nous discuterons brièvement la méthode d'accès par des fonctions de hachage qui n'existe pas encore dans MIMER. Il serait très intéressant de les y incorporer, semble-t-il, vu ses très grandes performances d'accès.

5.2 La Méthode d'accès Séquentielle

Nous avons déjà vu qu'une relation est stockée comme une suite séquentielle de tuples dont chacun précédé par un témoin d'existence. Un accès séquentiel sur une relation (dans l'interface C) consisterait-il en un parcours de la relation telle que'elle stockée physiquement au moyen d'un balayage simple (cf §4.1.4).

Ce type d'accès est conseillé quand ne veut pas imposer un ordre logique quelconque sur l'accès aux tuples et qu'on veuille parcourir la relation toute entière; il est fortement deconseillé quand il s'agit d'un accès selectif défini par des prédicats, nous le verrons dans le paragraphe suivant.

5.3 Les B*-arbres

Les paragraphes suivants sont consacrés à l'étude de la méthode d'accès B*-arbres <FER81c> qui permet d'accéder à un tuple d'une relation dont on connaît la clé ou, d'accéder à tous les tuples d'une relation selon un ordre différent de celui dans lequel la relation est physiquement stockée. En outre, nous étudierons le coût d'accès à un tuple d'une relation et nous présenterons les B*-arbres comme ils sont implémentés dans MIMER.

Nous verrons qu'un B*-arbre est stocké sous la forme de deux relations qui sont traitées comme n'importe quelle relation de la Base de Données. Ceci est très intéressant car, l'adressage de leurs tuples n'est autre que celui déjà existant dans MIMER; ce qui facilite la tâche d'implémentation et n'augmente pas l'encombrement de mémoire MIMER.

5.3.1 Définition d'un B*-arbre.

Bayer et McCreight <BAY72> ont mis au point en 1972 une structure d'accès aux données stockées en mémoire secondaire, qu'ils ont baptisée B-arbre. L'origine du B est assez incertaine, cela peut provenir de "Balanced", "Broad", "Bushy", "Boeing", ou même "Bayer".

Ultérieurement, la définition B-arbre a subi plusieurs modifications et une diversité de B-arbres sont apparus

<COM79>, <HEL78>, <BAY77>, <WED74>, <BAY76>, <WIR76>. Parmi, la "forêt" des B-arbres existant à l'heure actuelle, nous retenons une "varieté" appelée B*-arbres dont nous donnons tout de suite la définition:

Soit d et p des nombres entiers positifs; un B*-arbre d'ordre d et de profondeur p est défini comme une arborescence ayant les proprietés suivantes:

i) Chaque noeud a au plus d fils,

ii) Chaque noeud, excepté la racine et les feuilles, a au moins $\lceil d/2 \rceil$ fils,

iii) La racine a au moins deux fils (à moins qu'elle ne soit une feuille),

iv) Toutes les feuilles apparaissent au même niveau (soit p ce niveau),

v) Un noeud ayant k fils ($k \leq d$) contient k-1 clés.

A ces proprietés se rajoute le fait que les données (les tuples de relations de base dans notre cas) sont rangées uniquement dans les feuilles. Les noeuds non-feuilles ne contiennent que des clés de tuples et des pointeurs vers les autres noeuds de l'arbre.

Chaque noeud non-feuille a un format fixe de la forme (P0, K1, P1, ..., Kd-1, Pd-1) où Pi ($0 \leqslant i \leqslant d-1$) représente un pointeur vers un autre noeud du B*-arbre et, Ki ($1 \leqslant i \leqslant d-1$) représente la clé d'un tuple d'une relation de base.

Les clés dans un noeud sont ordonnées; ceci veut dire que $Ki \lesssim Kj$ $(1 \leqslant i,j \leqslant d-1)$; $\lesssim$ étant un ordre total fixé par l'utilisateur, lors de l'appel de la primitive MIMER: CREER-RELATION-INVERSE (cf §4.1.5). La Figure 5.1 illustre le format d'un noeud non-feuille.

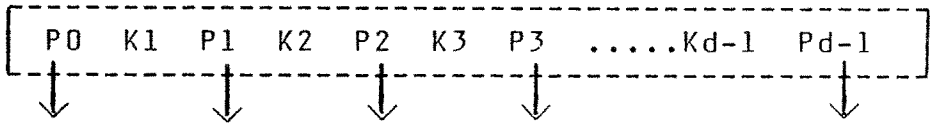


Figure 5.1

Un B*-arbre de MIMER a deux composants (Figure 5.2):

i) Un B*-index, constitué par l'ensemble de noeuds non-feuilles de et,

ii) Une relation de base dont les tuples sont les noeuds feuilles.

La proprieté iv) ci-dessus garantit que l'arbre est toujours équilibré et en conséquence le nombre d'accès necéssaires pour accéder à n'importe quel tuple est le même. En outre, de la proprieté ii) ci-dessus on peut déduire qu'un noeud non-feuille est toujours rempli au moins à moitié; nous dirons que le taux de remplissage d'un noeud est de 50%, ce qui est une "bonne" performance d'utilisation des noeuds. On verra que des opérations d'insertion ou de suppression de clés doivent preserver ce taux de remplissage ainsi que l'équilibre du B*-arbre.

Le taux de remplissage d'un B*-index est le pourcentage de clés existantes à tout moment dans chacun de ses noeuds.

Plus ce taux est élevé meilleure est la performance car, un taux de remplissage grand implique moins de "trous" dans les noeuds et en conséquence moins de niveaux dans l'arbre. Le niveau le plus grand est la profondeur de l'arbre et c'est elle qui détermine le nombre d'accès nécessaires pour accéder à un tuple quelconque. Ceci est en fait le coût d'accès à un tuple comme nous l'étudierons dans §5.3.6. Dans <KNU74> on trouve aussi une autre définition de B*-arbres (nous les appelerons des B**-arbres) qui sont plus performants que ceux que nous définissons ici, dans la mesure où leur taux de remplissage est meilleur; il est égal à 67%.

Bien que les B**-arbres soient plus performants que les B*-arbres, nous avons choisi ces derniers car, les algorithmes qui implémentent les premiers sont très complexes et occupent beaucoup de place mémoire (lors de leur traduction en programmes). Pour ces raisons, ils ne sont adaptés à une implémentation dans un environement de micro-ordinateurs comme le nôtre.

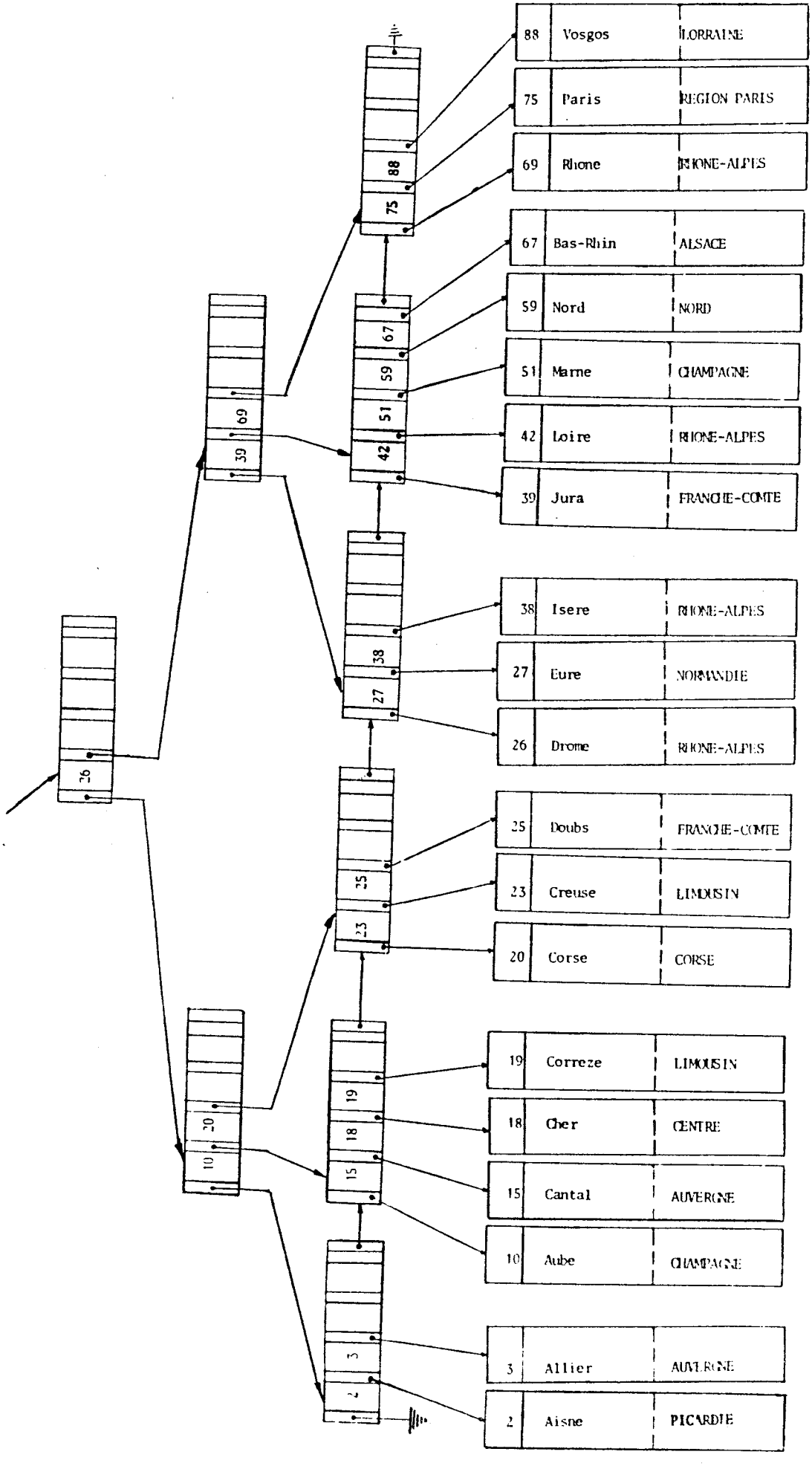


Figure 5.2 Un B*-arbre d'ordre 5 et de profondeur 4.
La relation de base DEPARTEMENTS (INDICATIF,NOMDEPT,REGION)
est pointée par un B*-index.

INVDEPT (P_0,K_1,P_1,K_2,P_2,K_3,P_3,K_4,P_4,LIEN)

idt	P_0	K_1	P_1	K_2	P_2	K_3	P_3	K_4	P_4	LIEN
(r,1)	r 2	26	r 3							
(r,2)	r 4	10	r 5	20	r 6					
(r,3)	r 7	39	r 8	69	r 9					
(r,4)		2	s 15	3	s 1					r 5
(r,5)	s 16	15	s 2	18	s 13	19	s 17			r 6
(r,6)	s 20	23	s 12	25	s 14					r 7
(r,7)	s 3	27	s 11	38	s 4					r 8
(r,8)	s 5	42	s 6	51	s 10	59	s 18	67	s 19	r 9
(r,9)	s 7	75	s 8	88	s 9					

DEPARTEMENTS(INDICATIF,NOMDEPT,REGION)

idt	INDICATIF	NOMDEPT	REGION
(s, 1)	03	Allier	AUVERGNE
(s, 2)	15	Cantal	CORSE
(s, 3)	26	Drôme	RHONE-ALPES
(s, 4)	38	Isère	RHONE-ALPES
(s, 5)	39	Jura	FRANCHE-COMTE
(s, 6)	42	Loire	RHONE-ALPES
(s, 7)	69	Rhone	RHONE-ALPES
(s, 8)	75	Paris	REGION-PARISIENNE
(s, 9)	88	Vosges	LORRAINE
(s,10)	51	Marne	CHAMPAGNE
(s,11)	27	Eure	NORMANDIE
(s,12)	23	Creuse	LIMOUSIN
(s,13)	18	Cher	CENTRE
(s,14)	25	Doubs	FRANCHE-COMTE
(s,15)	02	Aisne	PICARDIE
(s,16)	10	Aube	CHAMPAGNE
(s,17)	19	Correze	LIMOUSIN
(s,18)	59	Nord	NORD
(s,19)	67	Bas-Rhin	ALSACE
(s,20)	20	Corse	CORSE

Figure 5.3

5.3.2 Les types d'accès fournis par un B*-arbre.

La structure d'un B*-arbre induit directement deux modes d'accès:

i) Accès associatif: Un B*-index permet d'accéder à un tuple d'une relation de base dont la clé est connue sans avoir besoin de parcourir séquentiellement toute la relation.

ii) Accès séquentiel ordonné: Ici, nous attirons l'attention sur le fait que les feuilles du B*-index sont chainées (Figure 5.3). Ceci est très interessant car, cela permet de parcourir la relation de base pointée, d'une façon ordonnée (l'ordre $\preccurlyeq$ étudié dans le paragraphe précedent). En fait, il suffit de parcourir séquentiellement la chaine des noeuds feuilles du B*-index (c'est-à-dire le niveau p-1 du B*-arbre) pour obtenir un accès ordonné selon l'ordre des clés qui y sont stockées.

Ce type d'accès fait des B*-arbres une structure de données bien adaptée à la représentation des relations inverses (cf §3.2).

5.3.3 La Représentation d'un B*-arbre

Un B*-arbre est stocké dans la Base de Données sous forme de deux relations: une relation de type inverse et une relation de base, représentant respectivement le B*-index et les feuilles du B*-arbre. Dans cette optique le

B*-arbre de la Figure 5.2 est une visualisation de la relation de base DEPARTEMENTS (INDICATIF, NOMDEPT, REGION) et d'une relation inverse construite sur la première selon un ordre croissant établi sur l'attribut INDICATIF. Par ailleurs, c'est de cette manière que le concept de clé au sens du modèle relationnel a été implementé dans MICROBE. Dans notre exemple cette clé est représentée par INDICATIF.

Un noeud d'un B*-index est représenté par un tuple et l'ensemble de ces tuples constitue une relation de type inverse. Un tuple est composé de clés et des pointeurs. Un pointeur est ici un idt (identificateur de tuple) d'un tuple du B*-arbre. Nous rappelons qu'un idt est une adresse interne exprimée par le couple: (page, déplacement). Le stockage physique du B*-arbre de la figure 5.2 est représenté par la figure 5.3. Dans cette figure INVDEPT est une relation inverse établie sur la relation DEPARTEMENTS. Nous supposons que r et s sont des adresses des pages contenant respectivement les relations INVEDPT et DEPARTEMENTS. La racine du B*-arbre est le tuple identifié par (r,1).

5.3.3.1 Le format d'un noeud.

Dans ce paragraphe, nous décrivons le format exact d'un tuple représentant un noeud d'un B*-index. La figure 5.4 illustre un tel noeud. EX est un indicateur (témoin d'existence) que MIMER associe à chacun des tuples d'une relation de la Base de Données et NDC le nombre de clés

effectif stockées dans le noeud.

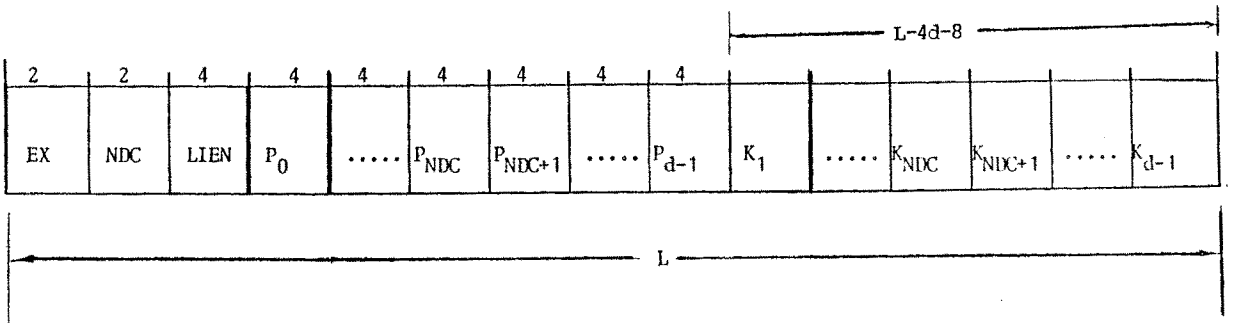


Figure 5.4

Voici trois remarques sur ce format :

i) $\lceil d/2 \rceil \leq NDC \leq d-1$ (d est un nombre <u>impair</u> superieur ou égal à 3),

ii) L'intervalle $[P_{NDC+1}, P_{d-1}]$ correspond à des trous de pointeurs en attente d'être remplis; de même que l'intervalle $[K_{NDC+1}, K_{d-1}]$ correspond à des trous de clés en attente d'être remplis,

iii) $L \leq LNMTPL$, où LNMTPL est une constante MIMER.

Etant donné qu'une clé est composée d'un ou plusieurs attributs (qui sont appelés les attributs-clé de cette clé); chaque Ki est en fait la concatenation d'un ensemble de valeurs, chacune d'elles appartenant à un attribut-clé. Si Ai représente l'ième attribut-clé d'une clé et NA le nombre d'attributs-clé d'une clé quelconque, alors, la longueur LKI de cette clé est :

$$LKI = \sum_{i=1}^{NA} \text{longueur}(Ai)$$

et

$$L = (4 + LKI) * d - LKI + 8$$

si la taille effective (en octets) d'une page est TUPAG et n le nombre de noeuds stockés dans cette page, nous cherchons à minimiser le nombre d'octets gaspillés par page; autrement dit, il nous faudra trouver une valeur de d tel que (TUPAG - n*L) soit minimal. Il est évident que ceci est strictement dépendant de LKI; c'est pour cela qu'à chaque B*-index nous avons associé un ordre d qui doit être calculé préalablement à la création de l'arbre.

Considérons, à manière d'exemple, un cas dans lequel LKI=26 et TUPAG=508, on aura donc, L=30d-18 et le tableau de valeurs suivant:

d	L	n	(TUPAG-n*L)
3	72	7	4
5	132	3	12
7	192	2	124
9	252	2	4
11	312	1	196
13	372	1	136
15	432	1	76
17	492	1	16

D'après la strategie proposée auparavant, d pourra avoir les valeurs 3 ou 9. Nous choisissons celle qui produit le moins de noeuds dans une page et en conséquence, le moins d'accès aux tuples. Concrètement, d est égal à 9.

Enfin, notre choix de $d_{optimal}$ peut être formulé ainsi:

$$d_{optimal} = MAX_d \left\{ d \mid MIN (TUPAG \text{ modulo } L) \right\}$$

5.3.4 Création d'un B*-arbre.

Comment peut-on créer un B*-arbre en MIMER ? En fait la création d'une telle arborescence est realisée en deux étapes: les feuilles de l'arbre sont créées à chaque fois que l'utilisateur crée une relation de base par l'intermédiaire de la primitive CREER-RELATION. Le B*-index est créé comme conséquence de l'appel à la primitive CREER-RELATION-INVERSE. Ceci ne modifie en rien la relation de base créée au préalable.

Une relation de base est indépendante de son B*-index cela permet donc, de créer autant de B*-index que l'on veut sur cette relation. Par contre, un B*-index conserve une dépendance très étroite avec la relation de base sur laquelle il pointe, en ce sens qu'une modification (ou insertion ou suppression) de tuples sur la relation de base se répercute sur le B*-index, provoquant parfois une réorganisation dans sa structure. Deux opérations pouvant produire un changement dans la structure du B*-index ont été définies: INSERER-CLE et SUPPRIMER-CLE; nous les étudierons en détail dans les paragraphes suivants.

5.3.5 Les opérations sur un B*-arbre.

Trois opérations fondamentales ont été définies sur un B*-arbre: consultation, insertion et suppression de tuples ayant une clé connue. Celles-ci sont materialisées par des primitives MIMER de l'interface application (cf §3.9.2): TUPLE-SUIVANT (dans un balayage indirect), INSERER-TUPLE et DETRUIRE-TUPLE (dans un balayage indirect).

Ces opérations sont implémentées au moyen de trois autres opérations (primitives de l'interface B) définies sur les B*-index, ce sont : CHERCHER-CLE, INSERER-CLE et SUPPRIMER-CLE. Par la suite, nous allons détailler ces dernières.

5.3.5.1 Consultation d'une clé d'un B*-index.

Cette opération est exécutée lors de l'appel de la primitive:

CHERCHER-CLE (IREL, CLE, TROUVE, IDTNOEUD, POSNOEUD)

Ceci permet de déterminer si une clé quelconque est stockée dans un B*-index dont le nom interne est IREL. Si cette clé fait partie du B*-index, le paramètre booleén TROUVE a la valeur vrai, autrement faux. Quand TROUVE est vrai, le paramètre IDTNOEUD transmet l'idt d'un des noeuds feuilles du B*-index qui stocke le pointeur vers le tuple qui correspond à CLE. Puisqu'un noeud peut allouer plusieurs pointeurs, le paramètre POSNOEUD indique lequel

d'entre eux correspond à CLE.

L'algorithme nécessité à l'exécution de cette opération consiste en un parcours de l'arbre, respectant des reglès bien définies: si la valeur de la clé qu'on cherche est inférieure/supérieure à la valeur la clé Ki dans un noeud, le parcours se poursuit vers le pointeur gauche/droite de Ki, sinon vers le pointeur droit/gauche.

Pour accélérer cette opération, une recherche dichotomique est faite à l'intérieur de chaque noeud ayant l'ordre d qui depasse un certain seuil.

Exemple:

Si on suppose que le nom interne de la relation INVEPT de la figure 5.3 est IINVEPT, à la fin de l'exécution de la primitive: CHERCHER-CLE (IINVEDPT, 19, TROUVE, IDTNOEUD, POSNOEUD); les paramètres TROUVE, IDTNOEUD et POSNOEUD auront respectivement les valeurs: vrai, (r,5) et 3.

5.3.5.2 Insertion d'une clé dans un B*-index.

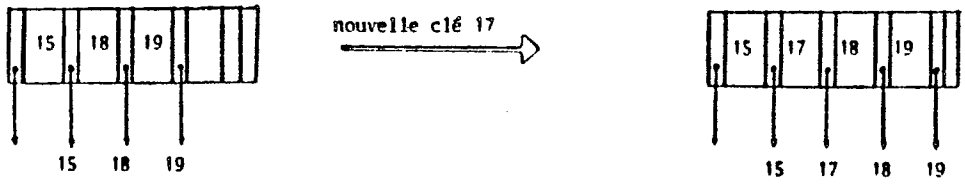
Cette opération permet d'insérer une nouvelle clé dans un B*-index, tout en préservant les proprietés de l'arbre. Deux cas peuvent se présenter: insertion simple et insertion avec éclatement. La primitive qui permet de réaliser cette opération est :

INSERER-CLE (IREL, CLE)

où IREL est le nom interne du B*-index et CLE est la clé que l'on veut insérer.

5.3.5.2.1 Insertion simple.

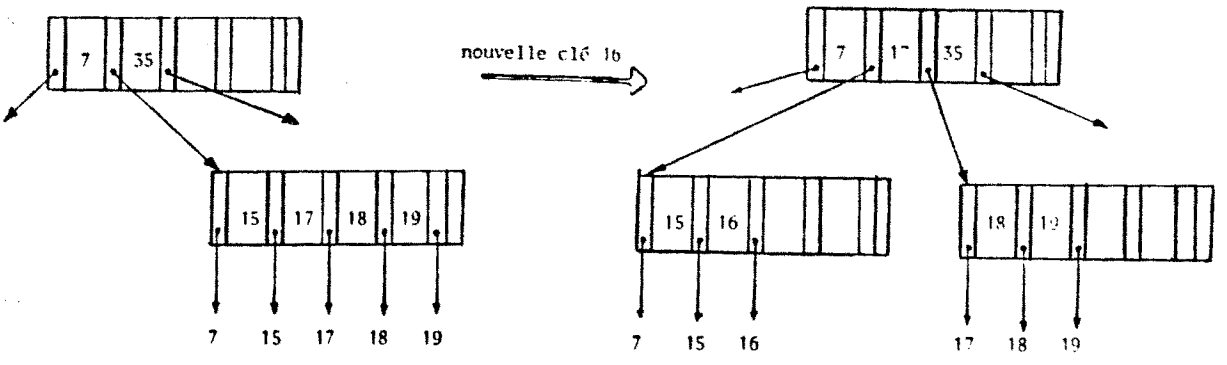
La nouvelle clé est rangée dans l'une des feuilles du B*-index; l'opération ne met en jeu qu'un seul noeud. Nous pouvons le voir sur un exemple:



5.3.5.2.2 Insertion avec éclatement.

Ce cas se présente quand un noeud feuille du B*-index, dans lequel l'insertion doit avoir lieu est saturé. Le noeud est éclaté en deux et l'une de ses clés doit monter au noeud père.

Voyons tout de suite un exemple sur un B*-arbre d'ordre 5:



Il se peut que l'éclatement se répercute aux niveaux supérieurs de l'arbre et qu'il arrive jusqu'à la racine laquelle est éclatée à son tour, produisant une nouvelle racine. Ceci augmente, bien entendu, la profondeur de l'arbre.

5.3.5.3 Suppression d'une clé dans un B*-index.

Cette opération est l'inverse de l'insertion; elle permet de supprimer une clé existante dans un B*-index. L'opération est materialisée par la primitive:

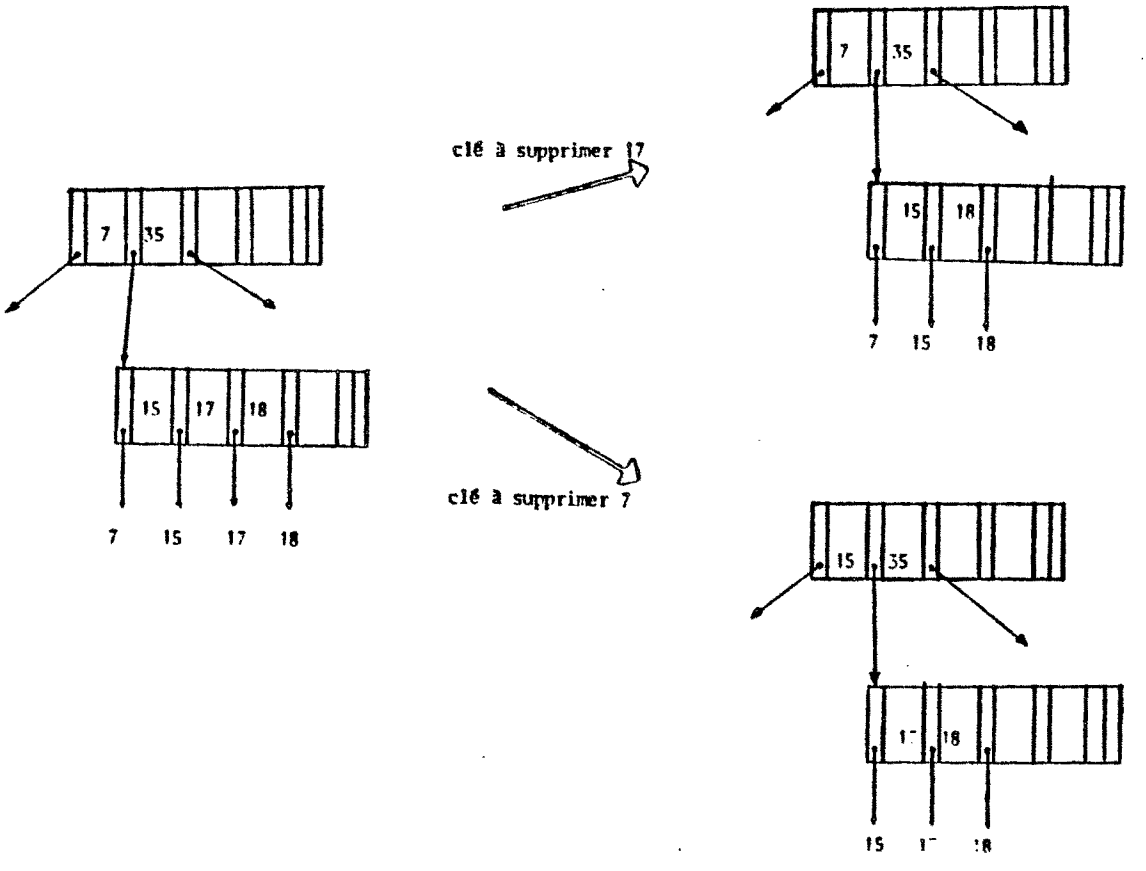
SUPPRIMER-CLE (IREL, CLE, IDT)

où IREL est le nom interne du B*-index et CLE est la clé que l'on veut supprimer. Si le B*-index est primaire (à clés non-répetées; cf §3.2 et §4.1.5), IDT n'a aucun sens. Dans le cas d'un B*-index secondaire (à clés répetées) IDT designe la clé à supprimer parmi l'ensemble des clés ayant la même valeur. IDT est donc, l'identificateur de tuple correspondant à un tuple de la relation de base, dont la clé

doit être supprimée du B*-index.

Trois cas de suppression peuvent se présenter: supression simple, suppression avec débordement et suppression avec fusion . Dans tous les cas il faudra garantir un taux de remplissage de 50%.

5.3.5.3.1 Suppression simple.



Elle se présente lors d'une suppression de clé ne diminuant pas le taux de remplissage de moins de 50%. Il existe deux cas:

i) La clé se trouve dans l'un des noeuds feuilles du B*-index; ceci ne met pas en jeu d'autres noeuds,

ii) La clé se trouve dans un noeud non-feuille du

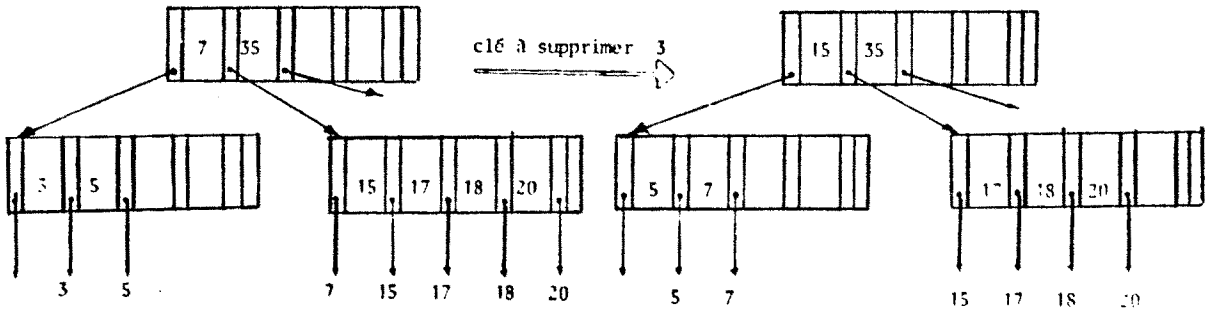
B*-index; ici une clé d'un des ses noeuds fils doit être montée.

L'exemple ci-dessus permet de visualiser ces deux cas.

5.3.5.3.2 Suppression avec débordement.

Ce cas se présente lors de la suppression d'une clé d'un noeud qui laisse celui-ci avec un taux de remplissage inferieur à 50%. Il faut redistribuer les clés de ce noeud avec celles d'un de ses noeuds frères (à gauche ou à droite).

Exemple:

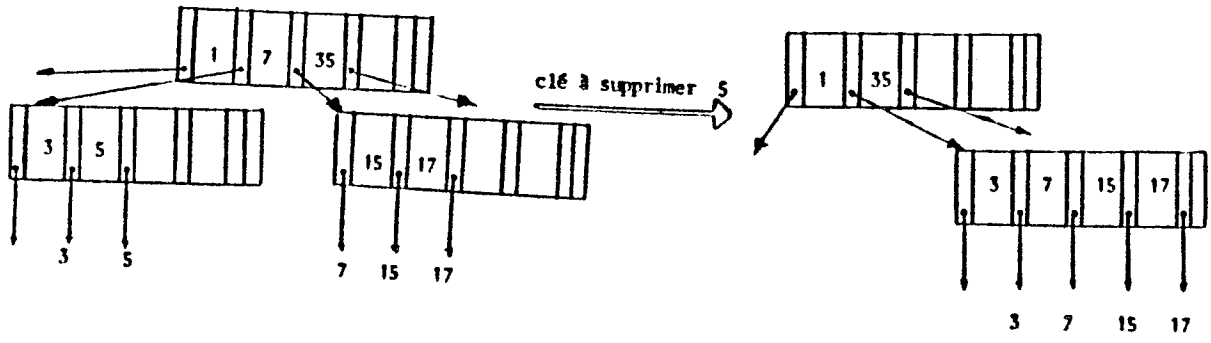


5.3.5.3.3 Suppression avec fusion.

Ce cas se présente lors de la suppression d'une clé d'un noeud qui laisse celui-ci avec un taux de remplissage inferieur à 50% mais, une redistribution de clés n'est pas possible car, ceci provoquerait un taux de remplissage inferieur à 50% dans les autres noeuds. Le traitement adopté consiste en éliminer le noeud où se trouve la clé à

supprimer et de faire passer éventuellement à ses noeuds frères, les clés qui restent.

Exemple:



5.3.5.4 Quelques remarques.

Nous venons d'étudier un ensemble élémentaire d'opérations sur un B*-index. Nous avons vu que dans quelques cas (en particulier, insertion avec éclatement et suppression avec fusion) la structure du B*-index peut changer en ce sens qu'elle augmente ou diminue de profondeur. Quoi qu'il arrive lors d'une telle opération, le chainage des feuilles du B*-index doit être garanti car, c'est lui qui pourvoit le moyen de balayer une relation de base de façon ordonnée.

MIMER permet la création de deux types d'index: primaires (des B*-index à clés non-répetées) et secondaires (des B*-index à clés répetées). Ces deux types d'index sont gérés avec les mêmes opérations. Or, l'opération INSERER-CLE est "aveugle" en ce sens qu'elle ne regarde pas si d'autres clés, ayant la même valeur que celle que l'on veut insérer, existent déjà dans le B*-index. En

conséquence, il est à primitive INSERER-TUPLE de MIMER de faire fâce à ce problème. De même, l'opération SUPPRIMER-CLE ne supprime que l'une des clés répetées d'un B*-index secondaire; ici aussi il incombe à la primitive DETRUIRE-TUPLE de MIMER de générer une boucle permettant de supprimer toutes les clés ayant une même valeur.

5.3.6 Le coût d'accès à un tuple.

Considérons une relation ayant une cardinalité N; l'opération d'accès, à l'un de ses tuples dont on connaît la clé, a un certain coût qui est fortement lié à la méthode d'accès choisie. Ce coût d'accès sera considéré comme étant le nombre de tuples, auxquels il faut accéder pour récupérer le tuple qui nous intéresse. Par la suite, nous allons analyser le coût d'accès à un tuple lorsqu'on utilise la méthode de B*-arbres.

La profondeur d'un B*-arbre est égale au nombre de ses niveaux. Le niveau 1 est la racine et le niveau le plus élevé est représenté par les feuilles de l'arbre (c'est-à-dire les tuples de la relation de base que l'on veut accéder).

Le "pire" B*-index que l'on puisse construire est celui dans lequel tous les noeuds ne sont remplis qu'à moitié et chaque noeud se trouve dans une page physique differente. La fonction nbm(i) nous donne le nombre de noeuds existants dans le niveau i de ce B*-index (où p' est la profondeur de

l'arbre).

$$nbm(i) = \begin{cases} d^{i-1} & \text{pour } 1 \leqslant i \leqslant p'-1 \\ N & \text{(cardinalité de la relation de base) si } i=p' \\ 0 & \text{autrement} \end{cases}$$

Si T est le nombre total de noeuds d'un tel B*-index alors:

$$T = \sum_{i=1}^{p'-1} d^{i-1} = \frac{d^{p'-1} - 1}{d-1}$$

puisqu'il est évident que N>T (sauf si N=1), nous pouvons déduire que

$$N > \frac{d^{p'-1} - 1}{d-1}$$

ce qui veut dire: $p' < 2 + \log_d N$

En fait, la profondeur d'un B*-arbre est implicitement le nombre de noeuds qu'il faut accéder pour récupérer un tuple donné. Autrement dit : la profondeur est le coût d'accès à un tuple.

D'autre part, le nombre d'accès minimum qu'il faudra faire pour récupérer un tuple est 2. Ce cas se présente dans des arbres n'ayant que deux niveaux. Il est évident que des arbres à un seul niveau n'existent pas dans la structure que nous proposons ici.

Nous pouvons maintenant conclure que le coût d'accès p à un tuple est donné par:

$$2 \leqslant p \leqslant 2 + \log_d N$$

Pour avoir une idée plus claire de ce que cela signifie, regardons le tableau ci-dessous qui indique le coût d'accès pour quelques valeurs de N et de d.

N \ d	5	10	50
10^3	7	5	4
10^4	8	6	5
10^5	9	7	5
10^6	11	8	6

Si l'accès à un tuple d'une relation est réalisée par une méthode séquentielle et non pas par un B*-index, on aurait un coût d'accès $p \leqslant N$. Evidemment, un B*-arbre établi une méthode d'accès selectif par clé, qui est en général beaucoup plus performante que la méthode séquentielle.

5.4 Le hachage

L'adresse d'un tuple est calculée au moyen d'une fonction, dite fonction de hachage, qui est appliquée sur la

clé d'un tuple, ceci lui permet d'accéder en O(1) essai.

Pendant des années, le hachage a été considéré comme une technique quasiment statique car, les insertions peuvent engendrer des collisions et les suppressions engendrent du gaspillage d'espace secondaire puisqu'il est alloué statiquement. Même très peu de collisions peuvent profondément déteriorer les performances d'accès.

Une Base de Données contenant beaucoup de collisions doit être réorganisée en changeant la fonction de hachage (cette opération s'appelle rehachage), cela nécessite un logiciel sophistiqué et peut rendre la Base de Données indisponible pendant longtemps. Cette difficulté a été surmontée dans les travaux récents de LITWIN <LIT79>, LARSON <LAR78> et FAGIN <FAG79> dans lesquels le hachage a été revalorisé en le pourvoyant de structures de stockage de données où l'espace d'adressage est geré dynamiquement et n'est pas réorganisée complètement.

Prenons le cas du hachage virtuel <LIT79>: l'espace d'adressage est représenté par une fonction de hachage, où chaque adresse identifie de façon unique une case allouant un nombre fixe de tuples. Cet espace peut changer car, la fonction est en fait un élément d'une famille de fonctions.

Par exemple, si $h_1(c)$ est l'adresse d'une case allouant un tuple dont la clé exprimée numériquement est c et, N une constante entière supérieure à zéro:

$$h_i(c) = c \underline{\text{modulo}}\ 2^i * N$$

ici hi est une famille de fonctions qui permet de "hacher" sur un espace de $2^i * N$ adresses. Le mécanisme de fonctionnement est très simple: avec i égal à 0, on obtient un espace adressable de N cases; il suffirait d'incrémenter i, lors du débordement de l'une de cases, pour avoir le double d'espace adressable.

Enfin, on ne réstructure que la case qui déborde et dans une "table de bits" on enregistre ces cas.

Cependant, un problème demeure encore difficile à résoudre: le parcours selon l'ordre des clés. Plusieurs solutions ont été proposées; nous retenons ici les deux suivantes:

.Il existe des fonctions de hachage dites <u>à distribution calculée</u> au moyen desquelles il est possible de ranger des tuples selon l'ordre croissant de leurs clés. Il s'agit de polynômes dont les coefficients sont choisis en fonction de la connaissance des clés. Hélas, cela implique forcément une très bonne connaissance de la distribution des clés. Ce qui n'est pas toujours un paramètre de connu (surtout dans le cas de clès non-numériques). Evidemment, un SGBD général ne pourrait jamais prevoir des tels coefficients. Ce serait au créateur d'une relation de les fournir.

.LITWIN présente dans <LITRO> l'algorithme de "Trie hash". Celui-ci permet d'accéder séquentiellement à un fichier qui peut être également accedé selectivement selon une fonction de hachage. Le fichier est créé d'une fois pour toutes selon un certain ordre et un parcours selon un ordre différent ne sera pas possible. Ceci n'est pas le cas pour les B*-arbres.

Enfin, une méthode d'accès dans un SGBD a un "prix" qui est fonction de la complexité de son algorithmique, de la taille de mémoire occupée par les données de contrôle, du taux de remplissage du fichier de données et du nombre d'accès nécessaires pour retrouver une donnée. Dans cette optique les B*-arbres fournissent un accès ordonné et selectif par clé à un prix élevé, tandis que le hachage ne fournit qu'un accès selectif, mais à un prix beaucoup plus bas.

6 CONCLUSIONS

Nous avons démontré qu'il est possible de réaliser, sur des micro-odinateurs, un outil de base à partir duquel on peut facilement mettre en oeuvre des systèmes relationnels. Par ailleurs, en réalisant le SGBD relationnel MICROBE nous avons donné un exemple pratique d'application de MIMER qui en constitue la couche de base.

Nos efforts on été dirigés vers la gestion d'un schéma conceptuel dynamique, c'est-à-dire capable d'évoluer selon les besoins changeants des applications. Dans la pratique: les tuples, les attributs et les relations peuvent être créés ou détruits au gré de ces besoins.

Nous n'avons pas voulu restreindre notre solution à une machine particulière. Ceci nous a amené à utiliser un langage de haut niveau: PASCAL, très répandu à l'heure actuelle et faisant partie du logiciel de beaucoup de micro-machines.

Des interfaces adéquates permettent de rehausser le niveau d'utilisation de MIMER. C'est ainsi qu'un utilisateur du niveau application se voit placé au niveau du modèle relationnel et son seul souci est le traitement des entités relationnelles: tuples, attributs et relations.

Une partie de notre travail a été consacré à l'étude de méthodes d'accès. Dans ce domaine nous avons adapté la

structure arborescente des B*-arbres qui nous permet d'accéder aux tuples d'une façon ordonnée et/ou sélective. La structure arborescente est représentée par des relations "normales", ce qui allège le système car le traitement de celles-ci ne requiert pas de procédures spéciales.

Comme suite à notre travail, il est envisageable de lancer des recherches dans différents domaines relatifs aux SGBD sur micro-ordinateurs. Avec MIMER comme support il est possible, en particulier :

i) d'envisager la réalisation d'un prototype multi-utilisateurs: une solution simple pour contrôler l'accès concurrent, consisterait à gérer des verrous au niveau de chaque relation,

ii) de protéger les informations; une solution simple et réalisable à court terme est possible, en associant à un utilisateur donné, un ensemble de droits d'accès. MIMER pourra stocker la relation interne

ACCES (UTILISATEUR, RELATION, DROITS)

où chacun de ses tuples établit les DROITS de l'UTILISATEUR (Lecture, Mise à jour, Insertion, Suppression, etc...) sur une RELATION.

iii) de permettre la reprise en cas de panne de machine ou d'avortement d'une requête,

iv) de mettre en oeuvre des contraintes d'integrité. Une solution simple est envisageable: implantation de ces contraintes au moyen d'une relation interne qui stocke une liste de prédicats pour chaque relation de la Base. Insertions ou modifications, sur des tuples d'une

relation, devront toujours préserver ces contraintes.

MIMER est donc un outil opérationnel pour la construction de systèmes de Bases de Données relationnels. Nous vous proposons de voir finalement, la Figure 6.1 qui indique quelques applications et extensions qui pourraient être développées par la suite.

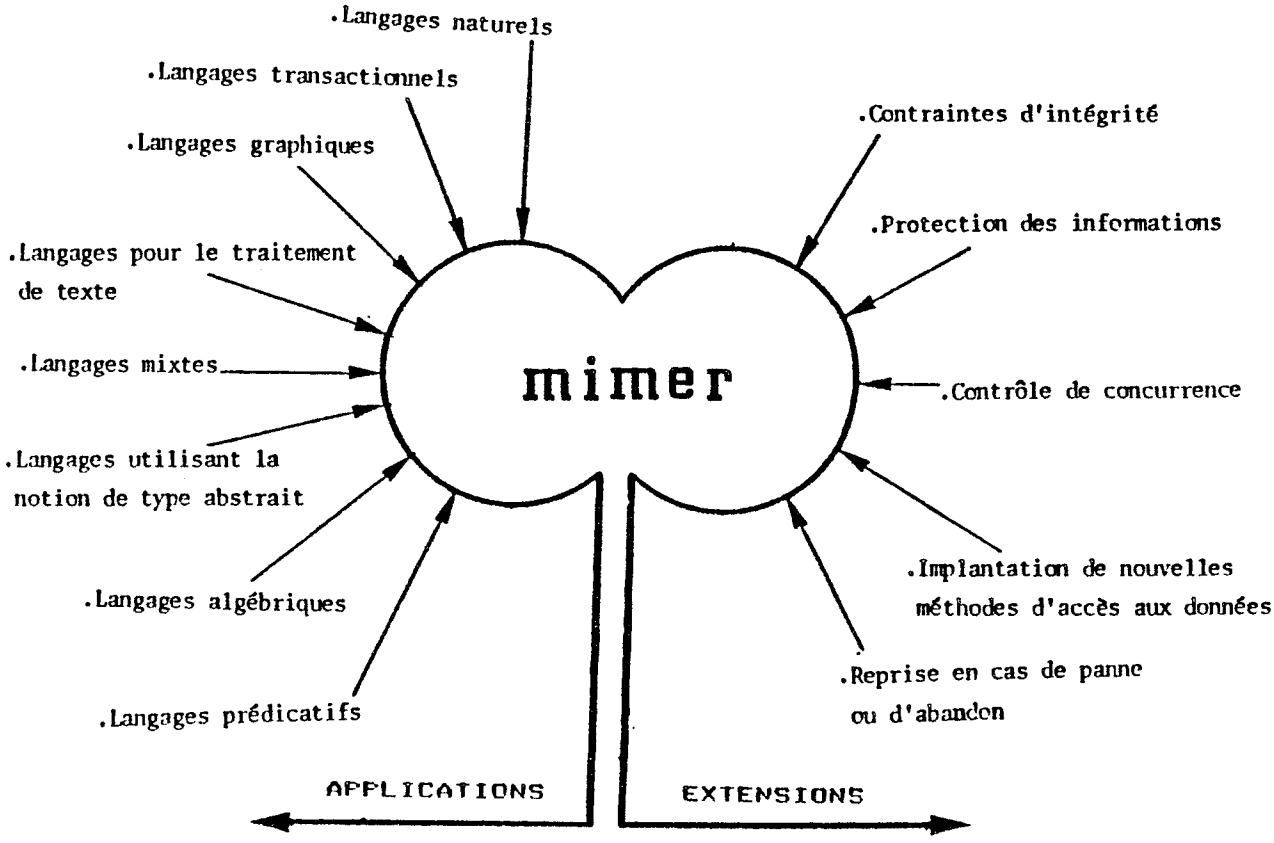


Figure 6.1

ANNEXE 1. La syntaxe du langage MIQUEL

[ ] : signifie que le contenu des crochets est optionnel.

```
<commande>      ::=  <requete> ;
<requete>       ::=  <creerbase>
                   / <supbase>
                   / <creerel>
                   / <suprel>
                   / <creeratt>
                   / <supatt>
                   / <creerindex>
                   / <insertuple>
                   / <suptuple>
                   / <modiftuple>
                   / <interrogation>
                   / <listrel>
                   / <caractrel>
                   / <listatt>
                   / <listindex>
                   / <ouvrirbase>
                   / <fermerbase>
                   / <fin-session>
```

Commandes de définition de données

```
<creerbase>     ::=  CREATE_DATABASE
                   / CB
<supbase>       ::=  SCRATCH_DATABASE
<creerel>       ::=  CREATE_RELATION <defrel>
                   / CR <defrel>
<defrel>        ::=  <identrel> <def-affect>
<def-affect>    ::=  := <rel-affect>
                   / <liste-att-def>
<rel-affect>    ::=  <identrel>
                   / <select1>
                   / <identrel> UNION <identrel>
                   / <identrel> INTERSECTION <identrel>
<liste-att-def> ::=  <att-def>
                   / <att-def> , <liste-att-def>
<att-def>       ::=  <identif> ( <type> )
<type>          ::=  INTEGER
                   / REAL
                   / CHARACTER ( <entier> )
<suprel>        ::=  DELETE_RELATION <identrel>
<supatt>        ::=  DELETE_ATTRIBUT <identrel> :
                     <liste-attributs>
<creerindex>    ::=  CREATE_INDEX  <def-index>
                   / CI <def-index>

<def-index>     ::=  <identrel> ( <identindex> :
                     <liste-attributs> ) <direction>
<creeratt>      ::=  CREATE_ATTRIBUT  <identrel> :
                     <liste-att-def>
```

Commandes de manipulation de données

```
<insertuple>      ::=  INSERT_INTO
<suptuple>        ::=  DELETE_FROM  <identrel>
                       WHERE  <booleen>
<modiftuple>      ::=  UPDATE  <identrel>  SET <nomcol>
                       := <terme> [ WHERE  <booleen> ]
<terme>           ::=  <nomcol>  /  <constante>
<interrogation>   ::=  <select1>
                       [ ORDERBY  <ordre-att> ]
<select1>         ::=  SELECT  <liste-exp-sel>
                       FROM  <liste-rel>
                       [ WHERE <booleen> ]
<select2>         ::=  SELECT  <attsel2>
                       FROM  <liste-rel>
                       [ WHERE <booleen> ]
<liste-exp-sel>   ::=  <uniselect>
                     / <fonction> <att-sel3>
                     / CARD       ( <uniselect> )
<uniselect>       ::=  [ UNIQUE ] <att-sel1>
<att-sel3>        ::=  ( <att-sel2> )
<att-sel1>        ::=  *
                     / <liste-attribut>
<att-sel2>        ::=  [ UNIQUE ] <attribut>
<fonction>        ::=  MIN / MAX / MOY / SOM
<liste-attribut>  ::=  <attribut>
                     / <attribut>, <liste-attribut>
<attribut>        ::=  <nom-col>
                     /  <identrel>.<nom-col>
                     /  <var>.<nom-col>
<ordre-att>       ::=  <attribut> <direction>
                     / <attribut>, <ordre-att>
<direction>       ::=  ASC  /  DESC
<liste-rel>       ::=  <identrel>
                     / <identrel>.<var>, <identrel>.<var>
<booleen>         ::=  <fact-bool>
                     / <fact-bool>  <op>  <booleen>
<op>              ::=  AND  /  OR
<fact-bool>       ::=  <predicat>
                     / ( <booleen> )
<predicat>        ::=  <attribut>  <oparithm>  <expression>
                     / SET <attribut> = <select2>
<oparithm>        ::=  = / <= / >= / > / < / <>
<constante>       ::=  <nombre> / "chaine"
<nombre>          ::=  <entier> / <reel>
<expression>      ::=  <attribut>
                     / <select1>
                     / <constante>
```

Commandes d'obtention de caractéristiques

```
<listrel>         ::=  LIST_RELATIONS
                     / LR
<caractrel>       ::=  CHARACTERISTIC_RELATION <identrel>
                     / CH  <identrel>
<listatt>         ::=  LIST_ATTRIBUTS_FROM <identrel> : <atts>
                     / LA <identrel> : <atts>
```

```
<atts>          ::=  <liste-attributs>
                   / *
<listindex>     ::=  LIST_INDEX   <identrel> ( <index> )
                   / LI <identrel> ( <index> )
<index>         ::=  <liste-index>
                   / *
<liste-index>   ::=  <identindex>
                   / <identindex>, <liste-index>
```

Commandes de contrôle de session

```
<ouvrirbase>    ::=  OPEN_DATABASE
                   / OB
<fermerbase>    ::=  CLOSE_DATABASE
                   / CD
<fin-session>   ::=  BYE

<identrel>      ::=  <identif>
<identatt>      ::=  <identif>
<identindex>    ::=  <identif>
<identadm>      ::=  <identif>
<var>           ::=  <identif>
<nom-col>       ::=  <identif>
```

```
      /*********************************************************/
      /*                                                       */
      /*          CONSTANTES MIMER                             */
      /*                                                       */
      /*********************************************************/

LNEXT  =  6;    /* LONGUEUR MAXIMAL D'UN NOM EXTERNE */
LNNBA  = 12;    /* LNEXT + 6 */

LARGCA = 20;    /* Longueur maximal d'un argument de balayage dans la relation CURSEURS-ACTIFS;
                   Si l'argument est d'une longueur superieure à cette valeur, il sera stocke
                   dans la relation ARGUMENTS.
                */
LNMTPL =508;    /* LONGUEUR MAXIMAL D'UN TUPLE */

DEMITPL=254 ;    /* DEMITPL = LNMTPL / 2 */
QRTPL  =127;    /* QRTPL = LNMTPL / 4 */
BUFLG  =512;    /* TAILLE D'UNE PAGE */
TUPAG  =508;    /* TAILLE UTILE D'UNE PAGE ( BUFLG - 4 ) */
NBUFF  =  4;    /* NOMBRE DE BUFFERS POUR PAGINATION */

/* LONGUEUR DES TUPLES DES CATALOGUES. CELA EVITE D'ALLER CONSULTER  A
 CHAQUE FOIS DANS LA RELATION MAITRE.
 */
LNTIX  = 16;    /* INDEX */
LNTAC  =  10;    /* ATTRIBUTS-CLE */
LNTAT  = 18;    /* ATTRIBUTS */
LNTM   = 30;    /* MAITRE */
LNTCA  = 144;    /* CURSEUR ACTIFS */   /* LARGCA + 124 */

LNDES  = 52;  /* LONGUEUR DU TUPLE DESCRIPTEUR DE LA B. DE D. */

/* LES 5 CONSTANTES SUIVANTES SONT LE RESULTAT DE DIVISER TUPAG PAR LES
 5 CONSTANTES PRECEDENTES.  EX:  MTPG = TUPAG / LNTM
 */
XTPG   = 31;    /* NB.MAXIMUN DE TUPLES, DE LA RELATION INDEX, PAR PAGE */
AXTPG  = 50;    /* NB.MAXIMUN DE TUPLES, DE LA RELATION ATTRIBUTS-CLE, PAR PAGE */
ATPG   = 28;    /* NB.MAXIMUN DE TUPLES, DE LA RELATION ATTRIBUTS, PAR PAGE */
MTPG   = 16;    /* NB.MAXIMUN DE TUPLES, DE LA RELATION MAITRE, PAR PAGE */
CATPG  = 3;    /* NB.MAXIMUN DE TUPLES, DE LA RELATION CURSEURS-ACTIFS,PAR PAGE */

NILENT = 2000; /* LA VALEUR INDEFINIE POUR LES ENTIERS=-2**15     */
NILREL = -1E-38; /* LA VALEUR INDEFINIE POUR LES REELS    */
NILCHR = ''; /* LA VALEUR INDEFINIE POUR LES CARACTERES (CTRL C) */

NUL    = 0      ;

MNREL  = 15;    /* NB.MAXIMUN DE RELATIONS DANS LA B. DE D. */
MNBATT = 10;    /* NB.MAXIMUN D'ATTRIBUTS D'UNE RELATION */
MNBINX = 10;    /* NB.MAXIMUN D'INDEX SUR UNE RELATION    */
MNBATC = 10;    /* NB.MAXIMUN D'ATTRIBUTS-CLE D'UN INDEX */


LNENT  =  2;    /*LONGUEUR EN BYTES D'UN ATTRIBUT DE TYPE ENTIER        */
LNREEL =  2;    /* LONGUEUR EN BYTES D'UN ATTRIBUT DE TYPE REEL          */
LNCHAR =  1;    /* LONGUEUR EN BYTES D'UN ATTRIBUT DE TYPE CHARACTER    */
```

ANNEXE 3. TYPES DE VARIABLES DEFINIES DANS MIMER

```
TYPE
   /*****************************************************************/
   /*                                                               */
   /*          TYPES      MIMER                                     */
   /*                                                               */
   /*****************************************************************/

   EXNAME  =      ARRAY °1..LNEXT§ OF CHAR ; /* NOMS EXTERNES DES ATTRIBUTS ET DES RELATIONS */
   NAME    =        ARRAY °1..LNNBA§ OF CHAR ;
   CHAR2       =    ARRAY °1..2§ OF CHAR ;
   CHAR12      =    ARRAY °1..12§ OF CHAR ;
   ADRESS  =
   RECORD
      ADRPG    :    INTEGER ;        /* ADRESSE D'UNE PAGE */
      DEPL     :    INTEGER ;        /* DEPLACEMENT PAR RAPPORT AU DEBUT DE LA PAGE */
   END ;

   TUPMAI  =      /* UN TUPLE DE LA RELATION MAITRE */
   RECORD
      EX
         :
         CHAR2 ;
      NOMEXT   :    EXNAME
      TYPER    :    CHAR2
      CARD     :    INTEGER
      DEGRE    :    INTEGER
      LONGTP   :    INTEGER
      TYPSTK   :    CHAR2    ;
      PTRINX   :    ADRESS
      PTRATT   :    ADRESS
      PTRPP    :    INTEGER
      PTRDP    :    INTEGER
   END ;

   TUPATT  =      /* UN TUPLE DE LA RELATION ATTRIBUTS */
   RECORD
      EX
         :
         CHAR2
      NOMEXT   :    EXNAME
      TYPEA    :    CHAR2
      LONGAT   :    INTEGER
      POSIT    :    INTEGER
      CHAIAT   :    ADRESS
   END ;

   TUPINX  =      /* UN TUPLE DE LA RELATION INDEX */
   RECORD
      EX
         :
         CHAR2
      PTRMAI   :    ADRESS
      TYPEX    :    CHAR2
      CHAIX    :    ADRESS
      PTRAC    :    ADRESS
   END ;

   TUPAIX  =      /* UN TUPLE DE LA RELATION ATTRIBUTS-CLE */
   RECORD
      EX
         :
         CHAR2
      CHAIAC   :    ADRESS
      PTRATT   :    ADRESS
```

```
DESCREL    = /* UTILISE PAR LES PRIMITIVES D'OBTENTION DE CARACTERISTIQUES DES RELATIONS */
RECORD
  NMEXT    :       EXNAME  ;
  NOMINT   :       ADRESS  ;
  TYPERE   :       CHAR2   ;
  CARDIN   :       INTEGER         ;
  DEGRER   :       INTEGER         ;
  LONGTR   :       INTEGER         ;
  TYPESTK          :       CHAR2
END ;
TABCB      =    ARRAY °1..MNREL§ OF DESCREL ;
DESCATTS   = /* UTILISE PAR LES PRIMITIVES D'OBTENTION DE CARACTERISTIQUES DES ATTRIBUTS
               D'UNE RELATION. */
RECORD
  NMEXTA   :       EXNAME  ; /* NOM EXTERNE DE L'ATTRIBUT */
  NOMINTA          :       ADRESS  ; /* NOM INTERNE DE L'ATTRIBUT */
  TYPEAT   :       CHAR2   ; /* TYPE DE L'ATTRIBUT        */
  LONGATT          :       INTEGER         ; /* LONGUEUR DE L'ATTRIBUT    */
  POSITA   :       INTEGER    /* POSITION DE L'ATTRIBUT DANS LE TUPLE */
END ;
TABCA      =     ARRAY °1..MNBATT§ OF DESCATTS ;
TYPREL     =       /* TYPES DE  RELATIONS */
( MAITRE, ATTR,INX, ATTINX, INVER, CURACT, BASE, BENT,BREEL, DESCRI,
 CARELA, CARATT ) ;

PAGE     =
RECORD
  NBTUPL   :       INTEGER         ;       /*EN-TETE; NB. DE TUPLES PAR PAGE */
  PTRSUI   :       INTEGER ;       /*EN-TETE; POINTEUR VERS LA PAGE SUIVANTE */
  CASE     TYPREL          OF

       MAITRE  :       /* une page de la relation MAITRE */

       ( PAGMAI        :       ARRAY °1..MTPG§ OF TUPMAI ) ;
       ATTR    :       /* UNE PAGE DE LA RELATION ATTRIBUTS */
       (PAGATT         :       ARRAY °1..ATPG§ OF TUPATT ) ;
       INX     :       /* une page de la relation INDEX */
       ( PAGINX        :       ARRAY °1..XTPG§ OF TUPINX ) ;
       ATTINX  :       /* UNE PAGE DE LA RELATION ATTRIBUTS-CLE */
       (PAGATX         :       ARRAY °1..AXTPG§ OF TUPAIX) ;
       CURACT  :       /* UNE PAGE DE LA RELATION CURSEURS-ACTIFS */
       ( PAGCAC        :       ARRAY °1..CATPG§ OF TUPCAC) ;
       BASE    :       /* UNE PAGE D'UNE RELATION DE BASE      */
       (PAGBAS         :       ARRAY °1..TUPAG§ OF CHAR        )  ;
       DESCRI  :       /* PAGE 1 : DESCRIPTEUR DE LA B. DE D.  */
       ( PAGDES        :       TUPDES )
END ;

INFB             =       /* DES INFORMATIONS PORTANT SUR UN BUFFER */
RECORD
  /* NUMERO DE LA PAGE DONT ELLE EST UNE IMAGE : */
  NROPAG   :       INTEGER         ;
  /* COMPTEUR UTILISE PAR LA POLITIQUE DE REMPLACEMENT : */
  POLREM   :       INTEGER         ;
  /* POUR MARQUER SI LA PAGE DOIT ETRE REECRITE : */
  MODIF            :       BOOLEAN
END ;
```

```
TARGMT          =   ( AE, AR, ACH, POINTEUR) ;
ARGMT   =       /* ARGUMENT DE BALAYAGE */
RECORD
  CASE  TARGMT OF
        AE : ( ARGI      :   INTEGER ) ; /* ARGUMENT NUMERIQUE ENTIER */
        AR : ( ARGR      :   REAL    ) ; /* ARGUMENT NUMERIQUE REEL */
        ACH: ( ARGC    :   ARRAY °1..LARGCA§ OF CHAR ) ; /* ARGUMENT ALPHANUMERIQUE */
        POINTEUR : ( ARGP        :   ADRESS  ) ; /* POINTEUR VERS LA RELATION ARGUMENTS */
END ;

ATTARG          =
RECORD          /* LISTE D'ARGUMENTS          */
  TYPAA   :       CHAR2   ;       /* TYPE DE L'ATTRIBUT-ARGUMENT */
  PSTAA   :       INTEGER         ;       /* POSITION DE L'ATTRIBUT-ARGUMENT DANS LE TUPLE */
  LNGAA   :       INTEGER         ;       /* LONGUEUR DE L'ATTRIBUT-ARGUMENT */
  OPECMP  :       CHAR2   ;       /* OPERATEUR DE COMPARAISON  */
  OPELGQ  :       CHAR2   ;       /* OPERATEUR LOGIQUE LIANT LA NOUVELLE
                                   CONDITION A L'ENSEMBLE DE CONDITIONS
                                   */

END ;
TUPCAC  =       /* UN TUPLE  DE LA RELATION CURSEURS-ACTIFS */
RECORD

  EX              :       CHAR2           ;
  NOMREL  :       ADRESS          ; /* NOM INTERNE DE LA RELATION BALAYEE. */
  VALCUR  :       ADRESS          ; /* LONGUEUR D'UN TUPLE DE LA REL. BALAYEE. */
  LNGTU   :       INTEGER                 ; /* LONGUEUR D'UN TUPLE DE LA REL. BALAYEE. */
  LSTAAR  :       ARRAY °1..MNBATT§ OF ATTARG ; /* LISTE D'ATTRIBUTS-ARGUMENTS */
  NOMRIN  :       ADRESS          ;  /* NOM INTERNE DE LA RELATION INVERSEE
                                        POUR DES BALAYAGES INDIRECTS
                                        */

  PTRELIN         :       ADRESS          ; /* POINTEUR DE PARCOURS DE LA RELATION
                                               INVERSEE (POINTANT SUR UNE FEUILLE
                                               DU B-ARBRE. */
  ELEMFE  :       INTEGER                 ; /* POSITION DE L'ELEMENT DANS LA FEUILLE
                                               DU B-ARBRE */
  INDIR   :       CHAR2           ; /* MARQUE D'INDIRECTION POUR L'ATTRIBUT ARG */
  ARG     :       ARGMT           ; /* ARGUMENT DE BALAYAGE */
END ;

TUPDES  =       /* LE TUPLE  DESCRIPTEUR DE LA BASE DE DONNEES */
RECORD
  NOMBAS  :       NAME    ;       /* NOM DE LA BASE */
  IDEABD  :       NAME    ;       /* IDENTIFICATION DE L'ADMINISTRATEUR */
  DISQUE      :       NAME    ;       /* NOM DU DISQUE ALLOUANT LA B. DE D.  */
  UIC         :       NAME    ;       /* IDENTIFICATION POUR LE SYSTEME RSX-11M  */
  PTRPGL  :       INTEGER     ;           /* PREMIERE PAGE DE LA LISTE D'ESPACE DISPONIBLE */
  TAIBAS  :       INTEGER                 ;   /* TAILLE DE LA B. DE D. EN NB. DE PAGES. */
END ;
```

```
TUPLE   =
RECORD
  CASE              TYPREL          OF

       MAITRE  :
       ( TMAI           :       TUPMAI ) ;
       ATTR    :
       ( TATT           :       TUPATT ) ;
       INX     :
       ( TINX           :       TUPINX ) ;
       ATTINX  :
       ( TAIX           :       TUPAIX ) ;
       CURACT  :
       ( TCAC           :       TUPCAC ) ;
       BASE    :
       ( TBASE          :       ARRAY °1..LNMTPL§ OF CHAR ) ;

       BENT    :  /* REDEFINITION ENTIERE D'UN TUPLE D'UNE RELATION DE BASE. */
       ( TBENT          :       ARRAY °1..DEMITPL§ OF INTEGER ) ;

       BREEL   :  /* REDEFINITION REELLE D'UN TUPLE D'UNE RELATION DE BASE. */
       ( TBREEL         :       ARRAY °1..DEMITPL§ OF REAL ) ;
       INVER   :  /* REDEFINITION D'UN TUPLE POUR LES RELATIONS INVERSES */
       (TADR   :                ARRAY °1..QRTPL§ OF ADRESS  ) ;
       DESCRI  :
       ( TDES           :       TUPDES ) ;
       CARELA  :
       ( TCRE           :       DESCREL ) ;
       CARATT  :
       ( TCAT           :       DESCATTS)
END ;
/* LISTE D'ADRESSES : */
/*
 LSTADR  =
 ARRAY °1..MNBATT§ OF ADRESS ;
 */
```

```
/*************************************************************/
/*                                                           */
/*          VARIABLES  MIMER                                 */
/*                                                           */
/*************************************************************/

Z       :       /* ZONE DE PAGINATION */
RECORD
  COMLRU  :       INTEGER         ;   /* compteur pour "Least Recently Used" */
  INFBUF  :       ARRAY %1..NBUFF% OF INFB ;
  BUFFS   :       ARRAY %1..NBUFF% OF PAGE
END ;


FILNAM          : ^INTEGER  ;
DNML            : INTEGER   ; /* DEVICE NAME DESCRIPTOR */
DNM             : ^NAME     ;
DINML           : INTEGER   ; /* DIRECTORY DESCRIPTOR   */
DINM            : ^NAME     ;
FINML           : INTEGER   ; /* FILE NAME DESCRIPTOR   */
FINM            : ^NAME     ;

BKVB,BKVB1,W    : INTEGER       ;
ABKVB           : ^INTEGER      ; /* ADDRES OF BKVB */

DESUTIL         : /* DESCRIPTEUR DE L'UTILISATEUR COURANT DE LA B. DE D. */
RECORD
  DNOMBAS     :     NAME    ;
  DUTLTEUR    :     NAME    ;
  DDISQUE             :     NAME    ;
  DUIC        :             NAME
END ;
```

ANNEXE4. La syntaxe des primitives MIMER (implémentation PASCAL).

A1) CREER-FICHIER;

```
procedure CF (var NOMFICH, DISQUE, UIC : NAME ; TAIBAS:
integer; var CE : integer);
```

A2) DETRUIRE-FICHIER;

```
procedure DF;
```

A3) OUVRIR-FICHIER;

```
procedure OVF (var NOMBASE, DISQUE, UIC : NAME; var CE:
integer);
```

A4) FERMER-FICHIER;

```
procedure FF;
```

A5) LIRE-BLOC;

```
procedure LB ( var BUF : PAGE; CLEBLC : integer);
```

A6) ECRIRE-BLOC;

```
procedure EB (var BUF : PAGE; CLEBLC : integer);
```

B1) LIRE-TUPLE;

```
procedure LT (IDT : ADRESS; LNTUP : integer; var T :
TUPLE; var CE : integer);
```

B2) ECRIRE-TUPLE

```
procedure ET (IDT : ADRESS; LNTUP : integer; var T :
TUPLE; var CE : integer);
```

B3) TROUVER-PLACE-POUR-TUPLE;

```
procedure PL (var IR1, IDT : ADRESS; var CE :
integer);
```

B4) LIRE-PAGE;

```
procedure LP (NROPAG : integer; var NROBUF, CE :
integer);
```

B5) DEMANDER-PAGE-LIBRE;

```
procedure DP ( var NROPAG, CE : integer);
```

B6) LIBERER-PAGE;

```
procedure FP (NROPAG : integer; var CE : integer);
```

B7) FORMATER-PAGE;

```
procedure FG (NROPAG : integer; var M : TUPLE; var
CE: integer);
```

C1) CREER-BASE;

```
procedure CB (NOMBAS, ABD, DISQUE, UIC : NAME; TAIBAS :
integer; var CE : integer);
```

C2) DETRUIRE-BASE;

```
procedure DB (NOMBAS, ABD, DISQUE, UIC : NAME; var CE :
integer);
```

C3) OUVRIR-BASE;

```
procedure OB (NOMBAS, UTLTEUR, DISQUE, UIC : NAME; var
CE: integer);
```

C4) FERMER-BASE;

```
procedure FEB;
```

C5) OBTENIR-CARACTERISTIQUES-BASE;

```
procedure OCB ( var DIM : integer; var TABL : TABCR;
var CE : integer);
```

C6) CREER-RELATION

```
procedure CR (R1 : EXNAME; var IR1 : ADRESS; TYP :
CHAR2; var CE : integer);
```

C7) DETRUIRE-RELATION;

```
procedure DR (var IR1 : ADRESS; var CE : integer);
```

C8) VIDER-RELATION;

```
procedure VI ( var IR1 : ADRESS; var CE : integer);
```

C9) DEFINIR-STOCKAGE;

```
procedure DS ( var IR1 : ADRESS; TYP : CHAR2; var CE:
integer);
```

C10) OBTENIR-CARACTERISTIQUES-RELATION;

```
   procedure OCR ( R1 :  EXNAME;  var CREL  :  TUPLE;  var
   CE:  integer);

C11) GROUPER-RELATION;
   procedure GR ( var IR1 :  ADRESS;  ORDONN :  CHAR2;  var
   LSTATT :  LSTADR;  var CE :  integer);

C12) CREER-ATTRIBUT;
   procedure CA ( var IR1 :  ADRESS;  A1  :  EXNAME;  var
   IA1:  ADRESS;  TYPA :  CHAR2;  LNAT :  integer;  var CE :
   integer);

C13) DETRUIRE-ATTRIBUT;
   procedure DA ( var IR1 :  ADRESS ;  A1  :  EXNAME;  var
   CE:  integer);

C14) OBTENIR-CARACTERISTIQUES-ATTRIBUT;
   procedure OCA ( var IR1 :  ADRESS;  A1  :  EXNAME;  var
   CATT:  TUPLE;  var CE :  integer);

C15) OBTENIR-CARACTERISTIQUES-ATTRIBUTS;
   procedure OCAS ( var IR1 :  ADRESS;  var TABL  :  TABCA;
   var CE :  integer);

C16) CREER-BALAYAGE;
   procedure BL ( var IR1, CURSEUR  :  ADRESS;  var CE  :
   integer);

C17) CONDITION-BALAYAGE;
```

```
    procedure CD ( var CURSEUR : ADRESS; OPERLOG : CHAR2;
    var IA1 : ADRESS; OPERCP : CHAR2; var XARGMT :
    TUPLE; var CE : integer);

C18) ANNULER-CONDITIONS-DE-BALAYAGE;
    procedure AC ( var CURSEUR : ADRESS; var CE :
    integer);

C19) TUPLE-SUIVANT;
    procedure TS ( var CURSEUR : ADRESS; var T : TUPLE;
    var CE : integer);

C20) FIN-RELATION;
    function FR ( var CURSEUR : ADRESS; var CE : integer):
    boolean;

C21) RECOMMENCER-BALAYAGE;
    procedure RB ( var CURSEUR : ADRESS; var CE :
    integer);

C22) DETRUIRE-TUPLE;
    procedure DT ( var CURSEUR : ADRESS; var CE :
    integer);

C23) REMPLACER-ATTRIBUT;
    procedure RA ( var CURSEUR, IA1 : ADRESS; var VALEUR :
    TUPLE; var CE : integer);

C24) COMPARER;
```

```
function COMP ( var CURSEUR, IA1 : ADRESS; OPER :
CHAR2; var XARGMT : TUPLE; var CE : integer) :
boolean;
```

C25) FIN-BALAYAGE;

```
procedure FL ( var CURSEUR : ADRESS; var CE :
integer);
```

C26) CREER-RELATION-INVERSE;

```
procedure CX ( var IR1 : ADRESS; RX : EXNAME; var
IR2: ADRESS ; ORDONN : CHAR2; TYINX : CHAR2; var
LSTATT : LSTADR; var CE : integer);
```

C27) OBTENIR-CARACTERISTIQUES-INDEX;

```
procedure OCX ( var IR1 : ADRESS; NBINX : integer;
var TABX : TABINX; var CE : integer);
```

C28) INITIAL;

```
procedure INITIAL;
```

C29) INSERER-TUPLE;

```
procedure IT ( var IR1 : ADRESS; XTUPLE : TUPLE;
LNTUP : integer; var CE : integer);
```

ANNEXE5. Les codes d'erreur de MIMER

Chaque primitive est munie du paramètre de sortie CE. La valeur de celui-ci peut se trouver dans l'un des intervalles suivants :

000 Tout est bien terminé. Pas d'informations sur la primitive qui est terminée d'une façon satisfaisante.

001-100 Tout est bien terminé. Chaque code correspond à un message portant sur la façon dont la primitive est terminée.

101-200 Avertissements ("Warnings"). Pas grave.

201-300 Erreur grave. La primitive n'a pas pu être exécutée.

301-400 Erreur fatale. La primitive ayant commencée à s'exécuter, s'est arretée. Attention: la base peut être incohérente.

002 Si lors d'une demande de lecture d'une page, elle se trouve déjà en mémoire centrale.

004 Si lors d'une lecture d'une page, une autre page en mémoire centrale a été remplacée.

201 Un idt (p,d) ayant d en dehors des bornes de la page.

202 On veut insérer des tuples dans une relation qui n'a pas encore d'attributs définis.

203 Un idt (p,d) ayant p en dehors de l'intervalle

[1,taille-de-la-bd].

204 La Base de Données est pleine. C'est-à-dire la liste d'espace disponible est nulle.

205 Si lors d'une insertion d'un tuple dans une relation ayant un index primaire, cette insertion ne garantit pas l'unicité des clés.

206 On veut associer à une relation, un type différent de 'B', 'T', ou 'I'.

207 La longueur d'un tuple d'une relation doit être inférieur ou égale à la constante LNMTPL (cf Annexe2).

208 On veut associer à un attribut, un type différent de 'I', 'R' ou 'C'.

209 Le nom externe de l'attribut qu'on veut créer, existe déjà dans la relation.

210 L'idt (p,d) correspond à un trou.

211 Le nom externe de la relation qu'on veut créer, il existe déjà dans la Base de Données.

212 La relation n'existe pas dans la Base de Données.

213 Un essai d'insertion d'un tuple dans une relation qui n'est pas de base.

215 La relation que l'on veut balayer n'a pas de pages affectées.

216 Le curseur n'a pas été crée au préalable.

217 Il n'est pas possible de remplacer un attribut faisant partie d'une clé d'un index.

218 Le nombre de conditions de balayage depasse le seuil fixé par la constante MNBATT (cf Annexe 2).

219 La longueur total de la chaîne de caractères

représentant l'argument de balayage depasse le seuil fixé par la constante LNMTPL (cf Annexe 2).

220 Un essai de détruire un attribut n'existant pas dans la relation.

221 Il n'est pas permis de détruire un attribut faisant partie de la clé d'un index.

222 On ne peut détruire que d'attributs appartenant aux relations de base.

223 On ne peut pas détruire l'attribut témoin (EX) d'une relation.

224 L'utilisateur n'a pas le droit à détruire la Base de Données.

225 La description de la Base de Données donnée par l'utilisateur ne correspond pas à celle de la Base de Données qu'il veut détruire.

226 Le Base de Données n'a pas pu être ouverte.

227 L'attribut n'existe pas dans la relation.

228 Les relations de type T (interne) ne peuvent pas être détruites.

229 La Base de Données n'a pas pu être créée.

ANNEXE 6. UNE SESSION MICROBE CENTRALISE

```
>RUN MICTST

    __ Bienvenue chez MICROBE __

>>>Créer, détruire, ouvrir la base ou FIN ; <<<
```

CREATION D'UNE BASE

```
MIQ>CB;


?...nom de la base : PLAN
?...votre nom : FERNANDEZ
?...nom de disque ou la base se trouve : DK2:
?...votre UIC : °330,145
?..la taille de la base : 40
.......La base PLAN          est creee.

>>>Créer, détruire, ouvrir la base ou FIN ; <<<
```

OUVERTURE D'UNE BASE

```
MIQ>OB;


?...nom de la base : PLAN
?...votre nom : FERNANDEZ
?...nom de disque ou la base se trouve : DK2:
?...votre UIC : °330,145
.......La base PLAN          est ouverte.

>>> Nouvelle requete MIQUEL ou fermer la base <<<
```

CREATION DES RELATIONS DEPT (INDIC, NOMDEP, REGION) ET VILLE (INDEP, NOMVIL)

```
MIQ>CR  DEPT :   INDIC (ENTIER), NOMDEP (CHAR(10)), REGION (CHAR(12)) ;

?...Voulez-vous l'affichage de l'arbre (O/N):N


.......La relation DEPT   est cree.
.......L'attribut INDIC  est cree.
.......L'attribut NOMDEP est cree.
.......L'attribut REGION est cree.

>>> Nouvelle requete MIQUEL ou fermer la base <<<

MIQ>CR  VILLE :  INDEP (ENTIER), NOMVIL (CHAR(12)) ;

?...Voulez-vous l'affichage de l'arbre (O/N):N
```

```
.......La relation VILLE  est cree.
.......L'attribut INDEP  est cree.
.......L'attribut NOMVIL est cree.
```

AFFICHAGE DES CARACTERISTIQUES DES RELATIONS DE LA BASE

```
>>> Nouvelle requete MIQUEL ou fermer la base <<<

MIQ>EB ;

?...Voulez-vous l'affichage de l'arbre (O/N)?N
```

caracteristiques des relations de la base :

nom	type	cardinalite	degre	longueur	type de stockage
DEPT	B	0	3	24	
VILLE	B	0	2	14	

AFFICHAGE DES ATTRIBUTS DE CHAQUE RELATION DE LA BASE

```
MIQ>EA DEPT : * ;

?...Voulez-vous l'affichage de l'arbre (O/N)?N
```

caracteristiques des attributs de la relation DEPT :

nom	type	longueur	position
INDIC	E	2	1
NOMDEP	C	10	3
REGION	C	12	13

```
>>> Nouvelle requete MIQUEL ou fermer la base <<<

MIQ>EA VILLE : * ;

?...Voulez-vous l'affichage de l'arbre (O/N)?N
```

caracteristiques des attributs de la relation VILLE :

nom	type	longueur	position
INDEP	E	2	1
NOMVIL	C	12	3

INSERTION DE TUPLES DANS LES RELATIONS

```
>>> Nouvelle requete MIQUEL ou fermer la base <<<

MIQ>INSERT_INTO  DEPT;

?...Voulez-vous l'affichage de l'arbre (O/N):N


***** ATTENTION !!! *****
   Pour chaque tuple, le systeme vous demandera la valeur de ses attributs.
   Si vous voulez inserez la valeur indefinie, tapez "/".

        1 eme tuple?
 NOMATT:INDIC , TYPE:E , LONGUEUR:  2, VALEUR:03
 NOMATT:NOMDEP, TYPE:C , LONGUEUR: 10, VALEUR:ALLIER
 NOMATT:REGION, TYPE:C , LONGUEUR: 12, VALEUR:AUVERGNE
...Le      1 eme tuple a ete insere.
?...Insertion d'un autre tuple (O/N) O
        2 eme tuple?
 NOMATT:INDIC , TYPE:E , LONGUEUR:  2, VALEUR:38
 NOMATT:NOMDEP, TYPE:C , LONGUEUR: 10, VALEUR:ISERE
 NOMATT:REGION, TYPE:C , LONGUEUR: 12, VALEUR:RHONE-ALPES
...Le      2 eme tuple a ete insere.
```

QUELQUES INSERTIONS APRES .....

LISTER TOUS LES TUPLES DE LA RELATION DEPT

```
MIQ>SELECT *
MIQ>FROM  DEPT ;
......Le resultat de votre requete.
+================================+
! INDIC   NOMDEP       REGION     !
+================================+
!       3  ALLIER       AUVERGNE   !
+--------------------------------+
!      15  CANTAL       CORSE      !
+--------------------------------+
!      26  DROME        RHONE-ALPES!
+--------------------------------+
!      38  ISERE        RHONE-ALPES!
+--------------------------------+
!      39  JURA         FRANCHE-COMT!
+--------------------------------+
!      42  LOIRE        RHONE-ALPES!
+--------------------------------+
!      18  CHER         CENTRE     !
+--------------------------------+
!      20  CORSE        CORSE      !
+--------------------------------+
!      10  AUBE         CHAMPAGNE  !
+--------------------------------+
!       2  AISNE        PICARDIE   !
+--------------------------------+
nombre de tuples resultat :      10
```

A QUELLE REGION APPARTIENT LA VILLE GRENOBLE ?

```
MIQ>SELECT REGION
MIQ>FROM   DEPT
MIQ>WHERE  INDIC = SELECT INDEP
MIQ>               FROM   VILLE
MIQ>               WHERE  NOMVIL = "GRENOBLE" ;

?...Voulez-vous l'affichage de l'arbre (O/N):O

        Parcours recursif de l'arborescence.
        ====================================

      indice          noeud                 type          fils      frere

       1------>       project               operateur        2          0
       2------>       join                  operateur       15          3
      15------>       DEPT                  relation         0          5
       5------>       project               operateur        6         16
       6------>       select                operateur        8          7
       8------>       VILLE                 relation         0         14
      14------>       egal                  connecteur      12          0
      12------>       NOMVIL                domaine          0         13
      13------>       'GRENOBLE'            constante        0          0
       7------>       INDEP                 domaine          0          0
      16------>       egal                  connecteur      17          0
      17------>       INDIC                 domaine          0         18
      18------>       INDEP                 domaine          0          0
       3------>       REGION                domaine          0          0

      ia   =  19      irel   =    3
      idom =   5      iconst =    2


......Le resultat de votre requete.

+==============+
! REGION       !
+==============+
! RHONE-ALPES  !
+--------------+
nombre de tuples resultat :       1
```

LISTER TOUTES LES VILLES DU DEPARTEMENT 38

```
MIQ>SELECT  NOMVIL
MIQ>FROM    VILLE
MIQ>WHERE   INDEP = 38 ;
```

```
.....Le resultat de votre requete.

 +==============+
 ! NOMVIL       !
 +==============+
 ! GRENOBLE     !
 +--------------+
 ! VOIRON       !
 +--------------+
 ! VIENNE       !
 +--------------+
 ! CHAMBERRY    !
 +--------------+
 nombre de tuples resultat :      4
```

LISTER TOUTES LES VILLES DE LA REGION RHONE-ALPES

```
MIQ>SELECT  NOMVIL
MIQ>FROM    VILLE
MIQ>WHERE   INDEP = SELECT  INDIC
MIQ>                FROM    DEPT
MIQ>                WHERE   REGION = "RHONE-ALPES" ;

?...Voulez-vous l'affichage de l'arbre (O/N):N

.....Le resultat de votre requete.

 +==============+
 ! NOMVIL       !
 +==============+
 ! VALENCE      !
 +--------------+
 ! ROMANS       !
 +--------------+
 ! TOURNON      !
 +--------------+
 ! MONTELIMAR   !
 +--------------+
 ! GRENOBLE     !
 +--------------+
 ! VOIRON       !
 +--------------+
 ! VIENNE       !
 +--------------+
 ! CHAMBERRY    !
 +--------------+
nombre de tuples resultat :      8
```

CREER UN NOUVEL ATTRIBUT DANS LA RELATION VILLE

```
MIQ>CA  VILLE : POPULA (ENTIER) ;

?...Voulez-vous l'affichage de l'arbre (O/N):N

.......L'attribut POPULA est cree.
```

AFFICHAGE D'ATTRIBUTS DE LA RELATION VILLE.

```
MIQ>EA VILLE : * ;

?...Voulez-vous l'affichage de l'arbre (O/N):N
```

caracteristiques des attributs de la relation VILLE :

nom	type	longueur	position
INDEP	E	2	1
NOMVIL	C	12	3
POPULA	E	2	15

LISTER TOUS LES TUPLES DE LA RELATION VILLE

```
MIQ>SELECT  *
MIQ>FROM    VILLE ;

?...Voulez-vous l'affichage de l'arbre (O/N):N

......Le resultat de votre requete.
```

INDEP	NOMVIL	POPULA
38	GRENOBLE	??????
38	VOIRON	??????
38	VIENNE	??????
38	CHAMBERRY	??????
26	VALENCE	??????

!	26 ROMANS	?????? !
!	2? TOURNON	?????? !
!	3? MONTELIMAR	?????? !
!	20 BASTIA	?????? !
!	20 BONIFACIO	?????? !
!	20 AJJACCIO	?????? !
!	20 CALVI	?????? !
!	15 AURILLAC	?????? !
!	15 MAURIAC	?????? !
!	18 VIERZON	?????? !
!	18 BOURGES	?????? !

nombre de tuples resultat : 16

DESTRUCTION D'UN ATTPIBUT DE LA RELATION VILLE

MIO>KILL_ATTRIBUT VILLE : POPULA :

?...Voulez-vous l'affichage de l'arbre (O/N):N

.....L'attribut POPULA est detruit.

AFFICHAGE D'ATTRIBUTS DE LA RELATION VILLE

MIO>EA VILLE : * :

?...Voulez-vous l'affichage de l'arbre (O/N):N

caracteristiques des attributs de la relation VILLE :

nom	type	longueur	position
INDEP	E	2	1
NOMVIL	C	12	3

DESTRUCTION DE LA RELATION VILLE

```
MIQ>KILL_RELATION VILLES ;

MIQ>..Erreur:la relation nommée n'existe pas. (ce=29)
MIQ>..ligne   :          1
MIQ>..symbole :      VILLES

>>> Nouvelle requete MIQUEL ou fermer la base <<<

MIQ>KILL_RELATION VILLE ;

?...Voulez-vous l'affichage de l'arbre (O/N):N

.......La relation VILLE  a ete supprimee.
```

FERMETURE DE LA BASE

```
>>> Nouvelle requete MIQUEL ou fermer la base <<<
MIQ>FB ;


OPE>La base est fermee.
```

FIN DE SESSION MICROBE

```
>>>Créer, détruire, ouvrir la base ou FIN ; <<<
MIQ>FIN ;


...MICROBE vous  remercie et à très bientot ! ...
```

BIBLIOGRAPHIE

<ADE78> E.ANDRE, P.DECITRE

"On providing distributed application programmers with control over synchronization"

Computer Network Protocol, Symposium Liège (Belgium), Fev. 78

<ADE80> E.ANDRE, P.DECITRE

"Projet POLYPHEME: The DEM Distributed Execution Monitor"

International Symposium on Distributed Data Bases, Paris, Mars 80 North-Holland Pub. Co

<ADI80a> M.ADIBA, J.M.ANDRADRE, P.DECITRE, F.FERNANDEZ, G.T.NGUYEN

"POLYPHEME: An experience in distributed database system design and implementation"

International Symposium on Distributed Data Bases, Paris Mars 80

North-Holland Pub.Co

<ADI80b> M.ADIBA, J.M.ANDRADE, F.FERNANDEZ, G.T.NGUYEN

"An overview of the POLYPHEME distributed Data Base Management System"

IFIP Conference, Melbourne, October 1980

<ANB78> N.ANDERSON, N.BURKHARD

"MINISEQUEL : Relational Data Management System"
Database: Improving usability and responsiveness
1978 Univ. of California, San Diego, Août 1978

<AND78> J.M.ANDRADE
"LAMB Specifications générales" Projet POLYPHEME
Note technique Nro.26 Juin 1978

<AND80> J.M.ANDRADE
"L'integrité et la mise à jour dans un système de Bases de Données réparties." Projet POLYPHEME
Thèse de Docteur ingénieur, INPG, Laboratoire IMAG, Nov. 1980

<AZR81> F.AZROU
"MIDEC, Un algorithme de décomposition de requêtes dynamique décentralisé" projet MICROBE
Rapport de D.E.A.,INPG, laboratoire IMAG, juin 1981

<BAM72> R.BAYER, E.MCCREIGHT
"Organization and maintenance of large ordered indexes"
Acta Informatica 1(3),1972

<BAN80> F.BANCILHON, M.SCHOLL
"Le filtrage des données dans la machine Base de Données VERSO"
Journées machines Bases de Données, Sept 80, Sophia Antipolis (France)

<BAY72> R.BAYER, E.MACCREIGHT

"Organization and maintenance of large ordered indexes"

Acta informatica 1, 173,189 (1972)

<BAY76> R.BAYER, M.SCHKOLNICK

"Concurrency of operations on B-trees"

IBM Research Laboratory RJ 1791 (Nro.25863) may 27, 1976

<BAY77> H.WEDEKIND

"Prefix B-trees"

TODS of ACM, Vol.2, No.1 March 1977 pp 11-26

<BCD73> D.BJØRNER, E.F.CODD, K.L.DECKERT, I.L.TRAIGER

"The GAMMA-0 n-ary Relational Database interface specification of objets and operations"

IBM Research Laboratory San Jose, CALIFORNIA

RJ 1200(Nro.19229) April 1973

<BEN81> A.BENSAID

"Conception d'un commutateur de processus réseau pour la mise en oeuvre d'applications reparties"

Mémoire d'ingénieur, Centre d'études et de recherches en informatique, Alger, Stage réalisé au laboratoire IMAG, Sept 81

<BRU80> P.BRUCE BERRA

"A summary of the state of the art in Data Base

Machines"
Journées machines Bases de Données, Sept 80, Sophia Antipolis (France)

<CEL80> CELLARY, MEYER
"Multi-request approach to distributed processing in a relational DBMS"
International Symposium on Distributed Data Bases, Paris Mars 80
North-Holland Pub.Co

<CHA76> M.ASTRAHAM et Al
"System R: Relational Approach to Database Management"
ACM Transactions on Database Systems, Vol.1, No.2 June 1976

<CHU79> W.W.CHU, P.HURLEY
"A model for optimal query processing for distributed databases"
Compcon 79, San Francisco, Fev. 79

<COD70> E.F.CODD
"A relational model of data for large shared data banks"
Comm. ACM Vol.13, Nro.6, Juin 1970

<COD72> E.F.CODD
"Relational Completeness of Data Base Sublanguages"

In Data Base Systems, courant computer science Symposia series, Vol.6, Englewood Cliffs, N.J.: Prentice-Hall, 1972

<COD74> E.F.CODD
"Seven steps to rendez vous with the casual user" proc. IFIP TC-2 Working conference on Data Base management systems
North-Holland April 1974

<COD81> E.F.CODD
"SQL/DS What it means"
Computerworld Fev. 81

<COM79> D.COMER "The ubiquitous B-tree" Computing Surveys, Vol.11, No.2, June 1979

<COR81> Groupe CORNAFION
"Systèmes informatiques répartis"
illustration 5, pp 305 DUNOD informatique

<DAN77> Ng.X.DANG, G.SERGEANT
"Expression of parallelism and communication in distributed network processing"
International Conference on parallel processing
Bellaire, Michigan 1977

<EPS78> EPSTEIN et Al
"Distributed query processing in a relational data

base system"

UCB-ERL, April 78

<FAG79> R.FAGIN, J.NIEVERGELT, N.PIPPENGER, R.STRONG

"Extensible Hashing -A fast acces method for dynamic Files"

TODS, Vol 4, Nro.3, September 1979

<FER79> F.FERNANDEZ

"Etude d'algorithms d'optimisation de requêtes dans les SGBD répartis. Mise en oeuvre de l'optimiseur DOPAGE"

Rapport de D.E.A., Université Scientifique et Médicale de Grenoble, Laboratoire IMAG, Sept. 79

<FER80> F.FERNANDEZ, L.FERRAT, G.T.NGUYEN, G.SERGEANT

"MICROBE: Un système de bases de données relationnelle répartie sur un réseau local de micro-ordinateurs".

Congrès de l'AFCET Nancy (FRANCE) Novembre 1980

<FER81a> F.FERNANDEZ

"MIcro MEmoire Relationnelle (MIMER)"

Rapport de Recherche No.233, Laboratoire IMAG, Janvier 1981

<FER81b> F.FERNANDEZ, L.FERRAT, G.T.NGUYEN, G.SERGEANT

"MICROBIO: Un sistema relacional de bases de datos repartidos en una red local de micro-computadores"

Convención Informática Latina (CIL 81) Barcelone juin 1981

<FER81c> F.FERNANDEZ, M.SARAJLIC
"Une implémentation des B*-arbres dans la mémoire relationnelle MIMER"
Congrès AFCET, Nov. 1981 Gif-sur-Yvette

<FOR78> J.M.FORESTIER
"Machine Relationnelle pour système Segmenté"
Thèse de 3ème. Cycle, INPG, Grenoble; Septembre 1978

<FRI80> L.FERRAT
"Traduction d'un langage de haut niveau en arborescences algébriques" Projet MICROBE
Rapport de D.E.A., Laboratoire IMAG, Sept. 80

<FRI81> L.FERRAT
"Projet MICROBE: guide utilisateur du langage MIQUEL"
Note technique, Laboratoire IMAG, Janvier 1981

<GAL81> H.GALY
"Application de l'intelligence artificielle à l'optimisation de requêtes relationnelles"
Thèse de 3ième. Cycle INPG, Laboratoire IMAG, Nov. 81

<GAR80> G.GARDARIN et Al

"La machine bases de données SABRE: motivations et architecture"

Rapport interne SIRIUS, INRIA, Rocquencourt, Sep.80

<HEL78> G.HELD et Al

"B-trees Re-examined"

CACM, Vol.21 No.2, February 1978

<ING76> M.STONEBRAKER, E.WONG, P.KREPS

"The design and Implementation of INGRES"

ACM Transactions on Database Systems Vol.1, No.3, September 1976

<ISO79> ISO

"Reference Model on Open Systems Interconnexion"

ISO/TC97/SC16/N227, June 79

<KNU74> D.E.KNUTH

"The art of computer programming", Vol.3, Addison Wesley, 1974

<LAN78> G. LE LANN

"Algorithms for distributed data sharing systems which use tickets"

3rd Berkeley worshop on Distributed Data Management, Aug. 78

<LAR78> P.LARSON

"Dynamic Hashing"
BIT 18,1978

<LEE81> Y.J.LEE
"Définition et réalisation dun interpréteur de requêtes dans MICROBE"
Rapport de D.E.A., INPG Laboratoire IMAG Grenoble Sept. 1981

<LIT79> W.LITWIN
"Hachage virtuel: Une nouvelle technique d'adressage de mémoires"
Thèse de doctorat d'état, Université Pierre et Marie CURIE, Paris Mars 1979

<LIT80> W.LITWIN
"Trie Hashing"
Rapport INRIA, projet SIRIUS, référence MAP-I-013, Novembre 1980

<LOC79> P.C.LOCKEMAN ET al
"Data abstractions for Databases Systems"
ACM TODS, Vol 4, Nro 1, March 79

<LOR74> R.A.LORIE
"XRM: An extended (n-ary) Relational Memory"
IBM Cambridge Scientific Center
Technical report Nro.320-2096, January 1974

<MRS79> R.HUDYMA, J.KORNATOWSKI, I.LADD

"Implementing A micro-computer Database Management System"

University of Toronto (Canada) Rapport No.M5S1A1, May 1979

<NAF79> N.NAFFAH, V.QUINT

"Protocole de transport pour réseaux locaux"

Document KAYAK REL 2.504

<NGU77> G.T.NGUYEN

"URANUS: Une approche relationnelle à la cooperation de bases de données"

Thèse de 3ième. Cycle, Université Scientifique et Médicale de Grenoble, Décembre 1977

<NGU79> G.T.NGUYEN

"A unified method for query decomposition and shared information updating in distributed systems"

1st. International Conference on Distributed Computing System

Huntsville (Alabama), October 79

<NGU81a> G.T.NGUYEN, G.SERGEANT

"Distributed Architecture and Decentralized Control for a local network Database System"

ACM International Computing Symposium, London March 1981

<NGU81b> G.T.NGUYEN

"Distributed Query Management for a local Network Database System"

2nd International Conference on Distributed Computing System, Paris April 1981

<RSX11> "Introduction to RSX-11M"

Order No.AA-2555C-TC RX-11M, Version 3.1, DIGITAL,

<SAL80> A.SALES

"Utilisation de la notion de type abstrait générique en bases de données relationnelles"

Congrès AFCET Nancy (France) Nov. 80

<SCH78> J.W.SCHMIDT

"Type Concepts for database definition: An investigation Based on Extension to Pascal"

Int. Conference on Database: Improving usability and responsiveness, Haifa, Israel, August 78

<SDD79> GOODMAN, BERNSTEIN, WONG et Al

"Query processing in SDD-1: a system for distributed databases".

technical report CCA 79-06 Oct. 79

<SER78> "Sytème d'interprétation généralisé orienté réseau"

SIGOR Spécifications de définition

Doc. SIRIUS XEC.I.007, INRIA, août 1978

<SER80> G.SERGEANT, TREILLE

"SER: A system for distributed execution based on decentralized control techniques"

ICCC 80, Atlanta, Oct.80

<SMI79> D.SMITH, J.M.SMITH

"Relational Data Base Machines"

Computer IEEE, march 79, pp 28-38

<SUL78> SU, LUPKIEWICZ, LEE et Al

"MICRONET: A micro-computer network system for managing distributed relational databases"

4th. VLDB Conf. Berlin, Sept. 78

<TIN78> TING, TSICHRITZIS

"A micro DBMS for a distributed database"

4th. VLDB Conf. Berlin, Sept. 78

<VEL81> F.VELEZ

"Une interface relationnelle à une Base de Données SOCRATE"

Rapport de D.E.A., INPG laboratoire IMAG Grenoble, Juin 81

<WED74> H.WEDEKIND

"On the selection of access paths in a data base system"

data base Management North-Holland, Amsterdam 1974

<WIR76> N.WIRTH

"Algorithms + data structures = programs"

Prentice-Hall Inc., Englewood Cliffs, N.J.,1976

Order No. AA-2555C-TC

<WIT78> D.J.DeWITT

"DIRECT -A multiprocessor organization for supporting Relational Database Management System"

IEEE Transactions on computers, Vol.C-28, Nro.6, June 79

<YAO79> S.B. YAO, A.R. HEVNER

"Query processing in distributed database systems"

IEEE Trans. on Software Engineering, May 79

<ZLO77> M.M.ZLOOF

"Query By Exemple: a data base language"

IBM Systems J. 16:4 pp 324-343, 1977

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 5 de l'arrêté du 16 Avril 1974,

VU les rapports de M.[?] ADIBA.....

M. .....

M. .....

M. FERNANDEZ-SANCHEZ Fernando est autorisé

à présenter une thèse en soutenance pour l'obtention du grade de

DOCTEUR

Fait à GRENOBLE, le 17.11.1981

Le Président de l'U.S.M.G.

Le Président,

J.J. PAYAN