Une approche pour optimiser les performanaces dans les SGBD relationnels : projet Microbe

Yoon Joon Lee

▶ **To cite this version:**

Yoon Joon Lee. Une approche pour optimiser les performanaces dans les SGBD relationnels : projet Microbe. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG; Université Joseph-Fourier - Grenoble I, 1983. Français. NNT : . tel-00307391

HAL Id: tel-00307391

https://theses.hal.science/tel-00307391

Submitted on 29 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l' Université Scientifique et Médicale de Grenoble

et à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR

par

LEE Yoon Joon

UNE APPROCHE POUR OPTIMISER LES PERFORMANCES DANS LES SGBD RELATIONNELS. PROJET MICROBE.

Thèse soutenue le 3 novembre 1983 devant la commission d'examen.

G. VEILLON	Président
M. ADIBA	
C. DELOBEL	Examinateurs
R. DEMOLOMBE	
G.T. NGUYEN	

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

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

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

.../...

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

.../...

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

J'exprime ma profonde reconnaissance à :

Monsieur Gérard VEILLON Directeur de l'ENSIMAG qui me fait l'honneur de présider ce jury,

Monsieur Claude DELOBEL professeur à l'USMG 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,

Monsieur Michel ADIBA professeur à l'USMG pour ses suggestions et sa particopation au jury,

Monsieur Robert DEMOLOMBE du CERT-ONERA à Toulouse pour sa participation au jury,

Monsieur Gia Toan NGUYEN responsable du projet MICROBE qui a eu la patience de lire et de critiquer le manuscrit de cette thèse et qui a bien voulu faire partie de ce jury.

Je tiens aussi à remercier mes collègues de travail, notamment ceux de l'équipe MICROBE - ainsi que le service de D. IGESIAS qui a assuré le tirage de cette thèse.

그리고 아버님, 어머님 께, 또한 처에게 감사를 드리며 ...

TABLE DES MATIERES

I. INTRODUCTION

1.1 SGBD relationnel et Performances

1.2 Environnement de l'étude: projet MICROBE

1.3 Objectif et Plan de la thèse

II. METHODES D'ACCES DANS MIMER

2.1 Description de MIMER

2.2 Organisation des relations et Accès aux n-uplets dans les mémoires relationnelles

2.2.1 Organisation en tas de données

2.2.2 Hachage

2.2.3 Indexation

2.3 Choix des B-arbres et Implantation dans MIMER

2.3.1 Notre choix

2.3.2 Implantation de relations index en B-arbre

2.3.2.1 Définition des B-arbres

2.3.2.2 Représentation sous MIMER

2.3.2.3 Coût de stockage

2.3.2.4 Opérations sur un B-arbre

2.3.3 Interfaces d'utilisation

III. OPTIMISATION DU CHOIX D'UN CHEMIN D'ACCES

3.1 Introduction

3.2 Exemples de choix d'un chemin d'accès

3.2.1 INGRES

3.2.2 SYSTEM-R

3.3 Choix d'un chemin d'accès dans MIMER

3.3.1 Définition du problème

3.3.2 Utilisation d'une relation index

3.3.3 Evaluation du coût d'accès

3.3.3.1 Balayage simple

3.3.3.2 Balayage indirect

3.3.3.3 Sélection d'un chemin

3.4 Choix d'un chemin d'accès pour les opérateurs de l'algèbre relationnelle

3.4.1 Les méthodes de jointure dans MICROBE

3.4.1.1 Méthode des boucles imbriquées

3.4.1.2 Méthode de fusion

3.4.1.3 Méthode par index

3.4.2 Comparaison et choix

3.4.2.1 Comparaison

3.4.2.2 Choix d'une méthode de jointure

IV. AMELIORATION DES PERFORMANCES DE L'INTERPRETEUR

4.1 L'interpréteur

4.2 Techniques d'interprétation

4.2.1 Mode global

4.2.2 Mode pipe-line

4.3 Une approche d'interprétation améliorée

V. CONCLUSION

Annexes

Bibliographies

CHAPITRE I

INTRODUCTION

1.1 SGBD relationnel et Performances

CODD a défini en 1970 le modèle relationel, modèle basé sur des notions mathématiques abstraites, en cherchant le maximum d'indépendance logique/physique <COD 70>. Jusqu'à présent, de nombreux produits expérimentaux l'utilisant ont été developpés. Il existe même des systèmes commercialisés: QBE, SQL/DS, INGRES, MRDS, ...

D'ailleurs les modèles de base de données et les langages de manipulation ayant le plus de généralité et d'abstraction intégrées au modèle, s'imposeront sans doute dans l'avenir. Cela vient du fait qu'ils permettent à l'utilisateur de formuler son problème sans se soucier trop de savoir ce qui se passe réellement dans le SGBD. En revanche, cela est réalisé au détriment du temps de réponse, ce qui diminue l'efficacité. Donc, une optimisation automatique d'accès doit être réalisée. D'une part, ceci peut être un avantage pour l'utilisation de langages de haut niveau. D'autre part, il reste la façon dont doit être réalisée l'interface éfficace entre l'utilisateur et la base de données, qui doit avoir suffisamment d'intelligence pour accéder à la base.

Plusieurs études pour rendre efficace les SGBD ont été présentées et se poursuivent à partir du bas niveau, par exemple l'organisation des données et la méthode d'accès rapide, jusqu'au haut niveau, par exemple l'optimisation des requêtes. En particulier, l'approche machine base de données commence à

apparaître grâce au developpement de technologies matérielles nouvelles.

Le problème sera donc pour nous de diminuer le manque d'efficacité des SGBD relationnels inhérent à la généralité et à l'abstraction du modèle. Pour cela, nous allons partir d'un SGBD relationnel MICROBE qui est developpé au laboratoire IMAG.

1.2 Environnement de l'étude: Projet MICROBE

Le projet MICROBE a été initialisé en 1979 <NGU 81>. Il consiste à concevoir et réaliser un SGBD relationnel réparti sur un réseau local DANUBE (projet KAYAK <NAF 79>). La version centralisée de MICROBE tourne depuis juin 1981 sur micro-ordinateur MICRO 1 <NGU 83> comportant une unité centrale LSI 11/23, 128 Koctets de mémoire centrale, des mémoires secondaires sur disques et disquettes et des unités d'entrée/sortie sous le système d'exploitation RSX-11M <RSX 11>. Pour la version répartie, des contraintes d'implantation draconniennes ont été imposées par le matériel et logiciel. Cependant plusieurs études ont mené à l'extension de MICROBE; l'interface graphique (INGRID) <CHA 82>, restructuraction de l'arborescence algébrique <WIN 82>, mécanisme de reprise <PRA 82>, et l'intégrité sémantique <FRT 83>.

MICROBE centralisé que nous mettons au point comporte quatre modules indépendants du point de vue de l'architecture et de leur fonctionnement (figure 1.1);

- module de traduction: Une requête en MIQUEL, langage de haut niveau permettant l'interrogation, la modification et la définition de données sur la base de données, soumise par l'utilisateur est analysée syntaxiquement et sémantiquement, puis transformée en arborescence d'opérateurs relationnels <FRT 80> <FRT 81>.
- module de restructuraction: Une fois la requête traduite par le traducteur (MIQUEL), l'arborescence produite peut

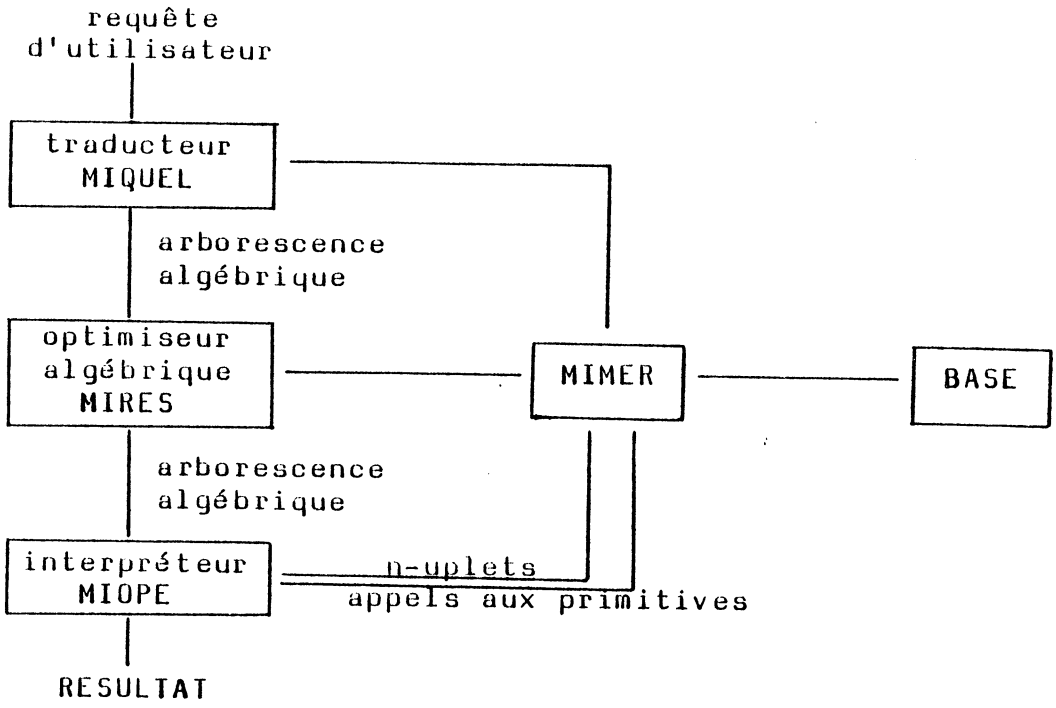


Figure 1.1 Architecture fonctionnelle de MICROBE centralisé

être modifiée en effectuant une optimisation algébrique via des règles prédifinies basées sur les opérateurs de l'algèbre relationnelle <WIN 82> <GAL 83>.

- module d'interprétion: L'arborescence est enfin interprétée et les résultats sont rendus à l'utilisateur grâce à l'évaluation d'un ensemble d'opérateurs de l'algèbre relationnelle en accédant à la base de données par l'intermédiaire des primitives de MIMER <LEE 81>.
- Micro Mémoire Relationnelle (MIMER) est une couche autour des données physiques, elle permet une vision relationnelle des fichiers constituant la base de données en gérant la mémoire centrale et secondaire, et différentes relations de la base de données et en établissant les méthodes d'accès <FER 81a>, <FER 81b>.

1.3 Objectif et Plan de la thèse

Depuis que MICROBE est opérationnel, nous avons constaté le manque d'efficacité, surtout dans le cas des requêtes imbriquées (qui produisent l'opération binaire JOINTURE, INTERSECTION ...). Cela provient des raisons suivantes;

- En ce qui concerne la méthode d'accès, MIMER n'avait que l'accès séquentiel dans l'ordre d'insertion éventuelle de n-uplets d'une relation. Ainsi un moyen rapide d'accès aux données était absent.
- Ceci a provoqué l'absence du choix de chemin d'accès. Par exemple, dans le cas de l'opération de JOINTURE, un seul algorithme en boucle impliquée a été adopté.
- Le mode "pipe-line" de l'interpréteur i.e. traitement n-uplet par n-uplet, même s'il a diminué l'accès à la base de données pour stocker les relations temporaires par rapport au mode global, fait trop d'accès au catalogue pour obtenir les informations concernant les n-uplets.
- Le chargement des procédures en mémoire (le module MICROBE est construit en "OVERLAY" sous RSX-11M), au cours de l'exécution d'une requête, augmente considérablement le temps de réponse. Ceci provient de l'insuffisance de mémoire par rapport à la taille de l'ensemble des procédures (le même problème se pose sous les systèmes d'exploitation utilisant des adresses virtuelles).

Notre objectif est l'optimisation des performances du système MICROBE. Les techniques que nous allons employer ne sont

pas nouvelles. Mais dans l'environnement où nous sommes, nous avons essayé d'aborder une bonne approche pour résoudre les problèmes décrits ci-dessus, en considérant les limites et contraintes matérielles et logicielles.

Le chapitre 2 est consacré à l'étude de la méthode d'accès permettant l'accès rapide aux n-uplets dans une relation. Dans le chapitre 3, nous allons exposer le choix d'un chemin d'accès moins coûteux dans MIMER et dans l'interpréteur. Une façon d'interpréter l'arborescence des opérateurs de l'algèbre relationnelle sera présentée dans le chapitre 4, afin d'augmenter les performances. Dans le dernier chapitre, nous présenterons et discuterons le resultat de notre travail.

CHAPITRE II.

METHODE D'ACCES DANS MIMER

2.1 Description de MIMER

MIMER est le logiciel noyau du système MICROBE permettant d'assurer une "vision" relationnelle des fichiers constituant la base de données <FER 81>. Pour cela il possède les fonctions suivantes:

i) Gérer les mémoires centrale et secondaire. Les relations sont découpées et stockées en pages de taille fixe. L'espace disque doit être alloué ou libéré quand c'est nécessaire (insertion/suppression de n-uplet). Lors de la demande d'accès aux n-uplets d'une relation, les pages les contenant sont amenées dans une zone de pagination en mémoire centrale à l'aide d'un mécanisme de mémoire virtuelle.

ii) Gérer trois types différents de relations dans la base de données; premièrement, les relations internes qui représentent les catalogues et les informations de contrôle de la base de données. Ce sont: la relation MAITRE dont chaque n-uplet est un descripteur de chacune des relations de la base, la relation ATTRIBUT décrivant les attributs des relations, la relation INDEX possédant les n-uplets présentant les index définis sur les relations, la relation ATTRIBUTS-CLE ayant les attributs constituant les clés des index, la relation CURSEUR-ACTIF et la relation ARGUMENT servant à établir le curseur pendant le balayage d'une relation. Les relations du deuxième type sont les relations de base, créées par l'utilisateur, elles servent aux stockage de données. Enfin les troisièmes les relations

index sont des index sur les relations de base et créées à la demande de l'utilisateur, elles permettent d'accèder rapidement à une relation de base en établissant non seulement un adressage associatif sur la relation mais aussi une "vue" ordonnées (ascendante ou descendante) de la relation selon un critère d'ordre fixé par un (ou plusieurs) de ses attributs.

iii) Etablir des méthodes d'accès; l'accès aux n-uplets d'une relation est assuré d'une façon directe en balayant séquentiellement les pages occupées par la relation ou d'une façon indirecte, par l'intermédiaire de la relation index définie préalablement.

La figure 2.1 illustre l'accès à la base de données pour obtenir les informations désirées par l'utilisation des primitives de MIMER.

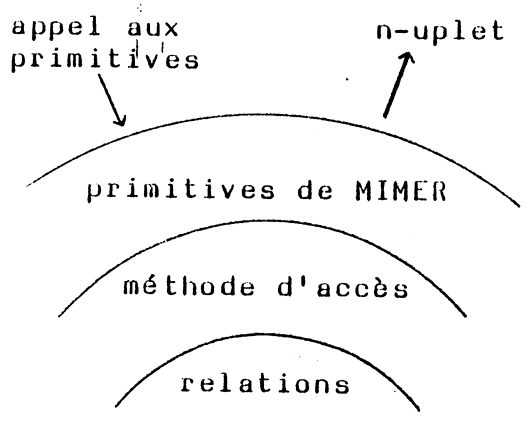


Figure 2.1 Architecture fonctionnelle de MIMER

Avant de parler des méthodes d'accès, nous allons présenter brièvement la structure d'une base de de données de MIMER. La base de données réserve un certain nombre de blocs (de 512 octets chacun) pour stocker physiquement l'ensemble de données sur le disque (ou disquette). Chaque bloc est indentifié par une adresse et correspond à l'unité physique d'entrée/sortie sur disque. Ce bloc peut être aussi équivalent à une page logique, l'unité de stockage de la base de données. Les blocs disponibles sont chaînés entre eux par la fonction de gestion de stockage. Il en est de même des blocs occupés (Figure 2.2 et Figure 2.3).

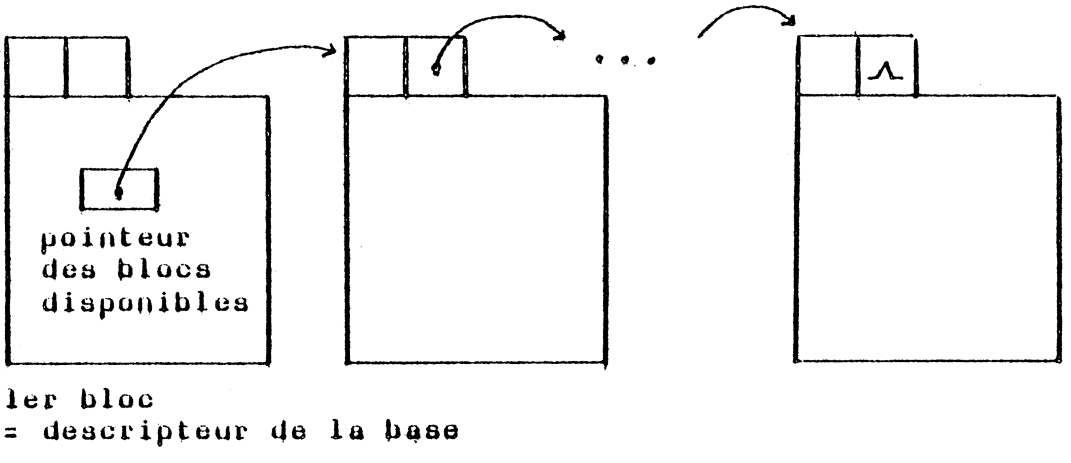


Figure 2.2 Le chaînage des blocs disponibles

Une relation est découpée en pages logiques de taille fixe (508 octets pour stocker les n-uplets, 4 octets de différence entre la page physique et logique sont utilisés pour indiquer le nombre de n-uplets dans la page et le chaînage à la page suivante de la même relation), puis à chaque page logique une transformation associe un bloc (i.e. une page physique) et un seul en mémoire secondaire (voir Figure 2.3).

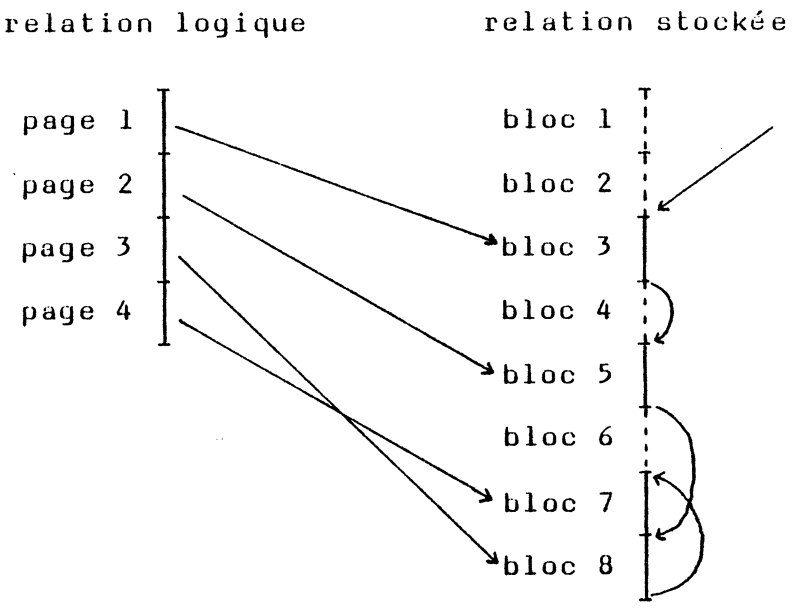


Figure 2.3 Transformation entre la page logique et celle physique

Les n-uplets d'une relation sont stockés physiquement les uns après les autres dans les blocs alloués et se distinguent au moyen d'un identificateur de n-uplet (appelé IDT) formé de l'adresse de la page physique et du déplacement (en nombre de n-uplets) par rapport au début de cette page.

Les exemples cités dans cette thèse sont tirés d'une base de données (figure 2.4) pour une application concernant la gestion des marchandises sur un réseau ferroviaire <DEL 81>. Considérons une relation WAGON (Figure 2.5) et supposons qu'une page ne peut contenir que deux n-uplets. La figure 2.6 nous montre la façon de stocker cette relation et ses n-uplets.

Chaque n-uplet est précédé d'un témoin d'existence de n-uplet (sur 2 octets) pour indiquer si cet emplacement est

disponible ("trou") ou occupé, et sa taille est limitée par celle de la page logique pour ne pas permettre sa fragmentation entre plusieurs pages.

RELATIONS
TRANSPORT(CLIENT, GARE-ORIGINE, GARE-DESTINATION, NO-WAGON, TYPE-MARCHANDISE, POIDS-MARCHANDISE, DATE-CHARGEMENT) RESEAU(GARE-ORIGINE, GARE-DESTINATION, GARE-SUIVANTE, NO-LIGNE) TRAFIC(NO-TRAIN, NO-LIGNE, JOUR) TRAIN(NO-TRAIN, NO-WAGON) WAGON(NO-WAGON, TYPE-WAGON, POIDS-VIDE, CAPACITE, ETAT, GARE)

Figure 2.4 Un ensemble de relations d'une base pour l'application ferroviaire

WAGON

IDT	NO-WAGON	TYPE	POID-VIDE	CAPACITE	ETAT	GARE
s0	1005	plat	3,5	30	chargé	marseille
s1	1051	couvert	10	25	attaché	lyon
s2	2007	citerne	6,5	16	chargé	grenoble
s3	2019	plat	5	15	attaché	tours
s4	2100	plat	10	40	attaché	tours
s5	2200	citerne	5	10	chargé	angers
s6	2308	couvert	10	30	attaché	grenoble
s7	2360	couvert	10	32	chargé	tours
s8	2401	frigo	9	30	libre	narbonne
s9	2212	frigo	8	10	libre	angers
s10	2018	couvert	7	20	libre	paris
s11	1114	frigo	10	15	attaché	angers
s12	1100	citerne	12	22	libre	grenoble
s13	2006	plat	5	35	libre	paris
s14	1001	citerne	5	40	libre	marseille
s15	1010	citerne	5,5	40	chargé	marseille
s16	1105	couvert	10	35	chargé	bordeaux
s17	2220	frigo	10	15	libre	paris
s18	2300	citerne	6,5	16	attaché	grenoble
s19	1112	frigo	10	15	chargé	tours

Figure 2.5 relation WAGON

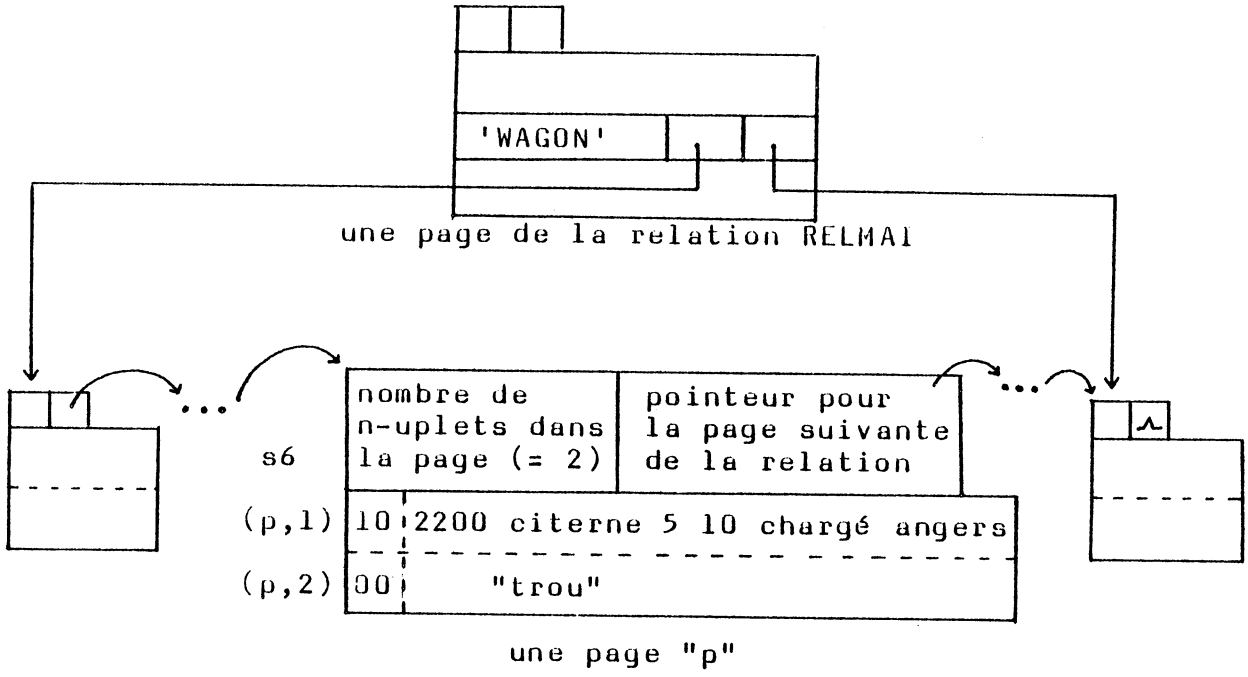


Figure 2.6 stockage de la relation WAGON

2.2 Organisations des relations et Accès aux n-uplets dans les mémoires relationnelles

Lorsque l'on aborde les méthodes d'accès, il est indispensable de montrer l'organisation des données qui peut les supporter. Nous allons donc voir, dans cette partie, les organisations de données adoptées dans les mémoires relationnelles. On peut trouver la synthèse fondanmentale dans <KNU 74>.

L'organisation des relations est importante d'une part pour le stockage des n-uplets d'une relation et d'autre part pour leur manipulation: insertion, suppression, modification et recherche avec une (ou un ensemble de) valeur(s) donnée(s) de l'ensemble des attributs.

2.2.1 Organisation en tas de données

Cette méthode, choisie par MIMER dans un premier temps, par INGRES par défaut, et par SYSTEM-R mais avec un index principal, est la plus simple. Elle consiste à stocker une relation comme une table (voir figure 2.5). En général, l'insertion se fait dans la dernière page, s'il reste une place disponible, sinon dans une nouvelle page. Dans ce cas, aucun ordre n'existe sur les n-uplets stockés. Le témoin d'existence (ou l'indication de suppression) situé dans l'en-tête du n-uplet permet d'indiquer la présence d'un trou.

Le problème se pose dans le cas de la recherche avec une (ou plusieurs) valeur(s) donnée(s). Ceci demande un balayage séquentiel de tous les n-uplets. Cette opération est parfois coûteuse lorsque la relation est dispersée sur de très nombreuses pages. Cependant l'espace occupé peut être maintenu relativement compact par un simple retassage.

2.2.2 Hachage

Le principe classique dans une telle organisation est de répartir les n-uplets dans les ensembles de pages ayant la même valeur calculée par une fonction de hachage qui est appliquée sur la clé du n-uplet. Prenons un exemple dans le SGBD relationnel INGRES; grâce à la commande MODIFY du langage QUEL, il est possible de changer l'organisation d'une relation (du tas par exemple) en une organisation avec hachage:

MODIFY WAGON TO HASH ON NO-WAGON

fait passer le stockage de la relation WAGON du tas vers une organisation avec hachage. La figure 2.7 montre l'organisation de la relation WAGON après l'exécution de cette commande. La fonction de hachage ici est choisie comme NO-WAGON modulo 10 et on suppose aussi que chaque page peut contenir deux n-uplets.

Cette organisation est particulièrement bien adaptée pour répondre efficacement aux gestions du type: "accèder au n-uplet ayant Attr = valeur" où Attr est le constituant clé sur lequel le hachage a été fait <DEL 82>. A ce point là, le hachage a un avantage incontestable au niveau des performances

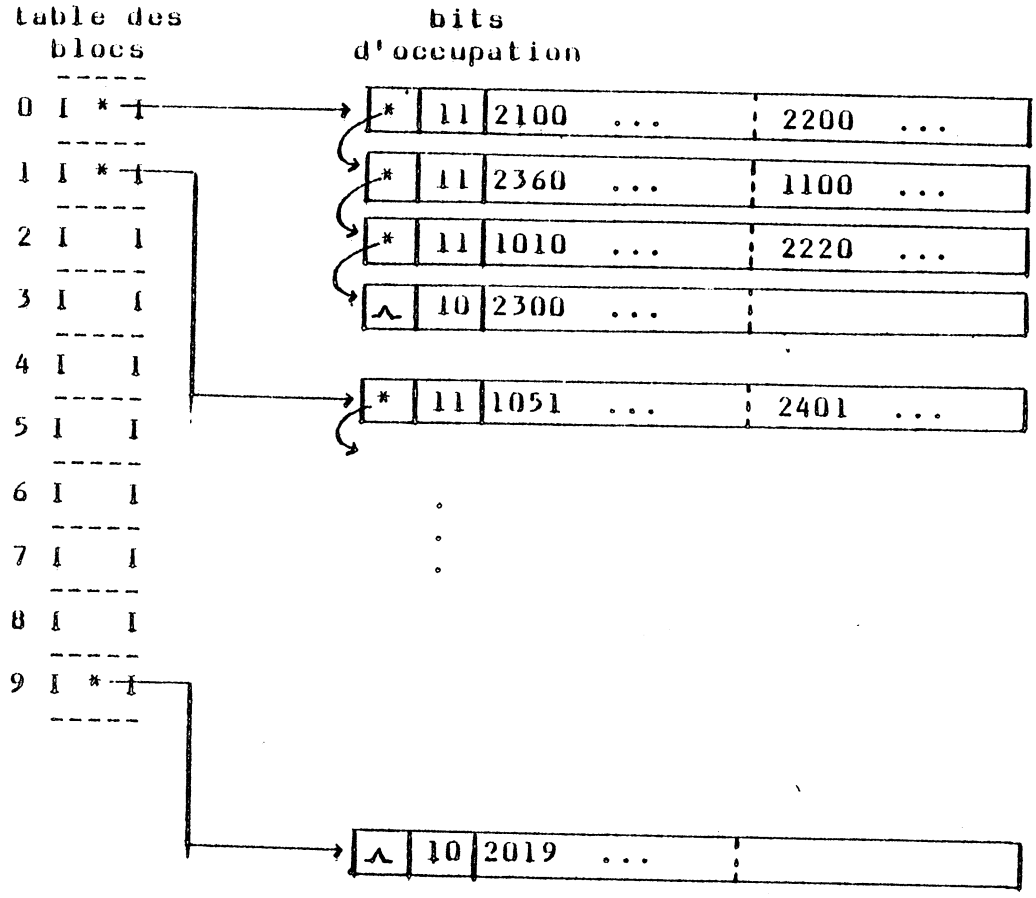


Figure 2.7 La relation WAGON organisée par hachage sur NO-WAGON

algorithmiques et de la gestion de pointeurs.

Cependant l'utilisation du hachage présente les points faibles suivants provenant de son organisation statique:

1) Peu d'insertion peuvent profondément détériorer les performances d'accès.
2) En pratique, le taux de remplissage est fréquemment faible.
3) Le changement de fonction de hachage nécessite un logiciel sophistiqué et peut rendre le fichier indisponible pendant longtemps.

4) Si le hachage devient non-uniforme, de mauvaises performances en découlent.
5) Un éventuel ordre total sur la clé n'est pas conservé par le rangement. Le prédecesseur et le successeur d'un enregistrement peuvent se trouver dans n'importe quelle case.

La plus grande difficulté est présentée par le point 3) (cette opération s'appelle rehachage), elle a été surmontée dans les travaux récents de LITWIN <LIT 79>, LARSON <LAR 78>, FAGIN <FAG 79> qui ont revalorisé le hachage en le pourvoyant de structures de stckage où l'espace d'adressage est géré dynamiquement et n'est pas réorganisé complètement. Les recherches se poursuivent encore pour résoudre les autres difficultés notament le point 5). Ceci mène à l'utilisation des fonctions de hachage dites à distribution calculée et de l'algorithme de "tri-hashing" <LIT 80>.

2.2.3 Indexation

Une autre manière d'organiser une relation en favorisant l'accès sur une clé donnée est de créer un index. Pour cela la relation est triée selon les valeurs de la clé. Ce tri peut se faire selon les valeurs croissantes ou décroissantes de la clé. Alors l'index contient des couples (valeur clé, adresse de page dans le cas d'index creux ou adresse de n-uplet dans celui d'index dense). Cette organisation nous permet de trouver les

n-uplets ayant "Attr = valeur", où Attr est le constituant clé, par une recherche dichotomique. De plus une recherche ayant une condition bornée ou semi-bornée et l'accès ordonné selon la clé sont des points avantageux par rapport aux techniques de hachage.

Dans les paragraphes suivants, nous allons voir les organisations de données pour construire l'index de cette façon. On peut classer cela en deux groupes: tri à enregistrements libres et tri à enregistrements liés. La première méthode, typiquement représentée par ISAM (Indexed Sequentiel Acces Method), est une technique largement employée par tous les constructeurs d'ordinateur. Mais elle pose un problème au niveau du regroupement en pages pour les insertions, car comme dans le cas du hachage, l'insertion peut trouver la page pleine, la technique d'enregistrement-débordements peut alors être employée. Cette méthode est utilisée dans le système INGRES pour organiser une relation selon un index. La commande MODIFY peut encore se charger de passer d'une organisation quelconque à une organisation indexée (cf. figure 2.6 et 2.7):

MODIFY WAGON TO ISAM ON NO-WAGON

La figure 2.7 montre l'effet de cette commande.

L'autre méthode, tri à enregistrements liés, peut être représentée fondamentalement par un arbre binaire. Ceci nous montre que si les clés s'insèrent en ordre aléatoire <KNU 74>, alors le nombre moyen d'accès aux clés est $(Log_2(N))$ où N est le nombre de clés. Toutefois les clés peuvent s'insérer dans n'importe quel ordre. Dans ce cas le nombre d'accès peut aisément

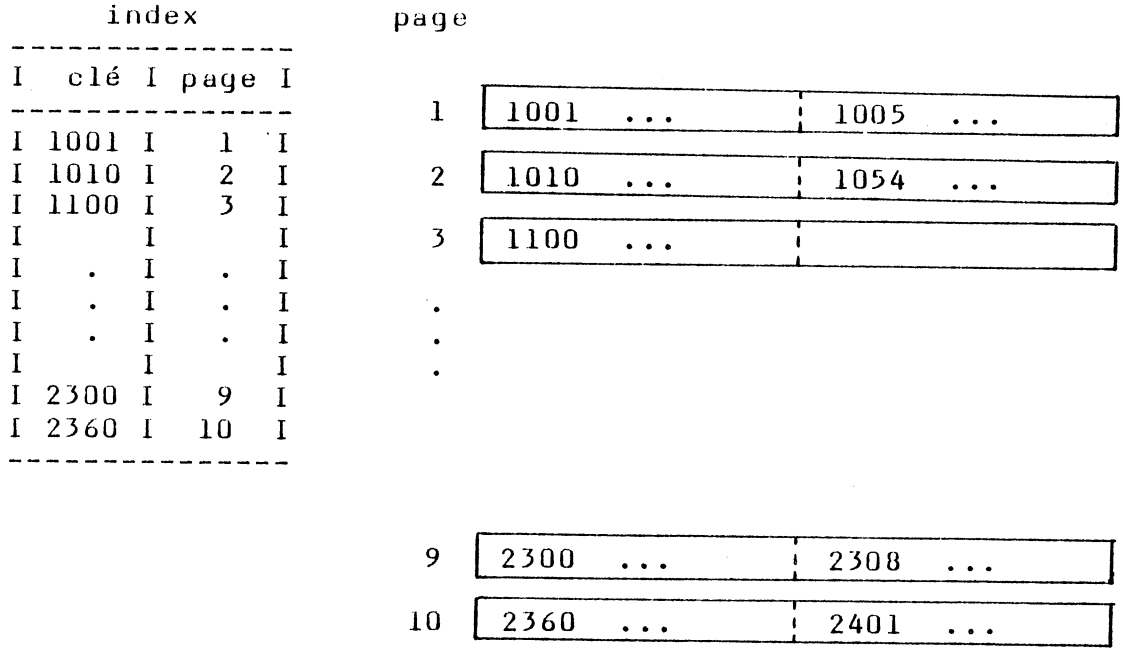


Figure 2.8 WAGON stockée en ISAM sur NO-WAGON

être plus grand que $(Log_2(N))$. Au pire, on risque d'avoir une liste chainée. Pour éviter cet inconvénient, on a introduit le concept de l'arbre binaire équilibré. Cet arbre réduit le nombre d'accès aux clés à $(Log_2(N))$ environ en garantissant que la différence entre les longueurs maximale et minimale de la racine à une feuille de l'arbre est au maximum égale à un.

Pour stocker l'arbre lui-même dans la mémoire secondaire, des arbres n-aires ont été développés. Ces deux derniers éléments conduisent au concept de B-arbre <BAY 72>. Ils ont été ultérieurement étudiés et/ou modifiés dans <BAY 76>, <WIR 76>, <WED 77>, <HEL 78>, <COM 79> et <KWO 82>. Nous en reparlerons plus en détail par la suite. L'augmentation du taux

de remplissage dans les noeuds de B-arbre (2/3 au lieu de 1/2) nous mène aux B*-arbres.

Avant de finir la présentation des index, citons l'organisation appelée "lien" et introduite dans SYSTEM-R <AST 76>. Elle peut être créée entre deux relations pour connecter un n-uplet d'une relation à un ensemble de n-uplets de l'autre. Cette notion est analogue au lien défini dans le modèle CODASYL (liaison 1,N). Ce concept peut aider à l'exécution de certains opérateurs de l'algèbre relationnelle notamment la jointure. Mais la difficulté d'une gestion dynamique de ces liens se pose.

Nous avons décrit les méthodes d'accès utilisées dans les SGBD relationnels. Dans la suite de ce chapitre, nous décrirons notre choix et donnerons sa justification. La figure 2.9 illustre l'état de l'art en ce qui concerne les méthodes d'accès employées dans les SGBD relationnels.

SGBD relationnel	méthodes d'accès
MRDS <MRD 79>	ISAM (B-arbre postérieurement)
MINISEQUEL <ANB 78>	liste séquentielle B-arbre
INGRES <STO 76>	tas avec ou sans hachage compression ISAM
SYSTEM-R <AST 76>	B-arbre(index principal et secondaire) lien
URANUS <NGU 77>	VDAM fourni par SIRIS 8

Figure 2.9 Méthodes d'accès dans les SGBD relationnels

2.3 Choix des B-arbres et implantation dans MIMER

2.3.1 Notre choix

Nous avons vu les méthodes d'accès supportées par les organisations de relations dans les SGBD relationnels. Pour ajouter une autre méthode d'accès dans MIMER, nous devons tenir compte des points suivants:

1) Comme MICROBE est mis en service, un moindre changement de son organisation de relations est souhaitable.
2) Une organisation de données peut servir à plusieurs type d'accès, par exemple l'accès séquentiel ordonné, l'accès associatif etc ...
3) Les performances, non seulement du point de vue de stockage mais aussi de celui des opérations définies sur une telle organisation, doivent être assez bonnes.

Nous avons choisi les B-arbres pour la première implantation de l'index. Cela peut être géré d'une façon indépendente de la méthode d'accès déjà utilisée.

2.3.2 Implantation de relations index en B-arbre

2.3.2.1 Définition des B-arbres

La méthode des B-arbres que nous avons implantée dans MIMER, n'est qu'une "variété" de la structure de données en B-arbre <FER 81>. Si d est un entier positif, un B-arbre d'ordre

d peut être défini avec les propriétés suivantes:

i) Tout noeud a moins de d fils.

ii) Tout noeud, sauf la racine et les feuilles, a plus de (m div 2) fils.

iii) Le noeud racine a au moins 2 fils.

iv) Un noeud non-feuille qui contient (j+1) pointeurs et j clés, se représente comme la figure 2.10-a). Toutes les clés qui se trouvent dans le sous-arbre pointé par Pi-1(Pi) ou 1<=i<=j sont inférieures (égales ou supérieures) à la clé Ki.

v) Les noeuds feuilles qui contiennent les pointeurs vers une relation (IDTs) apparaissent au même niveau (on l'appelle la profondeur du B-arbre) et sont reliés pour assurer l'accès séquentiel dans l'ordre ascendant (descendant) de la clé (figure 2.10-b)).

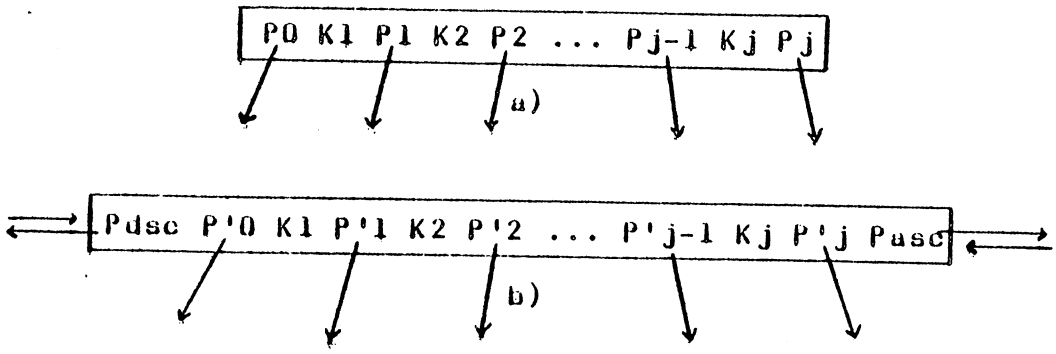


Figure 2.10 Noeuds dans un B-arbre
a) Un noeud non-feuille
b) Un noeud feuille

Nous pouvons constater les caracteristiques suivantes à partir de la définition ci-dessus;

- Un B-arbre est toujours équilibré. Par conséquent le nombre d'accès aux noeuds pour rechercher une clé donnée est

toujours le même (sa profondeur).

- La deuxième propriété nous assure plus de 50% de taux de remplissage d'un noeud.

Finalement l'implantation des B-arbres nous permet deux méthodes d'accès:

i) Accès sélectif: les n-uplets d'une relation de base contenant une clé donnée peuvent être obtenus sans parcours séquentiel.

ii) Accès séquentiel ordonné: grâce au chaînage des feuilles du B-arbre, on peut parcourir la relation de base d'une façon ascendante et descendante selon l'ordre des clés définies.

2.3.2.2 Représentation sous MIMER

Nous avons créé deux Relations internes (cataloguées), appelées RELINX et RELATC, pour le stockage des informations concernant l'index: par exemple la relation de base, ses attributs-clé, les autres relations index définies sur la même relation de base, la clé unique ou non-unique, les caractéristiques du B-arbre etc. En ce qui concerne le stockage du B-arbre lui-même, l'ensemble de ses noeuds correspond à l'ensemble des n-uplets d'une relation (appelée relation index). La différence de ce type de relation par rapport aux autres types est que seul l'accès par pointeur est significatif.

Un noeud de B-arbre représente un n-uplet d'une relation index et est composé de trois parties:

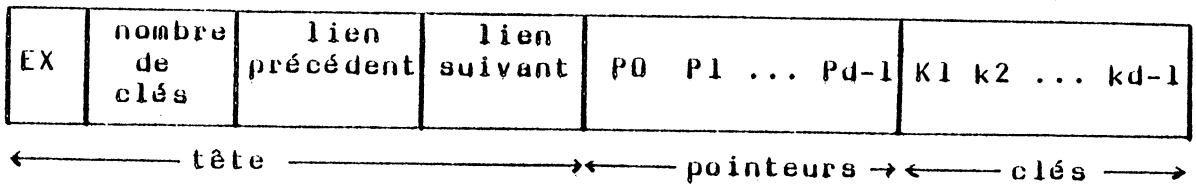


Figure 2.11 Format d'un noeud ayant l'ordre d

L'en-tête contient un témoin d'existence comme pour les n-uplets des autres types de relation, le nombre de clés stockées dans le noeud et les deux pointeurs, suivant (sens d'ordre croissant de la clé) et précédant (sens décroissant) pour les noeuds feuilles. Le pointeur Pi indique soit son sous-arbre dans le cas de noeud non-feuille, soit un n-uplet contenant la clé Ki pour le noeud feuille.

l'ordre d d'un arbre représentant le nombre possible de pointeurs (ou clés) dans son noeud, peut être modifié en fonction de la longueur d'une clé puisque la longueur maximale d'un n-uplet est fixée. Nous admettons qu'une clé peut être composée de plusieurs attributs, nous pouvons obtenir la longueur d'une clé en faisant la somme de la longueur des attributs participant à la clé et appelés attributs-clé:

$$Lc = \sum_{i=1}^{NAC} \text{Longueur}(Ai)$$

Lc : longueur d'une clé
NAC ; nombre des attributs-clé
Ai : ième attribut de clé

La longueur d'un noeud d'un B-arbre est calculée à partir de la longueur d'une clé et son ordre:

$$Ln = Lt + Lp*d + Lc*(d-1)$$

Ln : longueur d'un noeud
Lp : longueur d'un pointeur actuellement représenté par l'IDT (identificateur de n-uplet)
Lt : longueur de la tête fixée à deux octets pour "EX", un entier pour le nombre de clés actuel dans le noeud, et deux pointeurs suivant et précédant

Comme une page de taille constante TUPAG est une unité d'entrée/sortie, nous pouvons formuler le choix de d:

$$d = (TUPAG + Lc - Lt) \text{ div } (Lp + Lc)$$

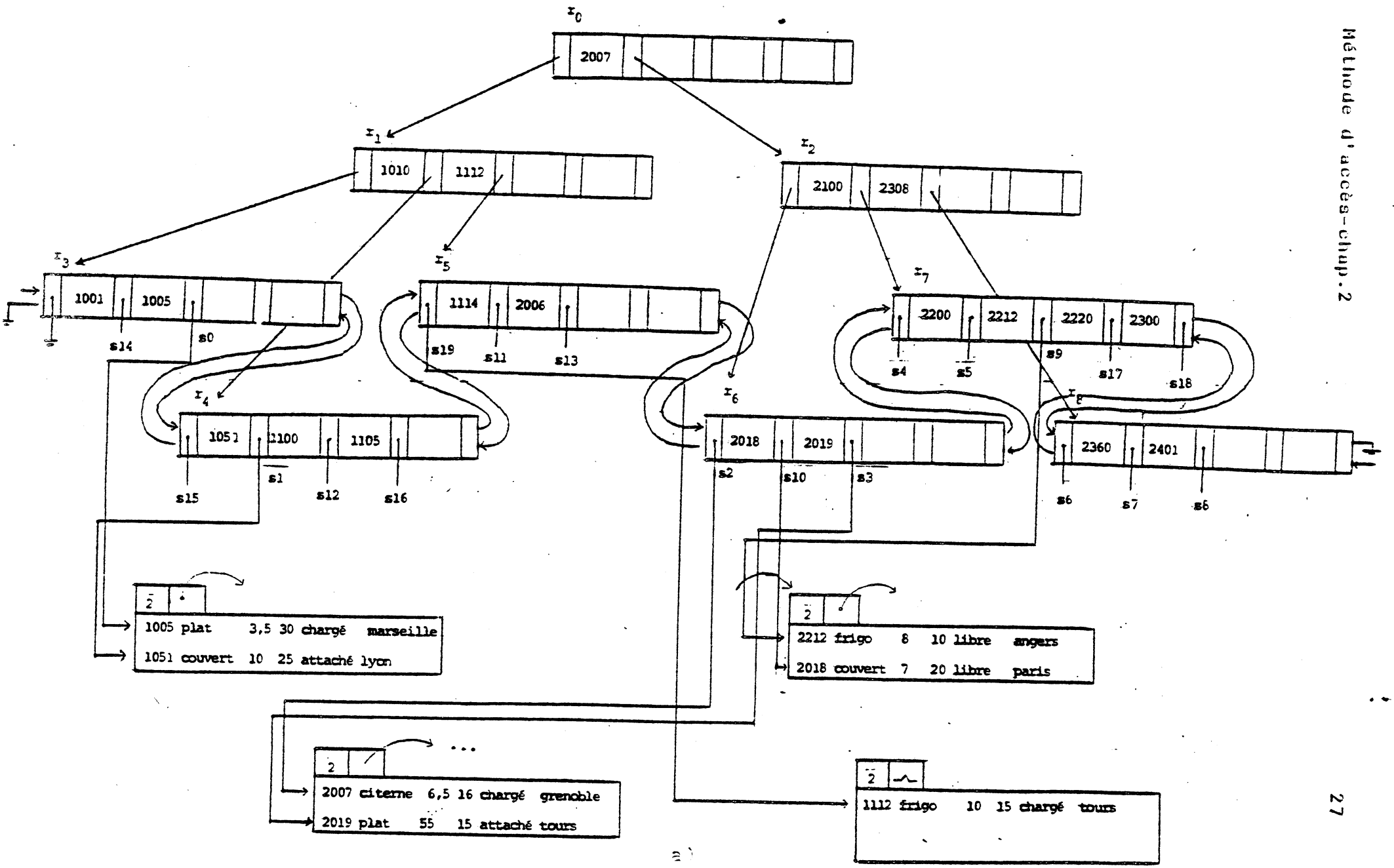
Une fois que le format pour organiser les noeuds est fixé, l'index en B-arbre est construit en ensemble de noeuds comportant les clés. Prenons notre relation WAGON (voir Figure 2.5). Nous allons construire un B-arbre sur l'attribut NO-WAGON en prenant la valeur 5 pour valeur de l'ordre d. La figure 2.12 nous montre le schéma logique du B-arbre et le stockage de la relation index.

INW(tête, pointeurs, cles)

page	tête	pointeurs							clés			
r0	1			r1	r2				2007			
r1	2			r3	r4	r5			1010	1012		
r2	2			r6	r7	r8			2100	2308		
r3	2	r4	0	0	s14	s0			1001	1005		
r4	3	r5	r3	s15	s1	s12	s16		1054	1100	1105	
r5	2	r6	r4	s19	s11	s13			1114	2006		
r6	2	r7	r5	s2	s10	s3			2018	2019		
r7	4	r8	r6	s4	s5	s9	s19	s18	2200	2212	2220	2300
r8	2	0	r7	s6	s7	s8			2360	2401		

b)

Figure 2.12 Relation index INW en B-arbre d'ordre 5
a) schéma logique du B-arbre
b) stockage de INW.


r0
2007
r1
1010
1112
r2
2100
2308
r3
1001
1005
s14
s0
r5
1114
2006
s19
s11
s13
r7
2200
2212
2220
2300
s4
s5
s9
s17
s18
r4
1051
1100
1105
s15
s1
s12
s16
r6
2018
2019
s2
s10
s3
r8
2360
2401
s6
s7
s8
2
1005 plat 3,5 30 chargé marseille
1051 couvert 10 25 attaché lyon
2
2212 frigo 8 10 libre angers
2018 couvert 7 20 libre paris
2
2007 citerne 6,5 16 chargé grenoble
2019 plat 55 15 attaché tours
2
1112 frigo 10 15 chargé tours

2.3.2.3. Coût de stockage

Nous pouvons obtenir exactement et facilement le nombre de pages occupées à la fin de la création d'un B-arbre, puisqu'un noeud correspond à une page sous forme d'un n-uplet d'une relation comme les autres. Une fois qu'une relation index est créée, on peut avoir sa cardinalité, c'est-à-dire le nombre de n-uplets, correspondant au nombre de pages occupées. Dans le cas de la figure 2.11, la cardinalité est 9.

On peut également estimer le nombre de pages nécessaire pour une relation index en B-arbre en fonction de la longueur d'une clé Lc et le nombre de clés à inserer N. A partir de ces deux nombres, l'ordre d du B-arbre est d'abord obtenu, ce qui permet de calculer le nombre de noeuds NN selon les propriétés des B-arbres ii) et iii) dans 2.3.2.1:

$$(N \text{ div } (d-1)) + 1 <= NN <= (N \text{ div } ((d-1) \text{ div } 2)) + 1$$

De plus NN doit également satisfaire la contrainte ci-dessous:

$$NN = 1 \qquad \text{où } p = 1$$

$$>= 1 + \frac{2*(d \text{ div } 2 + 1)^{p-1} - 1}{(d \text{ div } 2)} \qquad \text{où } p = 2,3,\ldots,n$$

et

$$<= \frac{(d^{p} - 1)}{(d - 1)}$$

Nous arrivons alors à obtenir un ensemble d'entiers E dont chaque élément sera le nombre de noeuds possible constituant le B-arbre. Finalement le nombre de pages occupées sera MAX(E)

dans le pire des cas, et MIN(E) dans le meilleur des cas. Prenons un exemple de notre relation index INW à construire en supposant que d soit égal à 5. L'ensemble E est obtenu comme suit:

$5 <= NN <= 10$

et

$3 <= NN <= 6$ si $p = 2$, $9 <= NN <= 31$ si $p = 3$

alors $E = \{5,6,9,10\}$

Pour concrétiser cette estimation, regardons le tableau de la figure 2.13, qui indique un ensemble de nombres possibles de noeuds NN du B-arbre ayant une profondeur p en prenant un certain nombre de clés N et leur longueur Lc.

Nous pouvons ici constater que l'ensemble E est très variable en fonction de l'ordre d, autrement dit de la longueur d'une clé et du nombre de clés, et aussi en fonction d'un facteur important qui ne nous apparaît pas ici, qui est l'ordre d'insertion de clés.

N	Lc	d	p	NN
100	2	87	2	3
	6	51	2	3.. 4
	10	37	2	3.. 6
	25	17	2	7.. 13
	50	9	3	16.. 25
500	2	87	2	6.. 12
	6	51	2	10.. 20
	10	37	2	14.. 28
	25	17	3	32.. 63
	50	9	3	63.. 91
			4	81.. 125
1000	2	87	2	12.. 24
	6	51	2	20.. 40
	10	37	2	28.. 38
	25	17	3	63.. 125
	50	9	4	125.. 250
5000	2	87	2	59.. 88
	6	51	3	100.. 200
	10	37	3	139.. 278
	25	17	4	313.. 625
	50	9	4	625.. 820
			5	625..1250
10000	2	87	3	133.. 233
	6	51	3	200.. 400
	10	37	3	278.. 556
	25	17	4	625..1250
	50	9	5	1250..2500
			6	2031..2500

Figure 2.13 Tableau des nombres possibles de noeud

2.3.2.4 Opération sur un B-arbre

Trois opérations fondamentales ont été définies sur un B-arbre; insertion, suppression et recherche. La modification peut être considérée comme une suppression suivie d'une insertion. Ces opérations sont transparentes à l'utilisateur de MIMER car elles sont automatiquement générées au moment de l'insertion, de la suppression et de la recherche de n-uplets.

INSERTION

Cette opération permet d'insérer une nouvelle clé dans un B-arbre en préservant les propriétés du B-arbre. Cela se passe pendant

1) l'insertion d'un n-uplet dans la relation de base sur laquelle une relation index a été préalablement définie
2) la définition d'une relation index sur une relation de base ayant déjà des n-uplets.

Deux cas peuvent se présenter: insertion simple et insertion avec éclatement. L'insertion simple, a lieu quand le noeud, dans lequel la clé sera insérée, n'est pas complet c'est-à-dire qu'il comporte moins de (d - 1) clés. On peut alors insérer une clé et son pointeur dans la case appropriée du noeud (figure 2.14). Lorsque le noeud dans lequel l'insertion aura lieu est complet, l'insertion avec éclatement a lieu. Le noeud est éclaté en deux et l'une de ses clés doit monter au noeud père (figure 2.15). La clé qui monte crée une nouvelle insertion dans

clé à insérer:1072

Figure 2.14 Insertion simple

le noeud père. Cela se poursuit jusqu'à l'insertion simple dans l'un des noeuds ancêtres ou à la création d'une nouvelle racine en augmentant la profondeur du B-arbre (figure 2.15). Donc un B-arbre grossit de façon ascendante.

clé à insérer: 2211

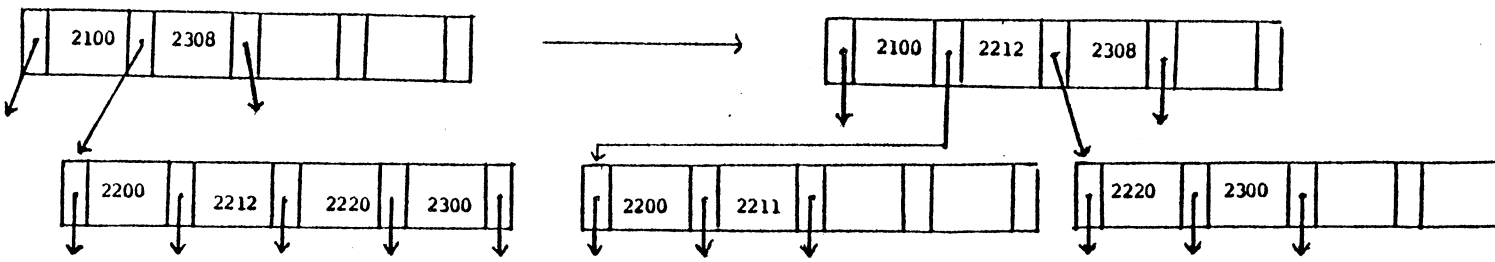


Figure 2.15 Insertion avec éclatement

L'insertion d'une clé dans un B-arbre de profondeur p, demande d'abord p lectures de pages de la relation index pour arriver au noeud feuille. Si l'insertion simple a lieu, il nous faut en plus une écriture de page. Par contre, il est nécessaire de modifier une page pour le noeud père et d'allouer une page pour le noeud frère en supplément à chaque éclatement. Supposons que K (0<=k<=p) éclatements se soient produits lors de

l'insertion d'une clé. Alors (p+1+4k) pages sont accédées en lecture et écriture.

SUPPRESSION

Cette opération est l'opération inverse de l'insertion; elle permet de supprimer une clé existante dans une relation inverse en préservant les propriétés du B-arbre. Cela peut se passer pendant la suppression de n-uplets seulement, car la modification des attributs participant à une clé n'est pas permise dans MIMER.

Trois cas de suppression peuvent se présenter; suppression simple, suppression avec débordement et suppression avec fusion.

La suppression simple, se produit quand le noeud contenant la clé à supprimer conserve le taux de remplissage supérieur ou égal à 50% lors de sa suppression. Dans le B-arbre que nous avons implanté, une clé peut se trouver dans un noeud-feuille et dans un noeud non-feuille. La figure 2.16 présente les deux cas de suppression simple.

La suppression avec débordement, se presente lors de la suppression d'une clé d'un noeud qui laisse celui-ci avec un taux de remplissage inférieur à 50% et la présence d'un des ses noeuds frères (à gauche ou à droite) dont les clés sont distribuées à condition que les deux noeuds puissent entretenir leur taux de

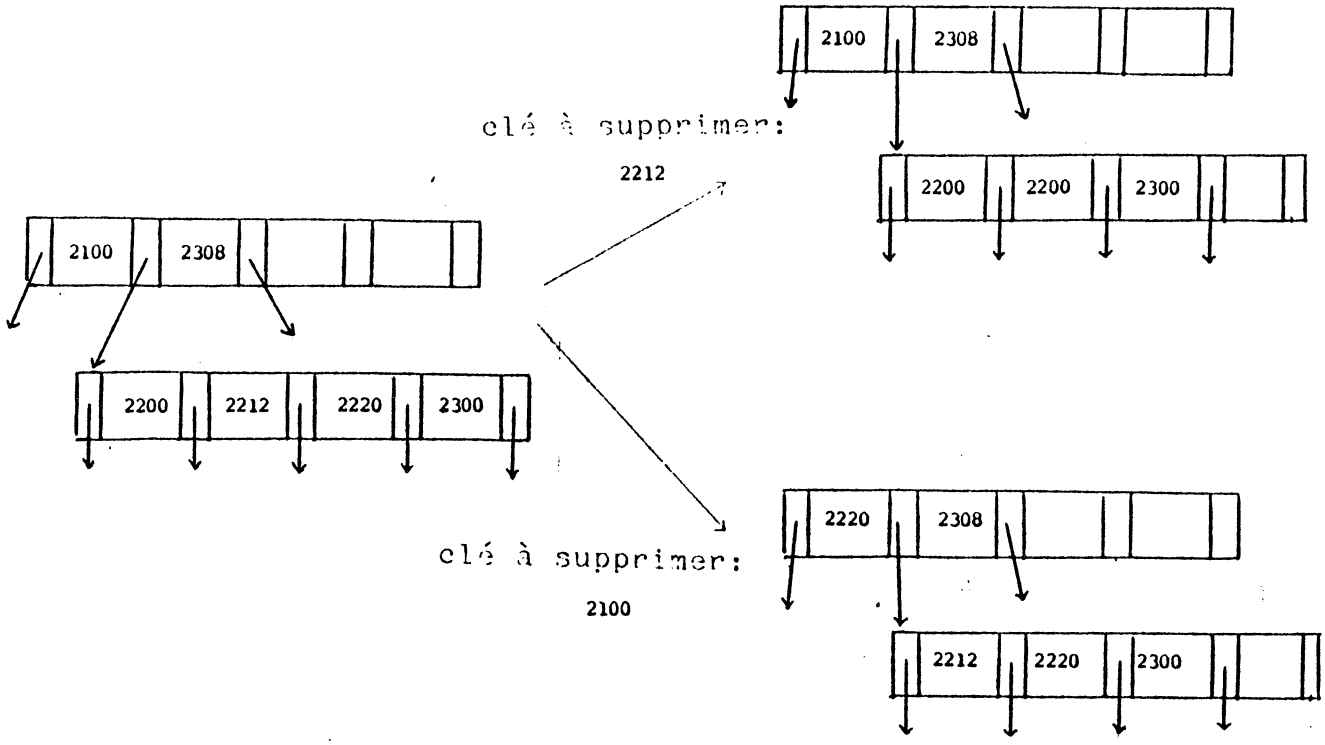


Figure 9.16 Suppression simple

remplissage supérieur ou égal à 50% (figure 2.17).

Bien que la suppression d'une clé dans un noeud se produise (comme le cas de suppression avec débordement), les deux noeuds frères peuvent ne pas répartir leur clé, car leur taux de remplissage peut ne plus se maintenir après la distribution. Dans ce cas là, la suppression avec fusion a lieu (figure 2.18). Cela continue jusqu'à ce qu'un des noeuds ancêtres maintienne son taux de remplissage. Si cela arrive à la racine du B-arbre, la profondeur p diminue.

Pour la suppression d'une clé dans un B-arbre de profondeur p, on commence dans tous les cas par la suppression simple. Dans ce cas, il nous faut p lectures de pages pour arriver au noeud feuille, même si la clé se trouve dans un noeud

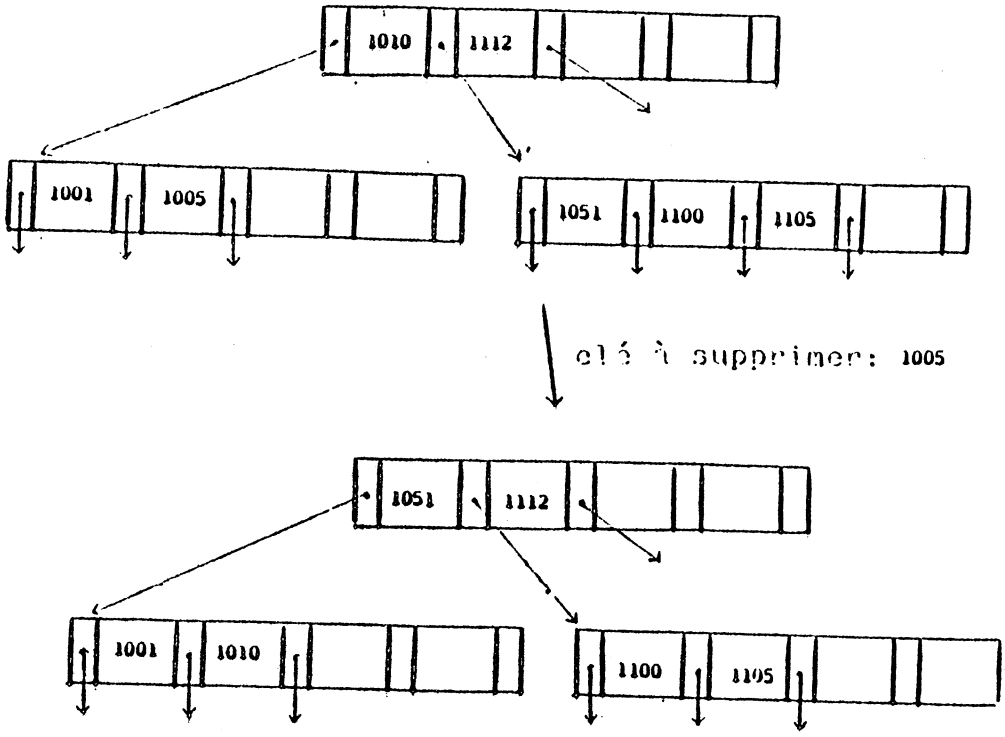


Figure 2.17 Suppression avec débordement

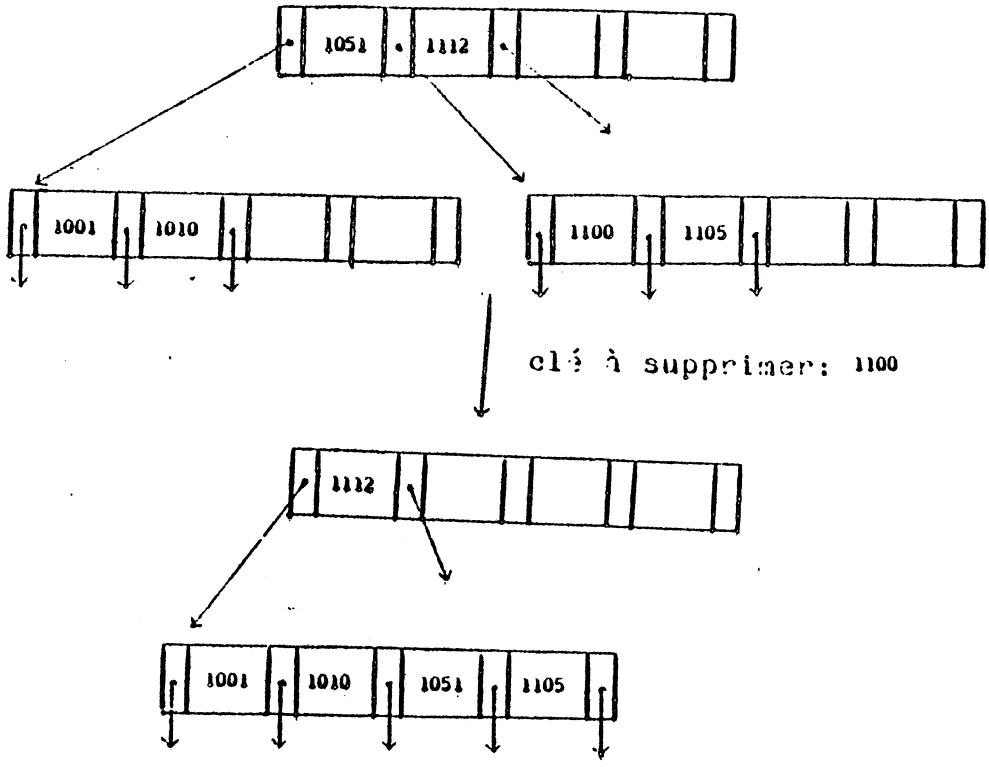


Figure 2.18 Suppression avec fusion

non-feuille, puisque le pointeur vers un n-uplet d'une relation de base est toujours dans un noeud feuille (voir figure 2.12). Dans le cas où la clé se trouve dans un noeud feuille, p lectures de pages et une écriture pour modifier le noeud sont nécessaires. Il faut compter une écriture de plus dans le cas contraire.

Si le taux de remplissage du noeud est inférieur à 50%, la suppression avec débordement ou la suppression avec fusion doivent se produire. Les deux opérations demandent deux lectures et deux écritures pour les noeuds père et frère (la libération d'une page a aussi besoin d'une écriture pour le chaînage des pages libres).

Comme la suppression d'une clé s'est effectuée soit avec suppression simple, soit avec suppression avec débordement, elle requiert un nombre d'accès aux pages formulé comme suit (en supposant que k suppressions avec fusion se soient produites lors de l'opération):

p	lecture de page
+	
(+ 1	si la clé se trouve dans un noeud non-feuille)
+	
4k	lectures et écritures de pages pour k suppressions avec fusion où 0 <= k <= p
+	
1	si elle se termine avec suppression simple
ou	
5	si elle se termine par suppression avec débordement

RECHERCHE

Cette opération est une des plus importantes et est aussi impliquée dans l'interface de MIMER. Elle est

automatiquement requise lors de l'accès aux n-uplets contenant une clé donnée. Nous allons l'étudier surtout du point de vue des performances d'accès à une clé donnée, dans les paragraphes suivants.

lorsqu'une relation index a été déjà construite, les catalogues possèdent assez d'informations pour permettre d'estimer facilement le coût; par exemple la profondeur du B-arbre, son ordre, le nombre de noeuds, ... etc. Considérons une recherche d'un n-uplet ayant une clé donnée, p accès aux pages sont nécessaires dans tous les cas, puisque le pointeur vers la relation de base se trouve dans un noeud feuille. En plus le coût de recherche dichotomique dans un noeud doit être compté. La comparaison des clés pour cela se fait au maximum Log Nc + 1 fois dans un noeud, où Nc représente le nombre de clés stockées dans un noeud, si la clé ne s'y trouve pas. Supposons que les clés du B-arbre soient distribuées régulièrement dans les noeuds, le nombre moyen de clés dans un noeud peut être calculé aussi:

nombre de clé (N) / nombre de noeuds (NN)

le coût maximum est p accès aux pages plus P*($Log_2(N/NN) + 1$) comparaisons de clés.

D'autre part, considérons une relation ayant une cardinalité N sur laquelle une relation index en B-arbre va être définie; le coût d'accès est très lié à la profondeur et au nombre de clés dans un noeud comme nous l'avons vu dans le paragraphe précédant (figure 2.13). Dans les paragraphes de 2.3.2.3, nous avons vu que la profondeur et le nombre de noeuds

du B-arbre peuvent être modifiés selon l'ordre d'insertion des clés et aussi en fonction de la cardinalité N et de la longueur d'une clé Lc.

le tableau de la figure 2.17 indique les accès nécessaires aux pages et le nombre moyen de comparaisons de clé à partir du tableau de la figure 2.13. Nous voyons dans ce tableau une belle performance de recherche d'une clé donnée; par exemple au maximum 6 accès pages et 18 comparaisons de clés sont nécessaires sur 10000 clés ayant 50 octets de longueur.

N	Lc	d	F	NN	NAF	NMC
100	2	87	2	3	2	12
	6	51	2	3	2	12
				4	2	10
	10	37	2	3	2	12
				4.. 6	2	10
	25	17	2	7.. 12	2	8
				13	2	6
	50	9	3	16.. 24	3	9
				25	3	6
500	2	87	2	6.. 7	2	14
				8.. 12	2	12
	6	51	2	10.. 15	2	12
				16.. 20	2	10
	10	37	2	14.. 15	2	12
				16.. 28	2	10
	25	17	3	32.. 62	3	12
				63	3	9
	50	9	3	63.. 91	3	9
			4	81.. 124	4	12
				125	4	8
1000	2	87	2	12.. 15	2	14
				16.. 24	2	12
	6	51	2	20.. 31	2	12
				32.. 40	2	10
	10	37	2	28.. 31	2	12
				32.. 38	2	10
	25	17	3	63.. 124	3	12
				125	3	9
	50	9	4	125.. 249	4	12
				250	4	8
5000	2	87	2	59.. 78	2	14
				79.. 88	2	12
	6	51	3	100.. 156	3	18
				157.. 200	3	15
	10	37	3	139.. 156	3	18
				157.. 278	3	15
	25	17	4	313.. 624	4	16
				625	4	12
	50	9	4	625.. 820	4	12
			5	625..1249	5	15
				1250	5	10
10000	2	87	3	133.. 156	3	21
				157.. 233	3	18
	4	51	3	200.. 312	3	18
				313.. 400	3	15
	10	37	3	278.. 312	3	18
				313.. 556	3	15
	25	17	4	625..1249	4	16
				1250	4	12
	50	9	5	1250..2499	5	15
				2500	5	10
			6	2031..2499	6	18
				2500	6	12

Figure 2.19 Tableau de perfomance de recherche d'une clé dans un B-arbre

2.3.3 Interfaces d'utilisation

Nous nous proposons de voir ici le langage de primitives appartenant à l'utilisation des B-arbres en suivant les mêmes définitions et conventions que celles données dans <FER 81>.

CREER_RELATION_INDEX (IR1, RX, IR2, <type-d'index>, <liste-d'attributs>, CE) ;

- paramètres en entrée : IR1, RX, <type-d'index>, <liste-d'attributs>
  paramètres en sortie : IR2, CE
- RX est le nom externe de la relation index que l'on veut créer.
- IR1 est le nom interne de la relation sur laquelle on veut bâtir une relation index (IR2). La clé de IR2 est donnée par la concaténation des noms internes 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 n-uplets 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 repétées seront admises.

OBTENIR_CARACTERISTIQUES_INDEX (IR1, <nombre-d'index>, <tableau>, CE) ;

- paramétres 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> résultat est structuré ainsi ;

	nom externe d'index	nom interne d'index	type d'index	ordre	longueur d'une clé	profondeur	nombre d'attributs de clé	nombre de clés différentes	attributs de clé (nom externe)
1									
2									
3									
...									
<nombre-d'index>									

Ces deux nouvelles primitives sont définies. De plus les autres primitives sont modifiées (sans changement de leurs paramètres) pour la manipulation des relations index: notament INSERER_TUPLE, DETRUIRE_TUPLE, TUPLE_SUIVANT et RECOMMENCER_BALAYAGE pour la manipulation des n-uplets d'une relation, et CREER_ATTRIBUT et DETRUIRE_ATTRIBUT pour maintenir l'aspect dynamique du schéma d'une relation.

Une relation index existant sur une relation peut être détruite par la primitive DETRUIRE_RELATION comme une relation de base. Grâce à l'accès séquentiel ordonné, une autre primitive d'application est introduite:

GROUPER_RELATION (IR1, <ordonnancement>, <liste-d'attributs>, CE)

- paramètres en entrée : IR1, <ordonnancement>, <liste-d'attributs>
  paramètres en sortie : CE
- A cause du mécanisme d'insertion et de suppression des n-uplets d'une relation stockée séquentiellement, des "trous" sont laissés dans les pages; c'est-à-dire que l'espace de la relation n'est pas réorganisé. Ces trous peuvent être supprimés par l'appel à la primitive GROUPER_RELATION.
- Cela permet donc, de grouper une relation en la triant simultanément selon les attributs IA1, IA2, ... IAn de la <liste-d'attribts>.
- Le groupement n'est pas maintenu; c'est ainsi qu'une relation restera complètement groupée jusqu'à ce que de nouvelles insertions de n-uplets ou groupement (sur d'autre attributs) se produisent.
- Puisqu'ici grouper est aussi synonyme de trier, <ordonnancement> peut être "A" (tri ascendant) ou "D" (tri descendant).
- Elle ne peut être appliquée sur des relation stockées séquentiellement.

Les primitives de MIMER sont classées en trois niveaux <FER 81>. Dans le niveau B, interface intermédiaire, nous ajoutons les primitives de gestion des B-arbres:

EXTRAIRE_CLE (<descripteur-d'index>, <n-uplet_1>, <n-uplet_2>, CE) ;

- paramètres en entrée : <descripteur-d'index>, <n-uplet_1>
  paramètres en sortie : <n-uplet_2>, CE
- Celle-ci sert à extraire la valeur d'une clé <n-uplet_2> à partir d'un n-uplet <n-uplet_2> selon la composition de la clé indiquée dans <descripteur-d'index>.
- Il est évident que cette primitive est utilisée avant toutes les opérations qui concernent la clé.

INSERER_CLE (IR1, IDT, <n-uplet>, CE)
- paramètres en entrée : IR1, IDT, <n-uplet>
  paramètres en sortie : CE
- Cette primitive permet d'insérer une clé dans une relation index <IR1>, celle-ci est extraite du n-uplet <n-uplet>.
- l'endroit où ce n-uplet est stocké, est indiquée par IDT.

SUPPRIMER_CLE (IR1, IDT, <n-uplet>, CE)
- paramètres en entrée : IR1, IDT, <n-uplet>
  paramètres en sortie : CE
- Cette primitive permet de supprimer une clé à extraire d'un n-uplet dans une relation index IR1.
- IDT est nécessaire dans le cas d'un index secondaire pour identifier la clé exacte à supprimer.

CHERCHER_CLE_B-ARBRE (<descriteur-d'index, <clé>, IDT, <niveau>, <adresse-de-noeud>, <commande>, <existence-de-clé>, <indicateur>, CE)
- paramètres en entrée : < descripteur-d'index>, <clé>, IDT, <niveau>, <adresse-de-noeud>, <commande>

paramètres en sortie : <existence-de-clé>, <indicateur>, IDT, CE

- Cette primitive permet de trouver une place pour la clé dans le cas de l'insertion et de chercher une clé donnée dans le cas de la suppression et de la recherche, grâce au champ <commande>.
- Cette procedure s'exécute d'une façon recursive le long, d'une branche de B-arbre en ayant pour niveau et noeud courant <niveau> et <adresse-de-noeud>.
- L'IDT est utilisé en cas d'insertion et de suppression pour donner l'adresse du n-uplet contenant la clé et pour fournir l'adresse du noeud contenant la clé dans le cas d'une recherche.
- Les opérations sur un B-arbre sont indiquées par <commande> soit "I" pour insertion, soit "S" pour suppression, soit "C" pour recherche.
- Deux paramètres booléens <existence-de-clé> et <indicateur> sont définis pour indiquer l'existence de la clé et la mise à jour du descripteur d'index.

CHAPITRE III

OPTIMISATION DU CHOIX D'UN CHEMIN D'ACCES

3.1 Introduction

Dans les SGBD relationnels, les langages sont de haut niveau et permettent d'exprimer l'ensemble des données que l'on souhaite obtenir, sans indiquer la façon de le faire. D'un autre côté, l'accès aux données se fait par des méthodes d'accès qui dépendent à la fois de l'organisation physique et du chemin d'accès que l'administrateur de la base de données a définis, ou que le système lui-même crée dynamiquement pendant l'exécution d'une requête.

Lors de l'accès aux données, il n'y a pas de solution unique mais plusieurs. Il appartient alors au système de gestion de base de données de rechercher celle qui conduit aux performances optimales. Cette phase d'optimisation dans les SGBD relationnels est une de leurs caractéristiques essentielles.

Nous abordons ici seulement les techniques qui tiennent compte de l'existence et de la création dynamique de tel et tel chemin d'accès pour élaborer une solution optimale. Nous n'abordons pas les autres points, par exemple l'optimisation des requêtes exprimées dans un langage algébrique, à l'aide de techniques basées sur les propriétés des opérateurs de l'algèbre relationnelle.

Nous commençons d'abord par examiner les techniques proposées dans les autres systèmes relationnels, puis nous présenterons les techniques que nous avons adoptées dans le système MICROBE.

3.2 Exemples de choix d'un chemin d'accès

les systèmes relationnels possèdent des stratégies pour effectuer le choix des chemins d'accès, qui leur sont propres, qui prennent en consideration notamment leurs organisations de données. Certaines de ces stratégies sont baséees sur l'estimation du coût de chaque chemin possible avec les stratégies obtenues sur la base de données. Nous appelons une telle stratigie "stratégie dynamique". Par contre, certaines adoptent l'ordre d'utilisation fixé de tel ou tel chemin d'accès déjà analysé, une telle stratégie sera appelée "stratégie fixe".

Ces stratégies présentent en général deux cas selon le nombre de relations à traiter: optimisation sur une seule relation et optimisation sur plusieurs relations. Nous allons les étudier dans deux systèmes utilisant des langages de manipulation de types différents; INGRES et SYSTEM-R.

3.2.1 INGRES

Ce système a été développé à l'université de Californie à Berkeley sur les ordinateurs de série PDP-11 sous le système d'exploitation UNIX <STO-76>. Il est composé de quatre modules dont l'un est consacré à la commande de recherche d'informations. On y trouve l'optimisation adoptée par le système.

Commençons par l'optimisation sur une seule relation désignée par une "variable" dans une requête. Cela est réalisé

dans un processeur appelé OVQP (One-Variable Query Processor) qui permet d'accèder à des n-uplets appartenant à une seule relation. La partie initiale de ce logiciel appelée "STRATEGY" est chargée de déterminer le chemin optimal pour accèder à la relation. Etudions la politique de choix dans STRATEGY.

D'abord, il examine une formule atomique, spécifiant la valeur d'un attribut sous forme: XθA où X est un attribut de R, A une constante et θ est un opérateur de comparaison =, >, <, <=, >=. La constante dans la formule permettra de limiter le balayage de la relation.

Un balayage de la relation permettant d'obtenir des n-uplets peut se faire de deux façons; 1) par l'accès à la relation primaire qui stocke réellement les n-uplets entiers, suivi d'un balayage séquentiel de la relation entre les limites, 2) par l'accès intermédiaire d'une relation index suivi d'un balayage séquentiel sur cet index, entre les limites pour obtenir un ensemble d'indentificateurs de n-uplet.

Pour effectuer un choix, le chemin d'accès doit être déterminé en obtenant les informations concernant la relation primaire et la relation index. De plus, la compatibilité entre un chemin disponible et la formule atomique donnée doit être établie. Par exemple, une relation ayant une organisation en hachage requiert qu'une formule atomique spécifie l'égalité comme opérateur de comparaison pour pouvoir être utilisée.

Le logiciel STRATEGY vérifie cette compatibilité des chemins disponibles dans l'ordre suivant;

1) relation primaire hachée
2) relation index hachée
3) relation primaire ISAM
4) relation index ISAM

La raison de cet ordre est liée au nombre d'accès aux pages requises pour chercher les n-uplets satisfaisant la formule.

Prenons un exemple en supposant que la relation WAGON dans la figure 2.4 est une relation en ISAM avec une clé NO-WAGON, et qu'en plus une relation index hachée sur CAPACITE existe. Une requête pour chercher les wagons dont la capacité est supérieur à 25 tonnes est exprimée en QUEL:

```
RANGE OF W IS WAGON
RETRIEVE (W.NO-WAGON)
WHERE    W.CAPACITE > 25
```

Comme la relation index hachée n'est pas compatibile avec le prédicat (W.CAPACITE > 25), un balayage complet sur WAGON est requis. S'il y a un index sur CAPACITE en ISAM, STRATEGY peut prendre cet index comme un chemin d'accès.

Comme INGRES utilise le langage prédicatif à variable n-uplet, le principe fondamental d'optimisation sur plusieurs relations consiste à décomposer une requête portant sur plusieurs variables en une série de requêtes portant sur une seule variable <WON 76>. Cela est accompli par un logiciel appelé "DECOMP" avec comme objectifs d'éviter le produit cartésien et de réduire le nombre de n-uplets à balayer.

DECOMP comporte deux opérations principales:

substitution de n-uplet et détachement d'une variable. La plus grande part de l'optimisation a lieu dans la partie de sélection d'une variable, pour substituer une valeur à une variable. Enfin cela transforme une requête ayant plusieurs variables en une requête ayant une seule variable. Un balayage de relation n'est associé qu'à une seule variable. Pour cela, on estime le coût relatif de la substitution pour chaque variable. Une variable ayant le coût minimum doit être choisie pour la substitution. Ceci détermine non seulement la décomposition d'une requête, mais aussi son interprétation en la modifiant par le détachement d'une variable.

Pour la substitution, on a souvent besoin de garder une ou plusieurs relations temporaires qui sont les résultats de l'exécution de OVQP. Il est possible de choisir leur organisation et leur clé en prévoyant les accès nécessaires pour en assurer l'accès rapide. Ceci revient en fait à la création dynamique de chemins d'accès. Au moment de l'accès de ces relations temporaires, si ces chemins ne sont plus bons, on peut aussi les réorganiser.

Considérons une requête pour trouver les numéros de wagons de type frigo passeront le 830831 sur la ligne 110:

```
RANGE OF W is WAGON
RANGE OF T is TRAIN
RANGE OF R is TRAFIC
RETRIEVE (W.NO-WAGON)
WHERE     W.TYPE = "frigo"
  AND     R.NO-LIGNE = "110"
  AND     R.JOUR = "830831"
  AND     W.NO-WAGON = T.NO-WAGON
  AND     T.NO-TRAIN = R.NO-TRAIN
```

Ce qui nous intéresse, est la sélection d'une variable pendant la décomposition, après l'exécution de OVQP avec trois formules atomiques <W.TYPE = "frigo">, <R.NO-LIGNE = "110"> et <R.JOUR = "830831"> sur deux variables W et R. Nous avons donc la requête modifiée comme suit:

```
RANGE OF W is WAGON'
RANGE OF T is TRAIN
RANGE OF R is TRAFIC'
RETRIEVE (W.NO-WAGON)
WHERE    W.NO-WAGON = T.NO-WAGON
  AND    T.NO-TRAIN = R.NO-TRAIN
```

Ceci vient à engendrer les deux requêtes:

```
RANGE OF W is WAGON
RETRIEVE INTO WAGON'(W.NO-WAGON)
WHERE    W.TYPE = "frigo"

RANGE OF R is TRAFIC
RETRIEVE INTO TRAFIC'(R.NO-TRAIN)
WHERE    R.NO-LIGNE = "110"
  AND    R.JOUR = "830831"
```

Les relations temporaires WAGON' et TRAFIC', dans un premier temps, sont stockées en hachage sur NO-WAGON et NO-TRAIN. Maintenant il faut choisir une variable pour la substitution de n-uplet. En première versibn, INGRES choisit une variable ayant le moins de n-uplets. Mais ce n'est pas toujours une bonne solution. Une technique est adpotée en estimant le coût de chaque substitution possible <WON 76>. Supposons que ce soit TRAFIC'. La requête est transformée par une requête:

```
RANGE OF T is TRAIN
RETRIEVE INTO TRAIN'(T.NO-TRAIN)
WHERE    T.NO-TRAIN = "NO-TRAIN de TRAFIC"
```

La variable R peut être détachée en substituant tous les n-uplets de la relation TRAFIC', puis la requête est modifiée:

```
RANGE OF W is WAGON'
RANGE OF T is TRAIN'
RETRIEVE (W.NO-WAGON)
WHERE    W.NO-WAGON = T.NO-WAGON
```

Finalement, nous pouvons avoir le résultat en substituant un n-uplet de TRAIN. Comme nous l'avons vu, chaque substitution de valeur fait activer OVQP pour chercher des n-uplets correspondants. Dans le cas de la substitution de TRAIN', nous avons une chance d'accès à la relation WAGON', car elle est hachée sur NO-WAGON et l'oprateur de comparaison est "=". Par contre, dans le cas de TRAFIC', on ne profite pas de son organisation. En plus, si on n'a pas de compatibilité entre un opérateur de comparaison et l'organisation de relation, la relation est réorganisée pour avoir une compatibilité.

En conclusion, l'optimisation dans INGRES commence par le choix d'un chemin en vérifiant la compatibilité entre les chemins disponibles et la formule (prédicat) donnée. Pour les requêtes portant sur plusieurs variables, la technique de décomposition et la création dynamique d'un chemin dans la relation temporaire sont successivement effectuées. Le résultat est enfin calculé par le module STRATEGY.

3.2.2 SYSTEM-R

Ce SGBD relationnel a été developpé au laboratoire IBM comme prototype <AST 76>. Les systèmes commercialisés SQL/DS sont basés sur ce projet. SEQUEL est un langage de description et de manipulation de données supporté par le système. Pour exécuter d'une façon moins coûteuse une requête SEQUEL dans un environnement donné, il dispose d'un optimiseur qui essaie de minimiser l'accès aux pages de la mémoire secondaire <SEL 70>

<DEL 82>. Le regroupement physique des n-uplets dans la base de données est d'une grande importance, car chaque relation peut avoir au maximum un index groupant appelé "clustering image".

L'optimisation commence par la classification d'une requête selon plusieurs types en fonction de la présence des diverses fonctions du langage, comme la jointure et le "group by". Ensuite, les catalogues du système sont examinés pour trouver un ensemble d'images et de liens qui sont caractéristiques d'une requête. Une procédure de décision s'exécute alors pour fixer l'ensemble des méthodes raisonables d'exécution de la requête. S'il y en a plusieurs, pour chaque méthode, son coût est estimé et la méthode ayant le coût minimum est chosie.

Commençons par les requêtes portant sur une seule relation présenté en SEQUEL;

```
SELECT X1,X2,...,Xn
FROM   R
WHERE  F
```

où X1, X2, ..., Xn sont les attributs de la relation R et F est un ensemble de formules atomiques définies sur les attributs de R.

L'optimiseur choisit un chemin en fonction des paramètres disponibles dans le système, soient la cardinalité de R, le nombre de pages nécessaires au stockage de R, pour chaque index I, le nombre des différentes clés et le nombre de pages nécessaires au stockage de I de la façon suivante:

1) S'il y a un index groupant qui correspond à la formule dont l'opérateur de comparaison est "=". Il est choisi comme chemin d'accès.

2) Si une des méthodes suivantes est disponibles;

2-a) utilisation d'un index groupant qui correspond à la formule dont l'opérateur de comparaison n'est pas "="

2-b) utilisation d'un index non-groupant qui correspond à la formule dont l'opérateur de comparaison est "="

2-c) utilisation d'un index groupant qui ne correspond à aucune formule atomique

2-d) accés à toutes les pages du segment où se trouve la seule relation

l'une d'entre elles sera choisie, mais s'il y en a plus d'une , il faut calculer le coût des chemins disponibles à l'aide des formules existant dans le système. Le chemin ayant le coût minimum sera choisi.

3) Si aucun chemin décrit dans 1) et 2) n'est disponible, alors l'une des méthodes sera choisie selon sa disponibilité dans l'ordre suivant;

3-a) utilisation d'un index non-groupant qui correspond à la formule dont l'opérateur de comparaison n'est pas "="

3-b) utilisation d'un index non-groupant qui ne correspond à aucune formule atomique

3-c) accès à toutes les pages du segment partagé pour stocker plusieurs relations.

Prenons une requête en SEQUEL pour chercher les wagons dont la capacité est égal à 25 tonnes:

```
SELECT NO-WAGON
FROM   WAGON
WHERE  CAPACITE = "25"
```

Supposons que la relation WAGON a un index groupant sur NO-WAGON et un index non-groupant sur CAPACITE et qu'elle se trouve seule

dans les pages de son segment. Dans ce cas, nous avons trois chemins disponibles: 2-a), 2-b) et 2-d). Alors il faut estimer le coût de chaque chemin en utilisant les formules disposées dans le système avec les paramètres décrit dans le paragraphe précédent. Les fromules sont donc:

R/(2*T) pour 2-a)

R/I pour 2-b)

(R/I) +(H*R*N pour 2-d)

où R est la cardinalité de WAGON, D le nombre de pages occupées par WAGON, T le nombre moyen de n-uplets dans une page de R (R/D), I le nombre de différentes clés, et H le coefficient de UC coût (1/H est le nombre de comparaisons de n-uplets pendant l'accès à une page de la mémoire secondaire).

Le choix est un chemin le coût minimum estimé en substituant les valeurs réelles aux paramètres.

Pour l'optimisation des requêtes portant sur plusieurs relations, il faut d'abord choisir l'ordre dans lequel i faut évaluer une séquence de produits. Une fois que c'est fait, la recherche de la stratégie optimale se fait par étapes successives en construisant un arbre des solutions possibles.

- A la première étape, on considère individuellement chaque relation impliquée dans le produit. Le principe décrit dans les paragraphes précédents, pour l'optimisation est appliqué. On construit les feuilles de l'arbre en ne gardant un chemin d'accès que si

  1) c'est le moins coûteux de ceux fournissant un résultat non ordonné,

2) c'est le moins coûteux de ceux qui donnent les résultats selon un ordre pouvant avoir un intérêt par la suite.

Pour avoir un exemple au cours de cette optimisation, nous reprenons la deuxième question de 3.2.1 représenté en SEQUEL comme suit:

```
SELECT NO-WAGON
FROM   WAGON,TRAIN,TRAFIC
WHERE  WAGON.TYPE = "frigo"
  AND  TRAFIC.NO-LIGNE = "110"
  AND  TRAFIC.JOUR = "830831"
  AND  WAGON.NO-WAGON = TRAIN.NO-WAGON
  AND  TRAIN.NO-TRAIN = TRAIN.NO-TRAIN
```

Il y trois index principaux: NO-WAGON de WAGON, NO-TRAIN de TRAIN et NO-TRAIN de TRAFIC, et des index secondaires sur chaque attribut TYPE, ETAT et CAPACITE de WAGON et JOUR de TRAFIC. Pour accèder à chaque relation, les formules applicables sont <TRAFIC.NO-LIGNE = "110"> et <TRAFIC.JOUR = "830831"> sur TRAFIC et <WAGON.TYPE = "frigo"> sur WAGON et les ordres intéressants sont NO-WAGON et NO-TRAIN. Quant aux principes 1) et 2), on garde les chemins pour chaque relation comme suit: index sur NO-WAGON et celui sur TYPE pour WAGON, index sur NO-TRAIN pour TRAIN et index sur NO-TRAIN et celui sur JOUR pour TRAFIC.

- A la seconde étape, on considère tous les couples de relation qui existent dans l'ordre d'une séquence de produits. Pour chacun de ces couples, les cas suivants sont plus considérés:

  1) une relation externe et l'autre interne ou inversement,

  2) le produit pouvant être effectué par boucles imbriquées ou par tri-composition,

3) tous les chemins d'accès aux deux relations qui ont été gardées à la première étape.

L'ordre résultant de cette seconde étape va refléter toutes les combinaisons de ces différents cas. Le coût d'accès minimum de chaque possibilité est alors évalué. A la fin de cette étape, l'ordre des possibilités est élagué pour ne garder que les chemins de coûts minimaux et ceux donnant les n-uplets dans un ordre intéressant.

Continuons avec l'exemple précédent, on considère tous les couples possibles pour une jointure: (WAGON, TRAIN), (TRAIN, WAGON), (TRAIN, TRAFIC), et (TRAFIC, TRAIN). En conservant l'accès à chaque relation et en appliquant une méthode disponible de jointure, le coût de chaque couple est calculé selon les formules du système.

- La dernière étape consiste à itérer le processus de la seconde étape d'abord pour trois relations jusqu'au nombre de relation de la requête.

Pour conclure notre exemple, considérons les produits de trois relations en supposant que les deux couples (TRAIN, WAGON) en utilisant l'index sur NO-WAGON pour TRAIN et celui sur NO-WAGON pour WAGON et (TRAFIC, TRAIN) en utilisant l'index sur NO-TRAIN pour TRAFIC et celui sur NO-TRAIN pour TRAIN sont obtenus commes des solutions optimales pour deux relations. On évalue le coût de (TRAIN*WAGON, TRAFIC) et (TRAFIC*TRAIN, WAGON). Le produit ayant le coût minimum parmi deux est retenu en gardant

les index indiqués comme son chemin.

Le résultat des techniques que nous venons de décrire est exprimé en un langage pour spécifier un chemin d'accès, dit "Access Specification Language (ASL)" <LOR 78>. Enfin il est traduit en langage machine pour exécuter la stratégie choisie par l'optimiseur.

3.3 Choix d'un chemin d'accès dans MIMER

Ici, nous présentons les techniques de séléction optimale des chemins d'accès employées dans MICROBE en distinguant deux cas comme nous l'avons décrit précédemment. Le traducteur MIQUEL produit une arborescence algébrique relationnelle en recevant une requête émise par l'utilisateur. L'optimiseur algébrique MIRES peut la modifier. Comme cette arborescence est constituée d'un ensemble d'opérateurs de l'algèbre relationnelle, nous ne traitons ici que l'interprétation des opérateurs (sans modification de leur séquence) et plus précisément, la meilleure façon d'exécuter les opérateurs par l'intermédiaire des primitives de MIMER.

L'optimisation sur une seule relation est donc impliquée dans le mécanisme d'accès aux relations de MIMER. Par contre, la sélection d'un chemin optimal sur plusieurs relations a lieu pour les opérateurs lors de leur interprétation.

L'accès aux n-uplets d'une relation donnée est assuré par les primitives de MIMER avec la définition d'une notion de "balayage". Ce balayage agit toujours sur une et une seule relation et on peut y définir une condition constituée d'un ensemble de formules atomiques pour qualifier les n-uplets de la relation. Cela peut être considéré comme un opérateur de l'algébre relationnelle SELECT. Les balayages peuvent être classés selon l'existence d'une condition comme précédemment; balayage inconditionnel et balayage conditionnel, et aussi

classés en fonction du chemin d'accès employé (balayage simple et balayage indirect).

Un balayage simple se fait en accédant à toutes les pages comportant des n-uplets de la relation à balayer. Un n-uplet peut aussi être obtenu par l'intermédiaire d'une relation index, dans ce cas, on a un balayage indirect. Lorsqu'un balayage est lancé sur une relation donnée, il doit être fait avec le coût minimum. Il faut faire un choix parmi un balayage simple et les balayages indirects disponibles. Cela est présenté tout au long des paragraphes suivants.

3.3.1 Définition du problème

Dans MIMER, une relation index comporte logiquement deux attributs;

- l'attribut CLE qui permet de distinguer un n-uplet des autres n-uplets de la même relation. Si cette clé est unique, nous l'appelerons "clé primaire". De même la clé peut être non-unique (possibilité de clés répétées). Nous l'appelerons alors "clé secondaire".
- l'attribut POINTEUR qui stocke l'identificateur du n-uplet (IDT) correspondant à la valeur d'une clé.

Une relation index est éventuellement créée sur une relation de base par l'administrateur et ses clés sont stockées dans l'ordre ascendant. Cependant les pointeurs dans une relation index se dispersent toujours d'une façon arbitraire vers les

pages où les n-uplets d'une relation de base se trouvent (figure 2.11-a)). Cela est équivalent au "non-clustering image" de SYSTEM-R <AST 76> et à la relation index de INGRES <STO 76>.

Un problème se pose lorqu'un balayage portant sur un ensemble de formules atomiques, dit balayage conditionnel, est lancé et qu'en plus une (ou plusieurs) relation(s) index sont préalablement créées pour effectuer le balayage indirect. Dans ce cas, il doit y avoir sélection d'un chemin entre le balayage simple et le balayage indirect, et détermination de la relation index à utiliser comme un chemin d'accès dans le cas du balayage indirect.

Expliquons ceci en prenant des exemples; supposons qu'il y ait trois relations index ayant comme clés secondaires ITYPE, IETAT, et ITGE créées sur la relation de base WAGON (figure 2.4). Leur clé correspond respectivement à l'attribut TYPE, à ETAT et à l'ensemble des attributs TYPE, GARE et ETAT (figure 3.1) et soient les quatre conditions de balayage suivantes;

1) ETAT = "libre"

2) TYPE = "frigo"

3) TYPE = "plat" et ETAT = "attaché" et GARE = "tour"

4) TYPE = "couvert" et GARE = "grenoble"

Tout d'abord, en ce qui concerne le balayage simple, il peut être appliqué sur n'importe quelle condition de balayage, car la verification de la condition donnée a lieu pour tous les n-uplets d'une relation de base après avoir accédé aux pages qui les stockent.

ITYPE

CLE	PTR
citerne	s2
citerne	s5
citerne	s12
citerne	s14
citerne	s15
citerne	s18
couvert	s1
couvert	s6
couvert	s7
couvert	s10
couvert	s16
frigo	s8
frigo	s9
frigo	s11
frigo	s17
frigo	s19
plat	s0
plat	s3
plat	s4
plat	s13

IETAT

CLE	PTR
attaché	s1
attaché	s3
attaché	s4
attaché	s6
attaché	s11
attaché	s18
chargé	s0
chargé	s2
chargé	s5
chargé	s7
chargé	s15
chargé	s16
chargé	s19
libre	s8
libre	s9
libre	s10
libre	s12
libre	s13
libre	s14
libre	s17

ITGE

CLE	PTR
citerne angers chargé	s5
citerne grenoble attaché	s18
citerne grenoble chargé	s2
citerne grenoble libre	s12
citerne marseille chargé	s15
citerne marseille libre	s14
couvert bordeaux chargé	s16
couvert grenoble attaché	s6
couvert lyon attaché	s1
couvert paris libre	s10
couvert tours chargé	s7
frigo angers attaché	s11
frigo angers libre	s9
frigo narbonne libre	s8
frigo paris libre	s17
frigo tours chargé	s19
plat marseille chargé	s0
plat paris libre	s13
plat paris attaché	s3
plat tours attaché	s4

Figure 3.1 Schéma logique de trois relations index ITYPE, IETAT et ITEG

Pour le balayage portant sur la condition 1), nous pouvons accéder aux n-uplets la satisfaisant par l'intermédiaire de la relation index IETAT; l'accès à IETAT, nous fournit les IDTs adéquats et nous arrivons ainsi aux n-uplets de la relation de base. Le choix entre cette méthode et le balayage simple n'est pas évident car les relations index de MIMER ne sont que des index "non-groupants". Nous discuterons ceci ultérieurement.

Pour la condition de balayage 2), Il y a deux relations index qui peuvent être utilisées; ITYPE et ITGE. Dans le cas de ITGE, on peut obtenir les n-uplets en ignorant les valeurs des attributs clé ETAT et GARE.

Nous avons trois relations index pour le cas 3). Ce sont ITYPE, IETAT et ITGE. Pour l'utilisation des relations ITYPE et IETAT, il faut encore vérifier le reste de la condition <GARE = "tour" et ETAT = "attcahé"> et <TYPE = "plat" et GARE = "tour">.

Dans le cas des deux balayages 2) et 3), la relation index qui correspond exactement à l'ensemble des attributs, qui se trouvent dans la condition, (appelés attribut de condition) sera naturellement choisie.

Pour la dernière 4), on peut utiliser ITYPE et ITGE comme index. Il est évident qu'il y a une formule atomique à évaluer <GARE = "grenoble"> lors d'une sélection de ITYPE. Par contre, ITGE peut être utilisé sans tenir compte de l'attribut-clé ETAT. Nous devons alors choisir un des deux si nous voulons utiliser un balayage indirect.

On constate ici qu'un ensemble de chemins d'accès possibles pour un balayage conditionnel sur une relation de base peut être obtenu intuitivement. Nous allons formaliser ceci dans le paragraphe suivant.

3.3.2 Utilisation d'une relation index

A l'aide de l'exemple présenté précédemment, nous pouvons déduire deux conditions nécessaires pour qu'une relation index soit "utilisable" dans un balayage conditionnel;

1) L'intersection I entre l'ensemble des attributs-clé d'une relation index et celui des attributs de la condition, ne doit pas être nulle.
2) Les attributs de l'intersection I doivent être les m premiers attributs-clé dans l'ordre de sa définition.

Appliquons-ceci sur la condition 1) dans l'exemple précédent; bien que l'intersection I soit ETAT , la relation index ITGE n'est pas utilisable, car l'attribut ETAT n'est pas le premier dans l'ordre de la définition de la clé de ITGE. Par contre, pour les conditions 3) et 4), IETAT est utilisable car l'attribut ETAT est seul attribut-clé.

Enfin, un ensemble de chemins disponibles, c'est-à-dire de relations index utilisables peut être déterminé par l'application des conditions présentées. Il ne reste qu'à faire le choix d'un chemin ayant les meilleures performances parmi eux, y compris le balayage simple.

3.3.3 Evaluation du coût d'accès

Le choix d'un chemin se fait par l'estimation du coût d'accès de chaque chemin utilisable pour les raisons suivantes:

- Les relations de base sont d'abord stockées comme un tas de données.
- Les relations index sont créées sans changement du stockage de leur relation de base, elles sont donc index non-groupants.

C'est alors le chemin optimal qui a le moindre coût estimé parmi les chemins utilisables. Nous commençons par le balayage simple, qui peut être adopté dans tous les cas de balayage conditionnel.

3.3.3.1 Balayage simple

C'est un chemin évident et simple, qui est toujours disponible. Comme les relations de base ont une organisation en tas de données en premier, un tel balayage effectue les accès aux pages où se trouvent les n-uplets de la relation de base.

Après avoir lancé ce balayage, l'accès commence par la première page de la relation de base sur laquelle le balayage est défini. A partir d'ici, on peut récupérer le premier n-uplet de la relation. Si la condition de balayage à verifier existe et le n-uplet considéré la satisfait, ce sera un n-uplet à conserver. Le n-uplet suivant dans la même page doit être accédé. Dans le cas où il n'y a plus de n-uplet dans la page actuelle, on passe à la page suivante. Ce processus est poursuivi jusqu'au dernier n-uplet de la dernière page afin de récupérer tous les n-uplets de la relation.

Une relation est découpée en pages de taille fixe TUPAG, à chaque page est associé par une transformation un bloc et un seul en mémoire secondaire. Une page ne possède que les n-uplets d'une et une seule relation (figure 2.1).

Si les trous peuvent être négligés (on a par ailleurs un

moyen de maintenir les pages compactes), nous pouvons estimer le nombre des pages occupées NBPAGE(R) pour stocker une relation R à partir de sa cardinalité CARD(R) et de sa longueur LNT(R) grâce au catalogue du système;

NBPAGE(R) = (CARD(R) div (TUPAG div LNT(R))) + 1

Le coût d'accès peut être considéré comme le temps d'exécution estimé. Le coût pour l'accès aux pages d'une relation R de la mémoire secondaire est (Cs * NBPAGE(R)) où Cs est le temps pour le transfert d'un bloc en mémoire centrale.

Dans le cas du balayage conditionnel, il est en plus nécessaire d'ajouter le coût de la vérification de la condition donnée. Cette verification se fait à chaque n-uplet et son coût peut varier en fonction du nombre de formules atomiques NFA constituant la condition de balayage et de la longueur de l'attribut qui se trouve dans les formules atomiques. Soit Ct le temps moyen d'évaluation d'une formule atomique, le coût d'évaluation de la condition de balayage pour un n-uplet est (Ct*NFA).

Or nous ne traitons que le cas où les formules atomiques sont liées par l'opérateur logique de conjonction, car si l'opérateur de disjonction est présent, le balayage simple est tout de suite choisi en raison de la simplicité de traitement. Lorsqu'une des formules atomiques se révéle fausse au cours de la vérification de la condition de balayage, on passe à le n-uplet suivant. L'évaluation de toutes les formules atomiques n'est donc pas nécessaire pour chaque n-uplet.

Soit FAi la ième formule atomique dans la condition de balayage et S(FAi) la possibilité de satisfaire FAi pour un n-uplet d'une relation R, c'est-à-dire la séléctivité d'une formule atomique. La comparaison pour la vérification des formules atomiques a lieu (1 + S(FA1) + S(FA1)*S(FA2) + ... + S(FA1)*S(FA2)*...*S(FAn-1)) fois. Nous pouvons obtenir ou bien estimer cette sélectivité d'une formule atomique à l'aide du catalogues des relations index. Par exemple pour la condition 4) dans l'exemple de 3.3.1, le nombre moyen de comparaison sera de (1 + 1/4) pour chaque n-uplet. Dans le cas où l'on ne trouve que la relation index Ri ayant une clé combinée et qu'on ne peut obtenir aucune séléctivité pour chaque formule atomique, le nombre de comparaisons est estimé en supposant que la sélectivité de chaque formule atomique est égale à:

$$1 + S(FA1) + S(FA1)*S(FA2) + \ldots + S(FA1)*S(FA2)*\ldots*S(FAn-1)$$

$$= 1 + S(FA1) + S(FA1)^{**}2 + S(FA1)^{**}(NFA-1)$$

(car S(FA1) = S(FA2) = ... = S(FAn-1))

$$= (1 - S(FA1)^{**}NFA) / (1 - S(FA1))$$

$$= (1 - 1/NDC(Ri)) / (1 - e^{-Ln(NDC(Ri))/NAC(Ri)})$$

(car S(FAi)**NDC(Ri) = 1/NDC(Ri) où NDC(Ri) est le nombre de clés différentes et NAC(Ri) le nombre des attributs-clé de Ri)

Dans le cas où on ne peut ni l'obtenir ni l'estimer, on lui donne la valeur 0,2. Une fois le nombre de vérifications de formules atomiques NVFA est estimé, le coût de l'évaluation d'une condition de balayage pour tous les n-uplets d'une relation R est (Ct*NVFA*CARD(R)).

Comme le coût total est considéré comme la somme du coût d'accès à la mémoire secondaire et du coût d'évaluation d'une

condition de balayage simple conditionnel, il est alors égal à Cs*NBPAGE(R) + Ct*NVFA*CARD(R).

3.3.3.2 Balayage indirect

Le balayage indirect est défini par le parcours d'une relation index préalablement créée sur une relation de base, qui fournit les adresses des n-uplets (IDTs) désirés de la relation de base. Donc le balayage indirect se compose d'abord de l'accès à la relation index, puis de l'accès aux n-uplets pointés par la relation index.

Nous avons défini les relations index "utilisables" dans un balayage conditionnel. Il y a différents cas pour l'intersection I définie dans les conditions nécessaires;

1) I est égal à l'ensemble des attributs des conditions,
2) I est strictement inclus dans celui-ci,
3) I est égal à l'ensemble des attributs-clé de l'index,
4) I est strictement inclus dans celui-ci.

Dans le cas 2), après le parcours de la relation index, il reste une partie de la condition à évaluer, Dans le cas 4), il faut ignorer les attributs-clé qui ne sont pas dans l'intersection I.

L'évaluation du coût pour l'accès à la relation index (ayant actuellement une structure de B-arbre) peut être obtenu à partir des statistiques sur catalogue des relations index. Soit

PROF(Ri) le profondeur du B-arbre représantant une relation index Ri, MNCN(Ri) le nombre moyen des clés stockées dans un B-arbre (obtenu en divisant le nombre de clés existant par le nombre de noeuds du B-arbre NN(Ri)) et NAC(Ri) le nombre d'attributs-clé. La recherche d'une clé dans une relation index peut être estimée à $((Cs + Ct*NAC(Ri)*Log_2 MNCN(Ri))*PROF(Ri))$.

Après avoir obtenu les IDTs des n-uplets contenant une clé donnée, Les n-uplets doivent être récupérés dans les pages de la relation de base. Pour estimer cela, il faut obtenir la sélectivité d'un ensemble de formules atomiques dont l'attribut est un élément de l'intersection I. La sélectivité d'une condition est définie comme la proportion des n-uplets satisfaisant la condition, par rapport à tous les n-uplets participant aux comparaisons. Le nombre de clés différentes NDC(Ri) et le nombre d'attributs-clé NAC(Ri) d'une relation Ri permettent de l'estimer. Dans le cas 3), les opérateurs de chaque formule atomique étant l'égalité ("="), on peut dire que la sélectivité est égal à 1/NDC(Ri). Dans le cas 4), on part de l'hypothèse que chaque attribut-clé a la même possibilité d'avoir une valeur donnée. Si m attributs constituent l'intersection I, la sélectivité S(I) d'un ensemble des formules atomiques données I peut être obtenue ainsi:

$S(FAi)^{NAC(Ri)} = 1/NDC(Ri)$ où S(FAi) est la sélectivité d'une formule atomique

$$\text{or } S(I) = S(FAi)^{m}$$

$$\text{donc } S(I) = e^{-m*LN(NDC(Ri))/NAC(Ri)}$$

Pour la condition représentée sous forme d'un intervalle, par exemple <NO-WAGON >= "4001" et NO-WAGON <= "5000"> ou <NO-WAGON > "3000">, la sélectivité pourra être estimée en supposant que la distribution des clés est uniforme. Soient MAXCLE(Ri) (respectivement MINCLE(Ri)) la valeur maximale (respectivement minimale) des clés de la relation index Ri et VL(FAi) (VU(FAi)) la valeur de la borne inférieure (supérieure) de la formule atomique FAi;

$$S(FAi) = (VU(FAi) - VL(FAi))/(MAXCLE(Ri) - MINCLE(Ri))$$

Dans le cas d'un intervalle semi-borné, VU(FAi) (VL(FAi)) peut être remplacé par MAXCLE(Ri) (MINCLE(Ri)). Nous récapitulons dans la figure 3.2 la sélectivité en fonction des attributs des conditions de balayage, des attributs-clé de la relation index utilisée et de la nature de la condition de balayage.

type de condition de balayage	utilisation de attributs-clé	sélectivité S(FA)
les m opérateurs de comparaison sont "="	m = NAC(Ri)	1/NDC(Ri)
	m < NAC(Ri)	$-m*LN(NDC(Ri))/NAC(Ri)$ e
recherche des clés dans un intervalle	m = 1	$\frac{VU(FA) - VL(FA)}{MAXCLE(Ri) - MINCLE(Ri)}$
	m > 1 et m-1 opérateurs de comparaison sont "="	$-(m-1)*LN(NDC(Ri))/NAC(Ri)$ e

Figure 3.2 Calculs de sélectivité

Enfin nous pouvons estimer le nombre de n-uplets NTA(I,R) de la relation de base R à accéder;

$$NTA(I,R) = CARD(R)*S(I)$$

S'il y a plusieurs IDIs, l'accès aux noeuds feuilles du B-arbre

est nécessaire soit: Cs*NTA(I,R)/NMCN(Ri).

Comme les relations index sont non-groupantes, dans le pire des cas on a NTA(FA,R) accès aux pages de la relation de base pour obtenir les n-uplets ayant une clé donnée (un accès à page pour un n-uplet).

Enfin, dans le cas 2), il est nécessaire d'évaluer le reste de la condition de balayage sur les n-uplets déjà obtenus par accès à la relation index. Le coût suppémentaire demandé est celui de la vérification du reste de la condition de balayage comme dans le cas du balayage simple.

Le coût d'accès d'un balayage conditionnel sur une relation R par l'intermédiaire d'une relation index Ri est donc la somme de:

i) parcours du B-arbre constituant la relation index;
si la clé est unique et les opérateurs de comparaison "="
PROF(Ri)*(Cs + Ct*NAC(Ri)*Log_2(NMCN(Ri)))
sinon
2*PROF(Ri)*(Cs + Ct*NAC(Ri)*Log_2(NMCN(Ri))).

ii) A partir du nombre de n-uplets à obtenir en accédant aux pages de la relation de base, après avoir estimé la sélectivité de l'ensemble des formules atomiques, l'accès aux pages de la relation de base et aux noeuds feuille du B-arbre est
Cs*(NTA(I,R) + NTA(I,R)/NMCN(Ri))

iii) S'il reste des formules atomiques de la condition de balayage à évaluer;

Ct*NTA(I,R)*S((F-I),R)

3.3.3.3 Sélection d'un chemin

Naturellement le chemin dit optimal sera un chemin de coût minimum parmi tous ceux estimés. L'estimation du coût d'accès d'un chemin est généralement propre à chaque système, puisque les chemins d'accès sont fortement liés à la structure physique des données et à leurs méthodes d'accès. En outre les informations statistiques sur la base sont souvent insuffisantes pour l'estimer.

Les formules que nous avons présentées, sont définies en essayant d'utiliser au maximum les données accèssibles dans les catalogues pour une estimation plus réaliste. Mais nous avons des paramètres à déterminer dans les formules. Ce sont Ct et Cs qui sont très dépendants de la configuration matérielle et logicielle du système utilisé. En outre la mesure des deux paramètres est peu facile sur un système d'exploitation. Nous avons quand même essayé d'obtenir la valeur relative de Ct par rapport à Cs au lieu des valeurs exactes <AST 76> en faisant des mesures, en faisant varier des formules atomiques. On a choisi le coefficient 0,05 de Ct par rapport à Cs pour notre système.

Nous terminerons par les exemples d'application de ce que nous avons présenté; la figure 3.3 montre une partie des

catalogues de notre base ferroviaire et décrit les informations relatives aux chemin d'accès pour la relation WAGON. Nous considérons trois questions Q1, Q2 et Q3, exprimées en MIQUEL, qui sont respectivement:

Q1) SELECT *
FROM WAGON
WHERE CAPACITE = 15 ;
(on cherche les wagons de capacité 15 tonnes)

Q2) SELECT *
FROM WAGON
WHERE CAPACITE >= 10 AND CAPACITE <= 25 ;
(on cherche les wagons dont la capacité est comprise entre 10 et 25)

Q3) SELECT *
FROM WAGON
WHERE TYPE = "frigo" AND POID >= 12
AND CAPACITE >= 10 AND CAPACITE <= 25 ;
(on cherche les wagons du type frigo, dont la capacité est comprise entre 10 et 25, et dont le poids est supérieur à 12)

RELATIONS

nom	CARD(R)	LNT(R)	NBAPGE(R)
WAGON	2000	30	125

nom	index sur	unique	NDC(Ri)	PROF(Ri)	NN(Ri) /NMCN(Ri)	MAXCLE(Ri) /MINCLE(Ri)
ITYPE	TYPE	non	200	3	74/27	
ICAP	CAPACI	non	400	2	32/62	5/50

Figure 3.3. Description d'une partie de la base

Pour chaque requête, il y a différents chemins d'accès possibles, à savoir:

Pour Q1,
1) balayage simple
2) balayage indirect par l'intermédiaire de la relation index ICAP

Pour Q2,
1) balayage simple

2) balayage indirect par ICAP

Pour Q3,
1) balayage simple
2) balayage indirect par ICAP
3) balayage indirect par ITYPE

La figure 3.4 donne les résultats des calculs que nous venons de décrire, appliqués aux trois questions Q1, Q2 et Q3. En ce qui concerne Q1, on constate que le chemin optimal consiste à utiliser la relation index sur CAPACITE. Cela est d'ailleurs prévisible pour la requête, qui comporte un prédicat d'égalité sur CAPACITE. Cependant, si l'on considère la question Q2 comportant elle aussi une condition de balayage sur CAPACITE avec une plage de valeurs, on constate que le chemin optimal consiste à utiliser le balayage simple sur la relation de base. Ceci est dû au fait que l'ordre physique des n-uplets n'est pas celui qui correspond à l'ordre (ascendant ou descendant) de l'attribut CAPACITE.

Le choix est effectué au moment de la recherche du premier n-uplet. Pour cela, MIMER contient deux procédures supplémentaires: l'une sert à choisir le chemin d'accès en utilisant les techniques décrites, l'autre remplit le curseur afin de garder le chemin d'accès choisi.

requête	chemin d'accès	coût			choix
		accès à MS	évaluation	total	
Q1	1)	125	2000	125 + 0,05x2000	optimal
	2)	2x2 + 2000x1/400(1+1/62)	2x(⌈log$_2$62⌉+1)	9,08 + 0,05x12	
Q2	1)	125	2000x(1+(50-10)/45)	125 + 0,03x3778	optimal
	2)	2x2 + 2000x1/3x(1+1/62)	2x(⌈log$_2$62⌉+1)	681,4 + 0,05x12	
Q3	1)	125	2000x(1+1/3+1/3x0,2)	125 + 0,05x2800	optimal
	2)	2x2 + 2000x1/3x(1+1/62)	2x(⌈log$_2$62⌉+1) + 2000x1/3x(1+1/200)	681,4 + 0,05x686	
	3)	2x3 + 2000x1/200x(1+1/27)	3x(⌈log$_2$27⌉+1) + 2000x1/200 x(1+40/45+40/45x0,2)	16,4 + 0,05x35,7	

Figure 3.4 Calculs de coût des chemins d'accès

3.4 Choix d'un chemin d'accès pour les opérateurs de l'algèbre relationnelle

Nous venons de voir le choix d'un chemin effectué sur une seule relation avec une condition de balayage donnée. Cela correspond à l'opérateur de l'algèbre relationnelle SELECT. Cependant, nous savons qu'un facteur limitatif essentiel aux performances des systèmes de gestion de bases de données relationnels vient du traitement des questions portant sur plusieurs relations. L'opération de jointure peut s'y trouver très fréquemment. Les paragraphes suivants sont essentiellement consacrés à l'opération de jointure et les manières d'accomplir son exécution.

La θ-jointure <COD 70> de deux relation R et S selon les attributs A de R et B de S est la relation résultat T obtenu en concaténant tout n-uplet de R avec tout n-uplet de S, tels que AθB où θ est un des opérateurs =, <, >, <=, >=.

Cette opération peut être très coûteuse en temps d'exécution. La jointure de deux relations est également à faire dans le cas de l'intersection de deux relations, où chaque élément de la première relation est comparé à chaque élément de la seconde. Ceci peut conduire pour deux relations de n éléments à un coût proportionnel à (n**2).

C'est la raison pour laquelle de récentes solutions ont été proposées en isolant les fonctions de gestion de base de

données du calculateur hôte, et en les regroupant dans une machine spécialisée, appelée machine base de données <VAL 81>. Par contre, les systèmes tels que MICROBE, n'ayant aucun dispositif spécial pour les fonctions de gestion de base de données, optimisent leurs performances un cherchant d'autres méthodes de jointure moins coûteuses pour un environnement donné. La synthèse des diverses méthodes de jointure est présentée dans <VAL 81>.

3.4.1 Les méthodes de jointure dans MICROBE

L'exécution d'un opérateur de jointure dans MICROBE est basée sur les trois méthodes: méthodes des boucles imbriquées <GOT 75>, méthode de fusion <AST 76> et méthode par index <GOT 75> pour les raisons suivantes;

- l'insuffisance des tampons dans la mémoire principale,
- le manque d'algorithme de hachage,
- l'absence d'index groupant,
- la façon de stocker une relation.

3.4.1.1 Méthode des boucles imbriquées

La méthode la plus triviale <GOT 75> consiste à composer le produit cartésien des deux relations, où la valeur des attributs de jointure de chaque n-uplet de la première est comparée à celle de la seconde. Nous rappelons cet algorithme dans la figure 3.5.

```
procedure  JBI(R,S)
     pour chaque n-uplet de R faire
          pour chaque n-uplet de S faire
               si r.A θ s.B  alors  former la jointure
fin  JBI
```

Figure 3.5 Algorithme par produit cartésien

Une telle procédure exige intuitivement (NBPAGE(R) + NBPAGE(R)*NBPAGE(S)) accès à la mémoire secondaire. En outre, l'évaluation du prédicat de la jointure nécessite CARD(R)*CARD(S).

Cet algorithme est en général prohibitif, d'où l'interêt de developper des méthodes plus efficaces. Mais il peut être employé dans certains cas, par exemple quand deux relations de jointure sont petites du point de vue du nombre de pages et de leur cardinalité.

3.4.1.2 Méthode de fusion

Cette méthode est proposée dans SYSTEM-R, lorsque les relations de jointure ne disposent pas de chemin d'accès via les attributs de jointure <AST 76> <BLA 77>. L'idée de cet algorithme est le tri des deux relations sur les attributs de jointure, suivi de leur fusion sur les attributs de jointure pour réaliser l'opération. Comme une relation index permet l'accès ordonné sur ses attributs-clé à la relation de base sur laquelle elle est

définie, nous pouvons adapter cette idée pour une méthode de jointure dans notre système (figure 3.6).

Algorithme par fusion

1) créer une relation index temporaire sur la relation de jointure si on n'y trouve pas de relation index sur les attributs de jointure.
2) balayer les deux relations index simultanément en fusionnant les deux relations de jointure et former la jointure.

Figure 3.6 Algorithme par fusion

Une telle procédure demande la création dynamique de relations index en l'absence de la relation index correspondant aux attributs de jointure. Soit Ri la relation index temporaire créée sur une relation de jointure R, les accès à la mémoire secondaire pour la création de Ri peuvent être formulé en uilisant ceux décrit pour une insertion de clé dans §2.3.4.1:

$$\sum_{i=1}^{CARD(R)} PROFi(Ri)+1+4k$$

où PROFi(Ri) est la profondeur actuelle du B-arbre et k (0<=k<=PROFi(Ri)) le nombre d'éclatements lors de la ième insertion.

Dans la figure 2.17, on constate que le nombre de noeuds du B-arbre est très variable pour un même nombre de clés. Pour estimer les accès à la mémoire secondaire nécessaires à la création dynamique d'une relation index temporaire, nous partons de l'hypothèse que le B-arbre maintient son taux de remplissage moyen à 75%. Soit (D(Ri) - 1)*0.75 clés dans un noeud, où D(Ri)

est l'ordre du B-arbre pour la relation index temporaire Ri (voir 2.2.2.2). L'estimation du nombre de noeuds nécessaires NN(Ri) pour construire le B-arbre de la relation index temporaire est important, car il est en fonction du nombre d'éclatement qui ont lieu tout au long des insertions.

Soit Lc(Ri) la longueur d'une clé formée des attributs de jointure, le nombre moyen de cléa dans un noeud D'(Ri), le nombre de noeuds NN(Ri), et la profondeur PROF(Ri) sont estimés comme suit;

$$D(Ri) = (TUPAG + Lc(Ri) - Lt) \text{ div } (Lp + Lc(Ri))$$

$$D'(Ri) = (D(Ri) - 1)*0.75$$

$$NN(Ri) = CARD(R) \text{ div } D'(Ri)$$

$$PROF(Ri) = \text{Log}_{D'(Ri)} CARD(R) + 1$$

Finalement pour créer une relation index Ri sur une relation R, l'accès à la mémoire secondaire peut être approximativement calculé par

$$\sum_{i=1}^{CARD(R)} (PROFi(Ri) + 1 + 4k) + NBPAGE(R)$$

$$= CARD(R)*(PROF(Ri) + 1) + 4*NN(Ri) + NBPAGE(R)$$

Pour le fusionnement (étape 2) de l'algorithme), l'accès ordonné ascendant aux clé de la relation index représente l'accès aux noeud-feuilles de son B-arbre. Le nombre de noeud feuille NNF(Ri) d'une relation index Ri d'une relation R peut être obtenu par (CARD(Ri) div (D'(ri) + 1)). L'accès ordonné aux n-uplets exige alors (CARD(Ri) + NNF(Ri)) accès à la mémoire secondaire.

Enfin, étant données deux relations R et S, cet algorithme de jointure demande un nombre d'accès à la mémoire secondaire égal à:

NBPAGE(R) + CARD(R)*(PROF(Ri) + 1) + 4*NN(Ri)
Si on ne trouve pas de relation index correspondant aux attributs de jointure sur la relation R

+

NBPAGE(S) + CARD(S)*(PROF(Si) + 1) + 4*NN(Si)
Si on ne trouve pas de relation index correspondant aux attributs de jointure sur la relation S

+

CARD(R) + NNF(Ri) + CARD(S) + NNF(Si)

3.4.1.3 Méthode par index

L'idée de l'algorithme que nous avons employé, vient de Gothlieb <Got 75>, mais avec une seule relation index. C'est-à-dire qu'on obtient d'abord un n-uplet d'une relation de jointure, soit S, puis on recherche un n-uplet de l'autre relation, soit R, par l'intermédiaire de l'accès à la relation index. Si celle-ci n'existe pas, on la crée dynamiquement. Cela est équivalent à CARD(S) fois le balayage indirect conditionnel sur la relation R (voir 3.3.3). Cependant le chemin d'accès est déjà choisi. Nous présentons cet algorithme dans la figure 3.7.

La création dynamique de la relation temporaire pour une des deux relations a déjà été présentée dans 3.4.1.2. Une relation index Ri soit existe préalablement soit est créée dynamiquement sur la relation R. Pour l'étape 2), l'accès à la mémoire secondaire peut être obtenu à l'aide de la formule que

Algorithme de l'accès par index

1) créer une relation index temporaire sur une des deux relations de jointure s'il n'existe pas de relation index correspondant aux attributs de jointure
2) pour chaque n-uplet de la relation S
   chercher les n-uplets satisfaisant le prédicat de jointure par accès à la relation R, par la relation index Ri
   s'ils existent, alors former la jointure

fin

Figure 3.7 algorithme par index

nous avons décrit dans 3.3.3.2;

NBPAGE(S) + CARD(S)*(l'accès pour un balayage indirect conditionnel)

Le nombre d'accès pour un balayage indirect exige soit (2*PROF(Ri) + S(Pj)*CARD(R)), soit (PROF(Ri) + S(Pj)*CARD(Ri)) selon l'unicité des clés de la relation index (S(Pj) est la sélectivité du prédicat de jointure).

La sélectivité S(Pj) du prédicat de jointure peut facilement être obtenue s'il existe une relation index correspondant aux attributs de jointure. Quant à la création dynamique nous pouvons l'estimer à (1/MAX(CARD(R),CARD(S))) <DEL 82>.

3.4.2 Comparaison et Choix

3.4.2.1 Comparaison

Nous venons de décrire les trois méthodes de jointure et leur accès à la mémoire secondaire. Un tableau récapitulatif est présenté dans la figure 3.8. Puis des comparaisons sont faites des coûts d'accès à la mémoire secondaire des méthodes qui ont été décrites.

méthode	environnement	accès à la mémoire secondaire
1) boucles imbriquées		NBPAGE(R) + NBPAGE(R)*NBPAGE(S)
par fusionnement	2) relation index sur les attributs de jointure pour chaque relation	CARD(R) + NNF(Ri) + CARD(S) + NNF(Si
	3) une des deux, (soit sur R)	NBPAGE(S) + CARD(S)*(PROF(Si) + 1) + 4*NN(Si) + CARD(R) + NNF(Ri) + CARD(S) + NNF(Si
	4) aucune donc deux créations dynamiques	NBPAGE(R) + CARD(R)*(PROF(Ri) + 1) + 4*NN(Ri) + NBPAGE(S) + CARD(S)*(PROF(Si) + 1) + 4*NN(Si) + CARD(R) + NNF(Ri) + CARD(S) + NNF(Si
par index	5) une relation index (soit R)	NBPAGE(S) + CARD(S)*(2*PROF(Ri) (ou PROF(Ri)) + S(Pj)*CARD(R))
	6) une création dynamique (soit sur R)	NBPAGE(R) + CARD(R)*(PROF(Ri) + 1) + 4*NN(Ri) + NBPAGE(S) + CARD(S)*(2*PROF(Ri) + S(Pj)*CARD(R))

Figure 3.8 Tableau récapitulatif des méthodes présentées

Il est assez peu facile de comparer une opération réalisée même dans un système dans laquelle elle existe. Nous nous proposons donc de comparer ces méthodes avec les hypothèses simplificatrices énoncées et fixant certains paramètres pour montrer leur influence. Le nombre d'accès à la mémoire secondaire est évalué en fonction de la cardinalité des relations de jointure est représenté par un graphe dans la figure 3.9. Les paramètres intervenant également pour chaque relation, leur cardinalité est supposée être la même, ceci pour simplifier.

Observations

Les méthodes présentées dans le tableau de la figure 3.8 sont comparées. Au vu du nombre d'accès à la mémoire secondaire, les formules de la figure 3.8 nous font remarquer les points suivants:

- La longueur d'un n-uplet domine présque le nombre d'accès à la mémoire secondaire. Mais la longueur des attributs de jointure l'influence peu. C'est-à-dire que la taille des relation de jointure est le premier facteur affectant le nombre d'accès à la mémoire secondaire.
- L'algorithme des boucles imbriquées exige moins d'accès à la mémoire secondaire pour de petites tailles de jointure.
- Dans tous les cas, l'algorithme par fusion a les meilleures performances, si deux relations index sur les attributs de jointure existent pour les relations de jointure, sauf quand elles ne sont pas petites.
- Le nombre d'accès à la mémoire secondaire pour les méthodes par index 5) et 6) peut varier en fonction de la

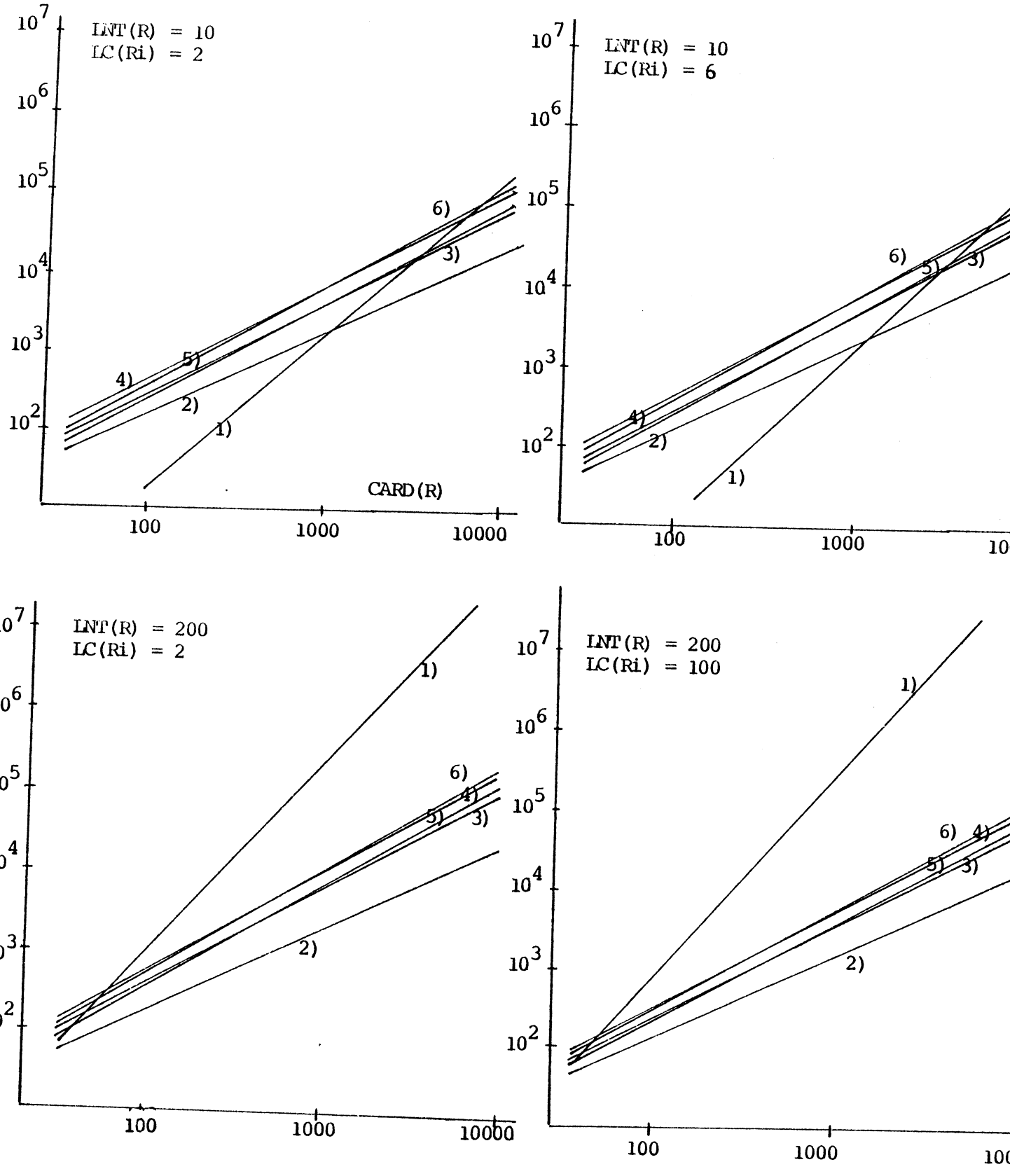

LNT(R) = 10
LC(Ri) = 2
CARD(R)
LNT(R) = 10
LC(Ri) = 6
LNT(R) = 200
LC(Ri) = 2
LNT(R) = 200
LC(Ri) = 100
1)
2)
3)
4)
5)
6)
100
1000
10000

Figure 3.9

sélectivité du prédicat de jointure. Comme nous avons pris sa valeur égal à 1/MAX(CARD(R),CARD(S)), le nombre d'accès à la mémoire secodaire de la méthode 3) est bien proche de celui de la méthode 5). En plus la méthode 4) et la méthode 6) sont dans la même situation.

En conclusion, à une méthode de jointure caractérisée par les paramètres: taille de relation, présence de relation index, séléctivité du prédicat de jointure, nature du prédicat de jointure (équi-join ou non), correspond un nombre d'accès à la mémoire secondaire pour l'exécution d'une jointure.

3.4.2.2 Choix d'une méthodc de jointure

Dans les paragraphes précédents, le nombre d'accès à la mémoire secondaire des méthodes de jointure est analysé. Cela joue un rôle essentiel pour en choisir une. D'ailleurs, il ne faut pas négliser l'évaluation des prédicats de jointure.

Cette évaluation est faite pour les valeurs des attributs de jointure de chaque relation. Donc ceci est fonction de la cardinalité des deux relations, le nombre de prédicats élémentaires (par substitution des valeurs des attributs de jointure d'une relation, il devient une formule atomique définie 3.2.1) et la méthode de jointure. Par exemple, nous pouvons le faire facilement pour la méthode des boucles imbriquées; l'évaluation du prédicat de jointure est CARD(R)*CARD(S). Dans le cas de la méthode de fusion, la comparaison sur les attributs de

jointure d'une paire de n-uplets doit avoir lieu deux fois, pour déterminer l'avancement du balayage ordonné d'une des deux relations. Lorsque la méthode par index est choisie, dès qu'un ensemble de n-uplets correspondant aux valeurs des attributs de jointure est fixé, aucune évaluation n'aura pratiquement lieu.

Nous formulons ce que nous venons de décrire comme suit:

i) méthode des boucles imbriquées; le coût C est CARD(R)*CARD(S)*NFA(Pj) où NFA(Pj) est le nombre de formules atomiques dans le prédicat de jointure Pj

ii) méthode de fusion; avec la supposition, que les n-uplets des deux relations participent deux fois à la comparaison,

C = 2*(CARD(R) + CARD(S))*NFA(Pj)

iii) méthode par index; la comparaison n'a lieu qu'au moment de l'accès au B-arbre, donc

C = CARD(S)*(2*PROF(Ri) ou PROF(Ri)) *Log_2D(Ri)*NFA(Pj)

Note: la jointure s'exécute dans la plupart des cas sur un petit nombre d'attributs de jointure. D'ailleurs le traducteur n'en recoit que deux, le nombre de comparaison pour un paire de n-uplet est alors traité comme précédemment.

Les formules ci-dessus nous montrent que le nombre de comparaisons de la méthode par index est inférieur à celui de la méthode par fusion. Alors nous préférons choisir la méthode 5) au lieu de la méthode 3), (la méthode 6) au lieu de la méthode 4)). Les quatre méthodes différentes de jointure 1), 2), 5), et 6) sont disponibles dans notre système. La figure 3.10 représente les méthodes possibles en fonction de l'existence de relations index sur les attributs de jointure, et nous y voyons facilement l'adaptation des diverse méthodes de jointure.

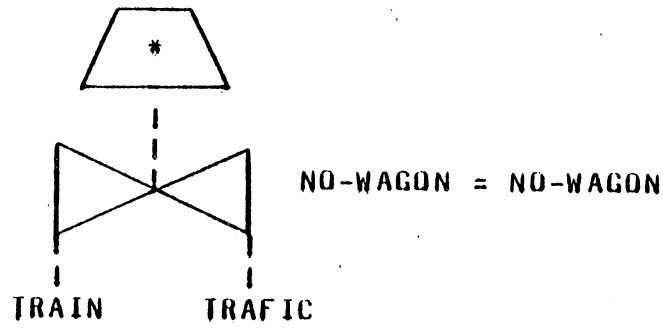
méthode	l'existence de relation index		
	aucune	une des deux	deux
boucles imbriquées	0	0	0
par fusionnement	X	X	0
Par index	0	0	X

Figure 3.10 Les méthodes disponibles selon l'existence de relation index sur les attributs de jointure

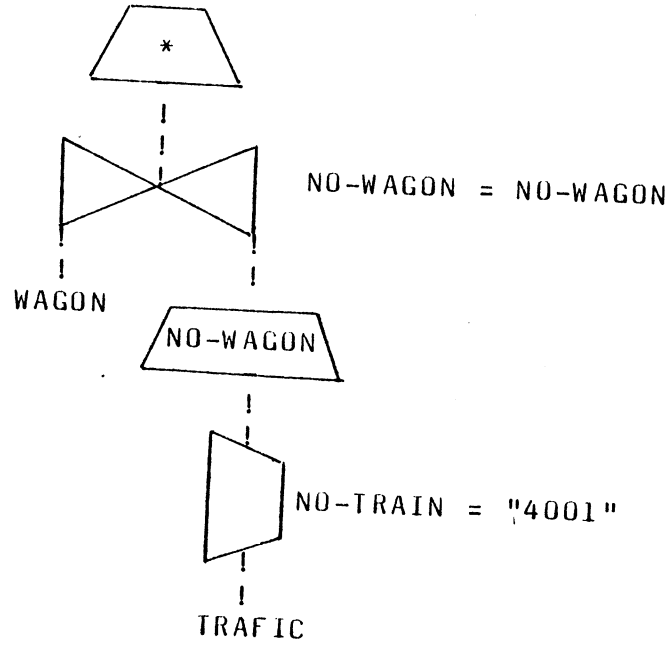
Pour conclure, le choix d'une méthode de jointure optimale est celle qui a le moindre coût d'exécution estimé parmi les méthodes qui sont disponibles.

Appliquons les méthodes décrites aux requête suivantes. La figure 3.11 montre une partie des catalogues de notre base ferroviaire et décrit les informations relatives aux relations index pour les relations WAGON, TRAIN et TRAFIC. Nous considérons trois questions Q4, Q5 et Q6, exprimées en MIQUEL et traduites en arborescence des opérateurs d'algèbre relationnelle, qui sont réspectivement:

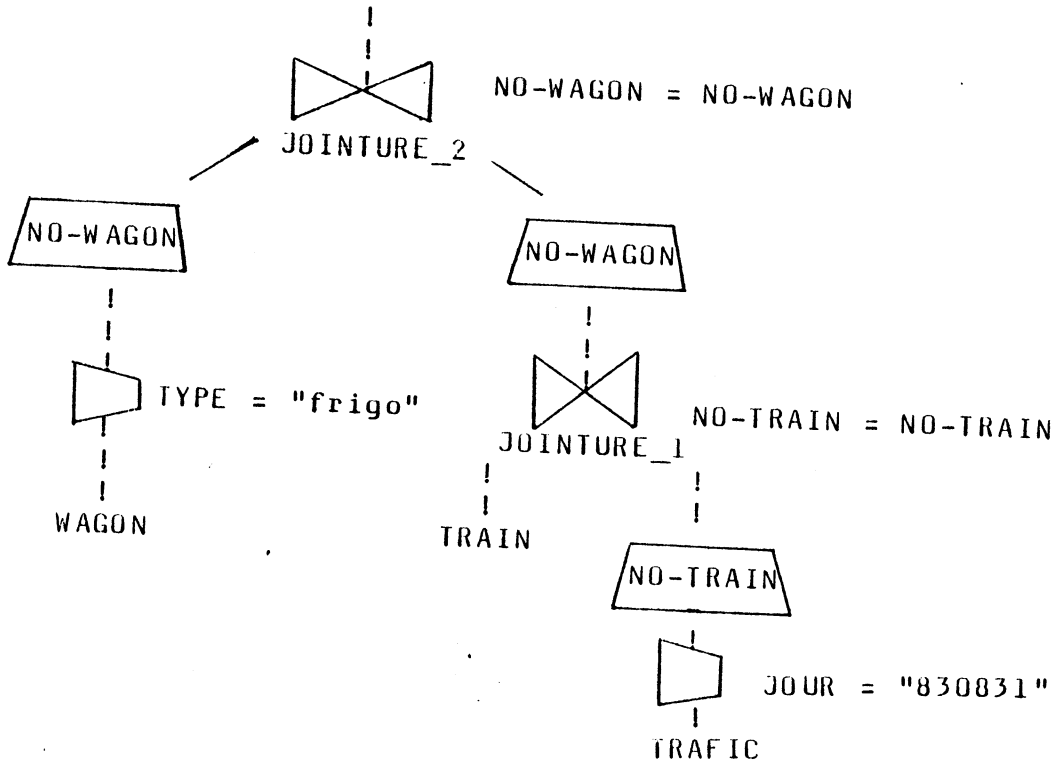
```
Q4  SELECT *
    FROM   TRAIN
    WHERE  NO-TRAIN = SELECT NO-TRAIN FROM TRAFIC ;
    (on cherche les trains qui sont dans la relation trafic)
```



Q5
```
SELECT *
FROM   WAGON
WHERE  NO-WAGON = SELECT NO-WAGON
                  FROM   TRAIN
                  WHERE  NO-TRAIN = 4001 ;
```



Q6
```
SELECT NO-WAGON
FROM   WAGON
WHERE  TYPE = "frigo"
  AND  NO-WAGON = SELECT NO-WAGON
                  FROM   TRAIN
                  WHERE  NO-TRAIN = SELECT NO-TRAIN
                                    FROM   TRAFIC
                                    WHERE  JOUR = "830831"
```
(on cherche les numéros des wagons de type frigo qui passeront le 830831)



RELATION

NOM	CARD(R)	LNT(R)	NBPAGE(R)
WAGON	20000	30	1250
TRAIN	6000	6	72
TRAFIC	8000	12	191

RELATION INDEX

Relation de base	index sur	unique	NDC(Ri)	LC(Ri)	D'(Ri) /D(Ri)	PROF(Ri)	NN(Ri)	NAF(Ri)
WAGON	NO-WAGON	oui	20000	2	60/81	3	333	327
	TYPE	non	200	8	30/41	3	660	645
TRAIN	NO-TRAIN	non	600	2	60/81	3	100	97
	NO-WAGON	oui	6000	2	60/81	3	100	97
TRAFIC	NO-TRAIN	non	2000	2	60/81	3	133	130
	JOUR	non	100	6	36/49	3	222	216

Figure 3.11 Description des relations et de leurs relations index

La figure 3.12 donne les résultats des calculs que nous venons de décrire, appliqués aux trois requête Q4, Q5 et Q6. On y trouve un facteur de 1/20 du temps de comparaison d'une formule atomique par rapport à l'accès à la mémoire secondaire (voir 3.3.3.3). En ce qui concerne Q4 et la première jointure de Q6, on constate que le temps de comparaison joue un rôle décisif, malgré la supériorité du nombre d'accès à la mémoire secondaire. C'est d'ailleurs prévisible au vu de la requête Q5 et la deuxième jointure de Q6.

Pour conclure ce chapitre, nous avons présenté essentiellement l'optimisation en fonction de l'implantation des relations index en B-arbre. Tout d'abord, le choix d'un chemin d'accès sur une seule relation nous a montré une méthode de selection automatique d'un chemin d'accès optimal pour l'opérateur algèbrique SELECT. Ensuite on a essentiellement présenté pour la jointure, la possibilité d'utiliser des chemin

requête	JOINTURE et relations		méthode 1)	méthode 2)	méthode 5)	méthode 6)
Q4			MS: 72+72x191 UC: 6000x8000x0,05	MS: 6000+97+8000+130 UC: 2x(6000+8000)x0,05 optimal	X	X
Q5	(NO-TRAIN="4001") 10*		MS: 1+1x1250 UC: 20000x10x0,05	X	MS: 1+(3+1/20000x20000)x10 UC: 10x(3x($\lceil \log_2 60 \rceil$+1))x0,05 optimal	X
Q6	JOINTURE_1	80*	MS: 1+72x1 UC: 6000x80x0,05	X	optimal MS: 1+(2x3+1/600x6000)x80 UC: 80x(2x3x($\lceil \log_2 60 \rceil$+1))x0,05	X
	JOINTURE_2	80* 100*	MS: 1+1x1 UC: 80x100x0,05 optimal	X	X	MS: 100x(2+1)+4x3+1+1 +(2x2+1/100x100)x80 UC: 2x(2x($\lceil \log_2 60 \rceil$+1))x0,05x80

* cardinalité donnée des relations de jointure

Figure 3.12 Calculs des coûts des méthodes de jointure

d'accès existant ou créés dynamiquement. Nous prouvons que ces optimisations même sur un "petit" SGBD comme MICROBE, peuvent être adaptées, car nécessaires compte tenu du matériel et du logiciel.

CHAPITRE IV

AMELIORATION DES PERFORMANCES DE L'INTERPRETEUR

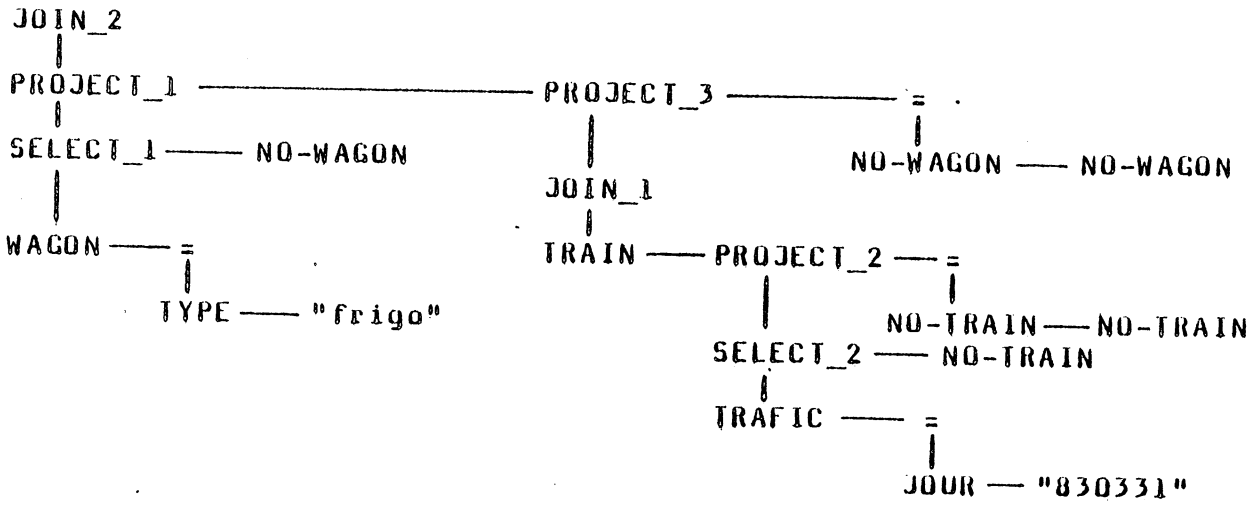
4.1 L'interpréteur

Pour l'exécution d'une requête émise par l'utilisateur, il y deux façons de procéder, comme dans tous les langages de haut niveau:

- par compilation; La requête est traduite en langage machine après les traitements syntaxiques et sémantiques et eventuellement optimisation <HAL 76> <AHO 79>. Cette approche a l'avantage que le résultat obtenu est immédiatement exécutable. Elle est commode dans un système où la requête peut être intégrée dans un autre langage de haut niveau <AST 76>. Il y a aussi facilité pour insérer la requête à une autre, comme sous-requête.
- par interpétation; La requête est généralement traduite en code intermédiaire après son analyse <SMI 75>. Le logiciel de l'interpéteur l'exécute selon les codes produits. Cela offre plus de facilité pour developper un prototype et rend le logiciel portable.

Nous avons choisi l'interprétation dans MICROBE pour les deux raisons précédentes en plus, en raison de l'utilisation interactive du langage MIQUEL. Dans notre système, les codes intermédiaires sont représentés sous forme d'une arborescence. A partir de la requête Q6 dans 3.4.2.2, la figure 3.1 montre l'arborescence que l'interpréteur va recevoir.

MIOPE est l'interpéteur de MICROBE. Il permet de produire le résultat d'une requête en poursuivant à exécuter les



Note: Les opérateurs sont numérotés pour identifier les mêmes opétareurs

Figure 4.1 Arborescence à recevoir

opérateurs de l'arborescence <LEE 81>. Il consiste donc en un ensemble de procédures correspondant à chaque opérateur défini dans le système et une procédure pour le lien entre ces procédures. En plus, il est nécessaire:

- de gérer les relations temporaires au cours de l'interprétation, soit pour garder les résultats intermédiaires, soit pour gérer la description des résultats d'un opérateur,
- d'avoir un mécanisme pour choisir le meilleur chemin d'accès dans un environnement donné,
- de gérer l'interface d'entrée/sortie utilisateur pour les données à insérer et les résultats d'une requête.

Chaque opération mise en oeuvre par l'exécution d'une requête sur son opérande consiste en une série d'appels aux primitives de MIMER. Les opérations effectuées par les opérateurs peuvent être classées selon les opérandes à manipuler:

- Opérations portant sur une base; la création, la destruction, l'ouverture et la visualisation de ses caractéristiques permettant de manipuler la base et de voir toutes les relations créées,
- Opérations portant sur une relation; création, suppression et visualisation de ses caractéristiques, repésentant respectivement l'insertion d'un n-uplet, la suppression d'un n-uplet et l'accès à un n-uplet de la relation cataloguée (RELation MAItre) pour séparer les manipulations concernant les catalogues de celles des relations créées par l'utilisateur,
- Opérations portant sur les attributs; création, suppression, et visualisation de leurs caractéristiques dans une relation donnée,
- Opérations portant sur une relation index; création, suppression et visualisation de ses caractéristiques sur une relation de base,
- Opération sur les n-uplets; insertion, suppression et mise à jour des n-uplets dans une relation donnée.
- Interrogation; représentée par une arborescence qui possède des opérateurs de l'algèbre relationnelle, qui permet d'obtenir des données désirées à partir des relations de l'utilisateurs.

Cette dernière est la fonction principale d'un SGBD. En plus, on peut dire sans exagération que toutes les fonctions concourent à cela. L'arborescence à effectuer consiste en un ensemble d'opérateurs de l'algèbre relationnelle, tel que ceux

définis par CODD <COD 70>, puis modifiés par chaque concepteur <NGU 77>, <DAT 77>, <PAI 78> et <DEL 81> et étendus aux valeurs indéfinies pour mieux s'adapter à la pratique <COD 79>.

L'interpréteur possède donc les opérateurs correspondant à non seulement un ensemble complet d'opérateurs algébriques composé des opérateurs ensemblistes traditionnels UNION, INTERSECTION, DIFFERENCE et PRODUIT CARTESIEN et des opérateurs spéciaux pour manipulation de relations n-aires SELECTION, PROJECTION, JOINTURE et DIVISION, mais aussi un ensemble d'opérateurs de fonctions aggrégats CARD, TOTAL, MOY, MAX, MIN, UNIQUE et ORDER BY <DAT77> <LEE 81>.

Un opérateur prend le relai d'un autre pour repondre à une requête d'interrogation. Les fonctions des opérateurs s'accomplissent soit en accédant directement à la relation comme un opérande, soit en recevant le résultat de l'opérateur opérande. Nous avons déjà présenté dans le chapitre 3, le choix d'une méthode optimale d'exécution des opérateurs, à l'aide des deux opérateurs SELECTION et JOINTURE, pour un environnement donné. Nous allons chercher dans les paragraphes suivants la meilleure façon d'interpréter les opérateurs d'une arborescence.

4.2 Techniques d'interprétation

L'interprétation d'une requête peut être mise en pratique de plusieurs façons différentes. Par la suite, nous allons voir deux méthodes: le mode global et le mode pipe-line.

4.2.1 Mode global

Pour tout opérateur, toutes les données nécessaires sont prêtes avant de lancer l'exécution d'un opérateur. Ces données sont représentées comme les opérandes, et elles sont stockées dans la plupart des interpréteurs sous forme d'une relation temporaire pour les résultats intermédiaires. Si nous considérons une arborescence comme une série d'opérateurs à exécuter, les résultats d'une requête ne peuvent être obtenus qu'après l'exécution de l'opérateur à la racine. Expliquons-le en reprenant l'arborescence de la figure 3.1.

L'interprétation de l'arbrescence de la figure 3.1 peut commencer par la selection des n-uplets de la relation WAGON dont le type est frigo, le résultat est stocké dans une relation temporaire WAGON'. Puis PROJECT_1 est exécuté en prenant le résultat pécédent WAGON', et son résultat est gardé dans une autre relation temporaire WAGON''. Cette relation WAGON'' n'est qu'un opérande de JOIN_2. Pour obtenir l'autre opérande, il faut interpréter la sous-arborescence dont la racine est PROJECT_3. Pour l'opérateur JOIN_1, nous avons déjà un opérande prêt, la relation TRAIN. SELECT_2 et PROJECT_2 sont exécutés de la même

façon que SELECT_1 et PROJECT_1 respectivement. Nouv avons donc deux relations temporaires TRAFIC' et TRAFIC'' comme leur résultat. Maintenant JOIN_1 peut lancer son exécution avec deux opérandes prêts TRAIN et TRAFIC'' et une relation temporaire TRAIN' sera produite. Après l'exécution de PROJECT_3 sur TRAIN', l'autre opérande TRAIN'' de JOIN_2 est obtenu. Enfin la jointure entre les deux relations temporaires WAGON'' et TRAIN'' sur le prédicat <NO-WAGON = NO-WAGON> nous donne le résultat de la requête.

Ceci est l'idée fondamentale du mode global, mais la plupart des interpréteur adoptent un façon, telle que l'on garde le résultat intermédiaires pour une seule opération de jointure dans le cas de fgure 4.1 <STO 76>, par exemple en éliminant WAGON', TRAFIC' et TRAIN'. Cette méthode demande des accès à la base pour les relations temporaires. Cela peut dégrader les performances de l'interpréteur si les relations temporaires à traiter sont grandes. De plus on n'obtient le résultat d'une requête qu'à la fin. Cela n'est pas souhaitable pour l'utilisateur d'un système interactif. En revanche, les liens entre les opérateurs sont simples à gérer et nous pouvons obtenir les statistiques exactes de résultats intermédiaires qui concerne la décision d'une méthode d'interpétation.

4.2.2 Mode pipe-line

On permet à tout opérateur de commencer son exécution aussi tôt que son opérande possède un nombre suffisant de n-uplet

<SMI 75>. La condition nécessaire au commencement d'un opérateur est différent selon les caractéristiques des opérateurs relationnels et nous les distinguons de ce point de vue <PAI 78>.

- Certains opérateurs peuvent commencer leur exécution dès la première arrivée d'un n-uplet venant de leur opérande, soit un sous-arbre, soit une relation. Ce type d'opérateur sera bien adapté à ce mode, par exemple PROJECT, SELECT et UNIQUE.
- L'exécution de certains opérateurs peut commencer avec une seule relation complète. Il s'agit des opérateurs binaires UNION, DIFFERENCE, PRODUIT CARTESIEN et JOINTURE. Dans le cas où aucun de ses opérandes n'est une relation complète, il faut attendre que l'une des deux le soit.
- Le dernier type d'opérateurs ne peut produire ses résultats qu'après la réception de son(ses) opérande(s) complét(s), par exemple DIVISION et ORDER BY.

Les opérateurs du premier type sont tout à fait conforme au mode pipe-line, même pour les opérateurs du second type, on peut l'adopter en commençant leurs opérations dès que une des deux complète. Mais les opérateurs du dernier type nous obligent à attendre son(ses) opérandes(s) complét(s).

Nous pouvons intuitivement reconnaitre que ce mode convient à l'exécution de requête dans un environnement de multiprocesseurs. Mais il pourrait être adapté aux systèmes n'ayant qu'un processeur au profit de la diminution des relations temporaires à garder et de la célérité du système interactif.

Nous expliquons la façon dont MICROBE l'utilise, en prenant l'arborescence de la figure 4.1 <LEE 81>. Comme l'opérateur JOIN_1 a la relation TRAIN en tant qu'opérande, cette jointure peut commencer à s'exécuter en recevant un n-uplet qui satifait la condition <JOUR = "830831"> et qui est ensuite projeté sur l'attribut NO-TRAIN. D'autre part, l'opérateur JOIN_2 doit avoir une relation temporaire en recevant le résultat de soit l'opérateur PROJECT_1 soit PROJECT_2. Si la relation temporaire stocke le résultat de PROJECT_1, nous pouvons obtenir le résultat de l'arborescence n-uplet par n-uplet en passant SELECT_2, PROJECT_2, JOIN_1, PROJECT_3 et JOIN_2 après l'accès à la relation TRAFIC. Enfin il n'y a qu'une relation temporaire à garder pour l'opérateur JOIN_1 pendant l'interprétation.

Cependent ce mode pipe-line a un inconvénient à savoir qu'il est nécessaire de conserver les catalogues pour les opérateurs qui modifient l'ensemble des attributs de leur résultat par rapport à celle de leur opérande, pendant toute la durée de l'interprétation. Ceci se passe pour les opérateurs de JOINTURE et de PRODUIT CARTESIEN qui les agrandissent, pour les opérateurs de PROJECTION et de DIVISION qui les réduisent.

4.3 Une approche d'interprétation améliorée

La manière optimale d'exécuter les opérateurs notament SELECT et JOIN a été présentée dans le chapitre 3. Comme l'interpréteur de MICROBE a été réalisé en mode pipe-line, nous nous proposons de voir comment notre amélioration peut être apportée quant à la façon d'interpréter une arborescence donnée. Cette tentative part du mode pipe-line en raison de l'extension future vers un systéme réparti.

Dans l'interpréteur, le passage des résultats entre les prcédures qui assurent l'exécution d'un opérateur relationnel est un n-uplet. En conséquence, nous avons réservé une longueur maximum (508 octets) pour une variable de n-uplet. Mais la pluspart des n-uplets ne l'utilisent pas (en particulier les résultats intermédiaires pendant l'interpétation). Nous y appliquons une idée de compression des n-uplets dans une page qui n'a que quatre octets de différence par rapport à la taille d'une variable n-uplet dans MICROBE, Cela vient de la technique des machine base de données pour optimiser la circulation de données <BOR 80>. Cependant, nous n'employons pas la compression à chaque passage, car cela demende un tampon supplémentaire entre les passage c'est-à-dire dans les procédures des opérateurs relationnels.

Le principe adopté par l'interpréteur est le suivant; les opérateurs des feuilles de l'arborescence font d'abord la compression des n-uplets pour leurs résultats. Les autres

opérateurs, bien que le taux d'occupation de la page diminue, soit par le nombre de n-uplets soit par la taille d'un n-uplet, continuent de traiter les n-uplets de la page reçue, sauf les opérateurs qui affectent les n-uplets de leur relation temporaire à ceux de leur résultat, par exemple la JOINTURE, la DIVISION et l'ORDER BY. Ils se font comme les opérateurs feuilles.

Expliquons-le plus concrètement en prenant les arborescence des requêtes Q2 et Q5 du chapitre 3; Nous pouvons voir facilement un seul opérateur SELECT pour la Q2. Donc l'opérateur envoie une page dans la quelle les n-uplets résultat sont en plus grand nombre possible. Mais pour la Q5, l'arborescence consiste en trois opérateurs de haut en bas, JOIN, PROJECT et SELECT. Dans ce cas, l'opérateur SELECT remplit une page de n-uplet résultat. Mais l'opérateur PROJECT ne fait pas la compression. Il s'exécute seulement avec les n-uplets de la page reçue. Il y aura donc des places libres, car l'opération réduit la longueur du n-uplet de résultat. Puis l'opérateur JOIN s'exécute en remplissant une page de son résultat comme nous l'avons décrit.

Nous constatons intuitivement que le parcours de l'arborescence peut être réduit. Cela nous mène directement non seulement à moindre exécution de la procédure correspondant à un opérateur, mais aussi à moin d'accès aux catalogues de la base tout au long de l'interprétation. Le chargement d'une procédure correspondant à un opérateur en mémoire centrale peut encore diminuer, surtout dans le cas de MICROBE, car le module est constitué en "Overlay".

Nous comparons approximativement deux techniques:

- passage des n-uplets un par un
- passage des pages une par une

sur les requêtes Q2 et Q5 dans les environnement de base de données décrit dans les figures 3.3 et 3.11.

requête	opérateur	nombre estimé de n-uplets du résultat	nombre de passage	
			n-uplet un par un	page une par une
Q2	SELECT	2000x1/3 (voir fig 3.4)	677	$\lceil 677/\lceil 508/30 \rceil \rceil + 1$
Q5	SELECT	6000x1/600	10	$\lceil 10/\lceil 508/6 \rceil \rceil + 1$
	PROJECT	10	10	1
	JOIN	10x20000 x1/20000	10	$\lceil 10/\lceil 508/30 \rceil \rceil + 1$

Figure 4.2 Comparaisons du nombre de passages des opérateurs

Comme la figure 4.2 le montre, surtout pour des n-uplets ayant une petite longueur, la réduction du nombre d'exécutions d'un opérateur est remarquable. En conclusion, cette approche même sans compression des résultats dans les opérateurs intermédiaires, nous apporte un réduction du nombre d'exécutions des procédures correspondant aux opérateurs qui se trouvent dans l'arborescence d'une requête. En outre, elle est fortement dépendante de la longueur et du nombre des n-uplets dont la compression est faite.

CHAPITRE V

CONCLUSION

Nous avons commencé par une implantation d'une organisation de données, et les B-arbres servent à la construction des relations index avec une moindre modification de la structure de la base de données, comme nous l'avons présentée dans le chapitre 2. Elle présente non seulement de bonnes performances au niveau de la recherche des n-uplets ayant une clé donnée, mais permet aussi l'accès ordonné selon les attributs désirés. Du point de vue des contrôles sémantiques du SGBD, elle fournit une facilité d'exmainer l'unicité des clés.

Au chapitre 3, nous avons d'abord montré les divers chemins d'accès possibles pour l'exécution des requête grâce à l'existence des relations index. Puis nous avons présenté un choix d'un chemin optimal en fonction d'une requête donnée et d'un environnement de la base de données. Ceci nous donne des performances améliorées au point de vue structural. C'est-à-dire que l'amélioration des performances se fait par l'accès optimal à la base de données grâce à une nouvelle organisation de données. Plus correctement, l'optimisation d'une requête portant sur une seule relation correspondant à l'opérateur SELECT est effectué par le balayage de la relation dans MIMER. Et l'opérateur de JOINTURE, traité fréquement comme un produit de deux relations, choisit une méthode optimale en considérant comme l'optimisation d'une requête portant sur plusieurs relations et en laissant de côté l'ordre d'exécution des opérateurs. Chaque choix d'un chemin d'accès est justifié par l'estimation de son coût.

Enfin, une approche, telle que la compression de n-uplets dans une page de taille fixe sans tampons supplémentaires au cours de l'interprétation d'une arborescence, mène à l'optimisation de l'interpéteur en mode pipe-line.

Pour conclure le travail mené tout au long de cette présentation, nous pouvons voir une amélioration des performances dans MICROBE grâce à une nouvelle organisation des données et à l'interprétation en mode pipe-line avec compression. Cela contribue à une nouvelle version de MICROBE. Cependent, nous n'avons pas d'évaluation complète du coût d'une requête donnée, car la mesure sur un système d'exploitation est difficile, en plus avec l'utilisation de l'overlay. Bien que nous n'ayons que partiellement montré les données quantitatives d'évaluation du coût, on peut facilement voir une amélioration. En plus les sessions conversationnelles de la nouvelle version nous le manifestent clairement.

Maintenant MICROBE poursuit une bonne route vers de nouvelles applications. Grâce à son dynamisme pour la création d'attributs et sa portabilité, Il est utilisé comme une interface relationnelle expérimentale du projet TIGRE (protype de SGBD de troisième génération) et a été, pour cela, transporté sur MULTICS en deux mois. Il devrait l'être prochainement sur la SM90 (UNIX).

Nous arrivons finalement à la conclusion que notre travail prouve une fois de plus la nécessité d'optimisation d'un SGBD relationnel même, s'il est petit, et nous a amené à une amélioration des performances de MICROBE.

ANNEXES

&

BIBLIOGRAPHIES

Annexe 1 Cadre fonctionnel pour MICROBE

ARCHITECTURE DU SGBD/DBMS ARCHITECTURE		CONCEPT DU SCHEMA RELATIONNEL/RELATIONAL SCHEMA CONCEPTS	LANGAGE DE DEFINITIONS DES DONNEES/DATA DEFINITION LANGUAGE							LANGAGES DE MANIPULATION DES DONNEES/DATA MANIPULATIONS LANGUAGE		UID INTERNE/DBL AT THE PHYSICAL LEVEL	
Niveau externe /external level	N	Attribut/ attribute	opération / type	Définition	Suppression/ insertion/ modification	MAJ clé primaire	Initialisation/ini-tialization	Lecture/ retrieval	LANGAGE DE REQUETE / QUERY LANGUAGE		Projection	Y
Niveau conceptuel /conceptual level	Y	Relation / relation	BB / DB	Y	Y	/	/	/	Fermeture/ Closure	Y	Sélection	Y
Niveau interne / internal level	Y	Clé primaire / primary key	Relation de base / base relation	Y	Y	Y	N	Y	Complétude/ Completeness	Y	Join	Y
		Domaine /domain	Domaine / Domain	N	N	N	N	N	Partitionnement/ partitioning (GROUP-BY)	Y	Mise à jour de tuple/ tuple update	Y
		Domaine primaire clé étrangère/ primary domain/ foreign key	Attribut/ Attribute	Y	Y	Y	/	Y	opérateurs statistiques/ functional operateurs	N	Cré./Sup.Relations temp./ crea.deletion temporary relations	Y
		contrainte/ constraint	Relation dérivée (vue)/derived relation (view	Y	Y	Y	Y	Y	TRI/SORT	Y		
		valeur nulle / null value	Contrainte Constraint	Y	Y	/	/	/	Terminaison explicite/ explicit commit	N		
			Chemin d'accès /access path (index)	Y	Y	/	Y	/	MAJ d'un ensemble de tuples/updates on a set of tuples	Y		
			Structure d'implantation / implementation structures	N	N	N	N	N	format de sortie/out put format	N		
									générateur de rapport/report generator	N		

Annexe 2 Type de variables définies pour la manipulation des relations index dans MIMER

```
CONST
 LNTEND =   14 ;    /* LONGUEUR DE TETE D'UN NOEUD DE B*-ARBRE EN OCTET *
 CPTDCL =    3 ;    /* LONGUEUR DE TETE D'UN NOEUD DE B*-ARBRE EN ADRESS
                       (LNTEND - LNEX) / (2*LNENT) */
 MXLNCL =  241 ;    /* LONGUEUR MAXIMUM DE NOEUD DE B*-ARBRE :
                       (LNMTPL - LNTEND - 3*4)/2 */
```

```
TYPE

 TUPINX  =                    /* UN TUPLE DE LA RELATION INDEX */
  RECORD
   EX       :  CHAR2   ;
   PTMARX   :  ADRESS  ;   /* NOM INTERNE DE LA RELATION INDEX */
   PTMARB   :  ADRESS  ;   /* NOM INTERNE DE LA RELATION DE BASE */
   CHRLIX   :  ADRESS  ;   /* CHAINAGE POUR LES RELATIONS INDEX DEFINIES
                              SUR LA MEME R.B. */
   CHAIX    :  ADRESS  ;   /* CHAINAGE POUR LES ATTRIBUTS-CLES */
   TYPERX   :  CHAR2   ;   /* TYPE DE RELATION INDEX */
   ORDREX   :  INTEGER ;   /* ORDRE DE B-ARBRE */
   LKIX     :  INTEGER ;   /* LONGUEUR D'UNE CLE */
   HAUTRX   :  INTEGER ;   /* PROFONDEUR DE B-ARBRE */
   NACCLX   :  INTEGER ;   /* NOMBRE DE ATTRIBUTS-CLES */
   NDCLEX   :  INTEGER ;   /* NOMBRE DE CLES DIFFERENTES */
   PTRACX   :  ADRESS  ;   /* ADRESSE DE LA RACINE */
   PTASCX   :  ADRESS  ;   /* POUR L'ACCES AUX CLES DANS L'ORDRE CROISSANT */
   PTDSCX   :  ADRESS      /* POUR L'ACCES AUX CLES DANS L'ORDRE DECROISSANT
  END ;

 TUPAIX  =                    /* UN TUPLE DE LA RELATION ATTRIBUTS-CLES */
  RECORD
   EX       :  CHAR2   ;
   CHAIAC   :  ADRESS  ;   /* CHAINAGE POUR LES ATTRIBUTS-CLES */
   PTRATT   :  ADRESS      /* NOM INTERNE DE L'ATTRIBUT-CLE */
  END ;
```

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

 ATTARG   =                    /* LISTE D'ARGUMENTS */
  RECORD
   INUTIN   :  INTEGER ;       /* INDICATEUR D'UTILISATION DU ATTRIBUT
                                  POUR BALAYAGE INDIRECT */
   PTRATC   :  ADRESS  ;       /* NOM INTERNE D'ATTRIBUT-ARGUMENT */
   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 ;

 ADRCLE   =                    /* POINTEUR D'UNE CLE DANS LA RELATION INVERSEE */
  RECORD
   PTRELIN :  ADRESS  ;        /* POINTEUR POUR PARCOURS DE LA RELATION INVERSEE
                                  (POINTEUR VERS UN NOEUD-FEUILLE DU B-ARBRE) */
   ELEMFE  :  INTEGER          /* POSITION D'UNE CLE DANS UN NOEUD-FEUILLE
                                  DU B-ARBRE */
  END ;

 TUPCAC   =                                                 /* UN TUPLE DE LA RELATION
                                                               CURSEURS-ACTIFS */
  RECORD
   EX       :  CHAR2   ;
   TYPBAY   :  CHAR2   ;                                    /* MODE DE BALAYAGE */
   NOMREL   :  ADRESS  ;                                    /* NOM INTERNE DE LA RELATION
                                                               BALAYEE. */
   VALCUR   :  ADRESS  ;                                    /* ADRESSE DU TUPLE A ACCEDER */
   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
                                                               INDIRECDES */
   LNGND    :  INTEGER ;                                    /* LONGUEUR D'UN NOEUD
                                                               DE B-ARBRE */
   STRCLE   :  ADRCLE  ;                                    /* LA CLE DEBUT DU B.I. */
   CURCLE   :  ADRCLE  ;                                    /* LA CLE ACTUELLE DU B.I. */
   FINCLE   :  ADRCLE  ;                                    /* LA CLE FINALE DU B.I. */
   INDIR    :  CHAR2   ;                                    /* MARQUE D'INDIRECTION
                                                               POUR L'ATTRIBUT ARG */
   ARG      :  ARGMT   ;                                    /* ARGUMENT DE BALAYAGE */
 END ;
```

```
NMATCL    =       /* NOMS DE ATTRIBUT-CLE */
 RECORD
  NXATCL   :  EXNAME   ;   /* NOM EXTERNE */
  NIATCL   :  ADRESS       /* NOM INTERNE */
 END ;

DESCRX =                                          /* UTILISE PAR LE PRIMITIVE
                                                    D'OBTENTION DE CARACTERISTIQU
                                                    DES INDEX */
 RECORD
  NMEXTX   :  EXNAME   ;                          /* NOM INTERNE DE R.I. */
  NMINTX   :  ADRESS   ;                          /* NOM INTERNE DE R.I. */
  TYRX     :  CHAR2    ;                          /* TYPE DE R.I. */
  ORRX     :  INTEGER  ;                          /* ORDRE DE B-ARBRE */
  LKRX     :  INTEGER  ;                          /* LONGUEUR D'UNE CLE */
  HARX     :  INTEGER  ;                          /* PROFONDEUR DE B-ARBRE */
  NACRX    :  INTEGER  ;                          /* NOMBRE DE ATTRIBUTS-CLES */
  NDCRX    :  INTEGER  ;                          /* NOMBRE DE DIFFRENTES CLES */
  LNMTCL   :  ARRAY°1..MNBATC§ OF NMATCL          /* LISTE D'ATTRIBUTS-CLES */
 END ;

TABINX    =    ARRAY°1..MNBINX§ OF DESCRX ;

LSTATC  =                                         /* LISTE D'ADRESSES */
 RECORD
   NLSTAT   :  INTEGER  ;                         /* NOMBRE D'ADRESSES */
   LSTADR   :  ARRAY °1..MNBATT§ OF ADRESS        /* ADRESSES */
END ;


ALOTY =                  /* DESCRIPTION D'UN ATTRIBUT-CLE */
 RECORD
   ATLON    : INTEGER ;  /* LONGUEUR */
   ATTYP    : CHAR2   ;  /* TYPE */
 END ;

CATIND =
 RECORD
   PTRLRX   :  ADRESS   ;                         /* NOM INTERNE DE R.I. */
   PTRLRB   :  ADRESS   ;                         /* NOM INTERNE DE R.B. */
   ORDRE    :  INTEGER  ;                         /* ORDRE DE B-ARBRE */
   LKI      :  INTEGER  ;                         /* LONGUEUR D'UNE CLE */
   HAUT     :  INTEGER  ;                         /* PROFONDEUR DE B-ARBRE */
   NACCLE   :  INTEGER  ;                         /* NB. DE ATTRIBUTS-CLES */
   NDCLE    :  INTEGER  ;                         /* NB. DE DIFFERENTES CLES */
   UNIND    :  CHAR2    ;                         /* UNICITE DES CLES */
   PVRAC    :  ADRESS   ;                         /* POINTEUR VERS LA RACINE */
   PPANF    :  ADRESS   ;                         /* POINTEUR POUR L'ACCES
                                                     CROISSANT */
   PPDNF    :  ADRESS   ;                         /* POINTEUR POUR L'ACCES
                                                     DESCROISSANT */
  ATRILOT :  ARRAY °1..MNBATC§ OF ALOTY           /* DESCRIP. DES ATTRIBUTS-CLES
END ;
```

BIBLIOGRAPHIES

<AHO 79> A.V.AHO, C.BERRI, J.D.ULLMAN
"The theory of Joins in Relational Databases"
ACM Transactions on databases systems vol.4, no.3, Sept 1979

<ANB 78> N.ANDERSON, M.BURKHARD
"MINISEQUEL:Relational Data Management System"
Database:Improving usability and responsiveness
Univ. of Califonria, San Diego, AoGt 1978

<AST 76> M.ASTRAHAN et Al
"SYSTEM-R: Relational Approach to Database Management"
ACM Transaction on database systems, vol.1, no.2, June 1976

<BAY 72> R.BAYER, E.MACCREIGHT
"Organisation and Maintenance of large ordered index"
Acta Informatica no.1, 1972

<BAY 76> R.BAYER, M.SCHKOLNICK
"Concurrency of Opertaion on B-trees"
IBM Research Laboratory, RJ1791, Mai 1976

<BLA 77> M.W.BLASGEN, K.P.ESWARAN
"Storage and Access in Relational Database"
IBM System Journal vol.16, no.4, 1977

<CHA 82> T.CHAIX, J.C.GUILLAUME
"INGRID: Interface Graphique d'Interrogation de Base de Données"
Projet MICROBE, Laboratoire IMAG, Juin 1982

<BOR 80> A.BORAL, D.J.DEWITT
"Design Consideration for Data-Flow Database Machine"
Proceeding of ACM-SIGMOD, Conference of Management of Data, Los Angeles, 1980

<COD 70> E.F.CODD
"A relational model of data for large shared data banks"
Comm. ACM vol.13, no.6, Juin 1970

<COD 79> E.F.CODD
"Extending the database relational model to capture more meaning"
ACM Transaction on database systems vol.4, no.4, Dec 1979

<COH 79> D.COHER
"The ubiquitous B-tree"
Computing Surveys, vol.11, no.2, June 1979

<DAT 77> C.J.DATE
"An introduction to database system"
Addison-Wesley Publishing Company, 2nd edition, 1977

<DEL 80> C.DELOBEL
"Les Bases de Données"
USMG, Sept 1980

<DEL 82> C.DELOBEL, M.ADIBA
"Base de données et Systèmes relationnels"
DUNOD informatique, Nov 1982

<FAG 79> R.FAGIN, J.NIEVERGEU, N.PIPPENGER, R.STRONG
"Extensible Hashing-A fast access method for dynamic files"
ACM Transaction on database systems vol.4, no.3, Sep 1979

<FER 81a> F.FERNANDEZ
"MIcro MEmoire Relationnelle (MIMER)"
Rapport de Recherche no.233, Laboratoire IMAG, Jan 1981

<FER 81b> F.FERNANDEZ, M.SARJLIC
"Une implanataion des B*-arbre dans la mémoire relationnelle MIMER"
Congrès AFCET Gif-sur-Yvette, Nov 1981

<FER 81c> F.FERNANDEZ
"La Micro Mémoire Relationnelle (MIMER):Un outil pour la construction de SGBD relationnels projet MICROBE"
Thèse de 3e cycle USMG/INPG, Nov 1983

<FRT 80> L.FERRAT
"Traduction d'un langage de haut niveau en arborescences algébriques projet MICROBE"
Rapport de DEA, laboratoire IMAG, Sept 1980

<FRT 81> L.FERRAT
"Projet MICROBE: Guide utilisateur du langage MIQUEL"
Note technique, laboratoire IMAG, Jan 1981

<FRT 83> L.FERRAT
"Expression et Contrôle de l'intégrité sémantique dans les bases de données relationnelles projet MICROBE"
Thèse de 3e cycle USMG/INPG, Mai 1983

<HAL 76> P.A.V.HALL
"Optimisation of single expression in a relational database system"
IBM J. Res. Develop., Mai 1976

<HEL 78> G.HELD et Al
"B-trees Re-examined"
CACM, vol.21, no.2, Fev 1978

<KNU 74> D.E.KNUTH
"The art of computer programming"
vol.3, Addison-wesley Publishing Company

<KWO 82> Y.S.KWONG, D.WOOD
"A new method for concurrency in B-trees"
IEEE Transaction on software engineering vol.se 8, no.3, Mai 1982

<LAR 79> R.A.LOIRE, J.F.NLSSON
"An access specification langage for a relational database system"
IBM J. Res. Develop., vol.23, no.3, Mai 1979

<LEE 81> Y.J.LEE
"Définition et Réalisation d'un interpréteur de requête dans MICROBE"
Rapport de DEA, laboratoire IMAG, Sept 1981

<LEE 83> Y.J.LEE
"Algorithme de choix des chemins d'accès dans la micro mémoire relationnelle (MIMER) de MICROBE"
Journées sur la conception, l'implantation et l'utilisation de SGBD relationnels sur micro-ordinateurs, Toulouse, Fev 1983

<LIT 79> W.LITWIN
"Hachage Virtuel: Une nouvelle technique d'adressage de mémoires"
Thèse de doctorat d'état, Univ. Pierre et Marie CURIE, Paris, Mars 1979

<LIT 80> W.LITWIN
"Trie Hashing"
Rapport INRIA, Projet SIRIUS, Réf.MAP-1-013, Nov 1980

<MRS 79> R.MUDYMA, J.KORNATOWSKI, I.LADO
"Implementing micro-computer database management system"
Univ. of Toronto, Report no.MSSIA1, Mai 1979

<NAF 79> N.NAFFAH, V.QUINT
"Protocole de transport pour réseaux local"
Document KAYAK Ref 2.504, 1979

<NGU 77> G.T.NGUYEN
"URANUS: Une approche relationnelle à la coopération de base de données"
Thèse de 3e cycle USMG, Dec 1977

<NGU 81> G.T.NGUYEN, S.SERGEANT
"Distributed architecture and decentralized control for a local network database system"
ACM international Computing Symposium, London, Mars 1981

<NGU 82> G.T.NGUYEN, F.FERNANDEZ, L.FERRAT, Y.J.LEE
"MICROBE Manuel de référence"
Laboratoire IMAG, Jan 1982

<NGU 83> G.T.NGUYEN, Y.J.LEE, P.WINNINGER
"MICROBE: Un système de Gestion de Base de Données Relationnelles"
Journées sur la conception, l'implantation et l'utilisation de SGBD relationnels sur micro-ordinateurs, Toulouse, Fev 1983

<PAI 78> I.S.PAIK
"Définition et réalisation d'un ensemble d'opérateurs relationnels pour une machine POLYTHEME"
Rapport de DEA, laboratoire IMAG, Sept 1978

<PRA 82> S.PRAWIROSUSANTO
"Sécurité de fonctionnement et Reprise après panne d'une base de données relationnelles mise en oeuvre au système MICROBE"
Rapport de DEA, laboratoire IMAG, Juin 1982

<RSX 11> "Introduction to RSX-11M"
Order no. AA-2555c-tc RSX-11M, Version 3.1, DEC

<SEL 79> P.G.SELINGER et Al
"Acces path selection in a relational databse management system"
ACM-SIGMOD 1979

<SMI 75> D.SMITH, P.Y.CHANG
"Optimizing the performance of relational algebra data base interface"
CACM vol.18, no.10, Oct 1975

<STO 76> M.STONEBRAKER, E.WONG, P.KREPS
"The design and implementation of INGRES"
ACM Transaction on Database Systems vol.1, no.3, Sept 1976

<VAL 81> P.VALDURIEZ
"Un processeur de jointure et tri pour une machine basee de données"
Thèse de 3e cycle, Univ. Pierre et Marie CURIE, Paris 1981

<WED 74> H.WEDEKIND
"On the selection of access paths in a database system"
Database management North-Holland, Amsterdam, 1974

<WIN 82> P.WINNINGER
"Un optimiseur algébrique de requête dans MICROBE"
Rapport de DEA, laboratoire IMAG, Sept 1982

<WIR 76> N.WIRTH
"Algorithm + data structures = progarmme"
Prentice-Hall Inc., Englewood clifts, N.J., 1976

<WON 76> E.WONG, K.YOUSSEFI
"Decomposition-a strategy for query processing"
ACM Transaction on database systems vol.1, no.3, Sept 1976

Annexe 3 La syntaxe des primitives de MIMER pour la manipulation des relations index (implantation en PASCAL)

B8) LIRE-DESCRIPTEUR-RELATION-INDEX-EN-B-ARBRE ;

```
procedure LDA ( var IARI : adress;  var DESARB : catind;
 var CE : integer ) ;
```

B9) ECRIRE-DESCRIPTEUR-RELATION-INDEX-EN-B-ARBRE ;

```
procedure EDA ( var IARI : adress;  var DESARB : catind;
 var CE : integer ) ;
```

B10) EXTRAIRE-CLE ;

```
procedure EC ( var DESARB : catind;  var IE , IS : tuple;
 var CE : integer ) ;
```

B11) INSERER-CLE ;

```
procedure IC ( var IARI , AIAI : adress; var IAI : tuple;
 var CE : integer ) ;
```

B12) SUPPRIMER-CLE ;

```
procedure DC ( var IARI , AIAI : adress; var IAI : tuple;
 var CE : integer ) ;
```

B13) CHERCHER-CLE-DANS-B-ARBRE ;

```
procedure SC ( var DESARB : catind;  var AIDC : adress;
 AIDC : adress;  CLEINX , NIV1 , NIV2 : integer;
 var SPLIND , INDQUE : boolean;  var CE : integer ) ;
```

B14) INSERER-CLE-DANS-B-ARBRE ;

```
procedure ICA ( var DESARB : catind;  var TUP : tuple ;
 var AIAI , AIDF : adress;  var IDF : tuple;  NIVEAU ,
 CLEINX : integer;  var SPLIND , INDQUE : boolean;
 var CE : integer ) ;
```

B15) SUPPRIMER-CLE-DANS-UN-NOEUD-NON-FEUILLE ;

```
procedure SS ( var DESRAB : catind;  var AIAC : adress;
 AIDS : adress;  CLEINX , NIV1 , NIV2 : integer;
 var SPLIND , INDQUE : boolean;  var CE : integer ) ;
```

```
B16) RANGER-NOEUD-UNDERFLOW ;
     procedure UF ( var DESARB : catind;  ATA , ATU : adress;
      S , NIVEAU : integer;  var SPLIND , INDQUE : boolean;
      var CE : integer ) ;

B17) FIXER-VALEUR-CURSEUR ;
     procedure FIXCUR ( var TUPCRA : tuple;  var TCASE :
      integer;  var INTVAL : boolean ) ;

B18) CHOISIR-CHEMIN-D'ACCES ;
     procedure DCAM ( var TUPCRA : tuple;  var TCASE :
      integer;  var INTVAL : boolean ) ;

-------------------------------------------------------------

C26) CREER-RELATION-INDEX ;
     procedure CX ( var IR1 : adress;  RX : exname;  var IR2 :
      adress;  TYINX : char2;  var LSTATT : LSTATC;  var CE :
      integer ) ;

C27) OBTENIR-CARACTERISTIQUES-INDEX ;
     procedure OCX ( var IR1 : adress;  NBINX : integer; var
      TABX : tabinx;  var CE : integer ) ;

C29) INSERER-TUPLE ;
     procedure IT ( var IR1 : adress;  XTUPLE : tuple;  var CE
      : integer ) ;

C30) VALEUR-CURSEUR ;
     procedure VC ( var CURSEUR , VALEUR : adress;  var CE :
      integer ) ;

C31) AFFECTER-CURSEUR ;
     procedure AV ( var CURSEUR , VALEUR : adress;  var CE :
      integer ) ;
```

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de Messieurs

. C. DELOBEL, Professeur

. R. DEMOLOMBE, Ingénieur

Monsieur LEE Yoon Joon

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Fait à Grenoble, le 13 octobre 1983

Le Président de l'INP-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

Une approche pour optimiser les performanaces dans les SGBD relationnels : projet Microbe

1983-11-03

Le but de la thèse est d'ameliorer les performances du SGBD relationnel Microbe, sur les micro-ordinateurs LSI-11 du laboratoire IMAG et sous Multics. On fait une estimation du coût d'accès aux données traitées dans une requête afin de selectionner un chemin d'accès optimal. On introduit enfin une méthode d'execution optimisée des questions représentées sous forme d'une arborescence d'opérateurs relationnels : le mode "pipe-line" paginé

SUDOC-004890248 ; MI2S-tu1600

Claude Delobel

Gérard Veillon (Président)
Michel Adiba (Examinateur)
Claude Delobel (Examinateur)
R. Demolombe (Examinateur)
G.T. Nguyen (Examinateur)

base de données;systèmes relationnels;optimisation;chemin d'accès;estimation du coût:interprétation;opération relationnels

Universités : Université scientifique et médicale de Grenoble et Institut national polytechnique de Grenoble