Expression et contrôle de l'intégrité sémantique dans les bases de données relationnelles : projet MICROBE

Lounas Ferrat

THESE

présentée à

l' Université Scientifique et Médicale de Grenoble

et à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE

« Genie informatique »

par

Lounas FERRAT

EXPRESSION ET CONTROLE DE L'INTEGRITE SEMANTIQUE DANS LES BASES DE DONNEES RELATIONNELLES.

PROJET MICROBE

Thèse soutenue le 19 mai 1983 devant la commission d'examen:

C. DELOBEL	Président
M. ADIBA G. GARDARIN G.T. NGUYEN J.M. NICOLAS	Examinateurs

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

année scolaire 1980-1981

Président de l'Université : M. J.J. PAYAN

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

PROFESSEURS DE 1ère CLASSE

Mlle	AGNIUS DELORD Claudine	Biophysique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Clinique dermatologie
	AMBROISE THOMAS Pierre	Parasitologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	Physique nucléaire
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale A
	BEAUDOING André	Clinique pédiatrie et puériculture
	BELORISKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
Mme	BERIEL Hélène	Pharmacodynamie
M.	BERNARD Alain	Mathématiques pures
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques pures
	BEZES Henri	Clinique chirurgicale & traumatologie
	BILLET Jean	Géographie
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET EYMARD Joseph	Clinique Hépato-gastro-entérologie
Mme	BONNIER Jane-Marie	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHET Yves	Anatomie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie

.../...

MM.	BUTEL Jean	Orthopédie
	CABANEL Guy	Clinique rhumatologie et hydrologie
	CARLIER Georges	Biologie végétale
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHARACHON Robert	Clinique O.R.L.
	CHATEAU Robert	Clinique neurologique
	CHIBON Pierre	Biologie animale
	COEUR André	Chimie analytique et bromotologique
	COUDERC Pierre	Anatomie pathologique
	CRABBE Pierre	C.E.R.M.O.
	DAUMAS Max	Géographie
	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELOBEL Claude	M.I.A.G.
	DELORMAS Pierre	Pneumo-phtisiologique
	DENIS Bernard	Clinique cardiologique
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Electrochimie
	DODU Jacques	Mécanique appliquée IUT 1
	DOLIQUE Jean-Michel	Physique des plasmas
	DUCROS Pierre	Cristallographie
	FONTAINE Jean-Marc	Mathématiques pures
	GAGNAIRE Didier	Chimie physique
	GASTINEL Noël	Analyse numérique
	GAVEND Jean-Micnel	Pharmacologie
	GEINDRE Michel	Electro-radiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	JEANNIN Charles	Pharmacie galénique
	JOLY Jean-René	Mathématiques pures
	KAHANE André	Physique
	KAHANE Josette	Physique
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	LACAZE Albert	Hermodynamique
	LACHARME Jean	Biologie cellulaire
	LAJZEROWICZ Joseph	Physique

.../...

Mme	LAJZEROWICZ Jeannine	Physique
MM.	LATREILLE René	Chirurgie thoracique
	LATURAZE Jean	Biochimie pharmaceutiques
	LAURENT Pierre	Mathématiques appliquées
	LE NOC Pierre	Bactériologie virologie
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Jean-Marie	Sciences nucléaires
	LOUP Jean	Géographie
	LUU DUC Cuong	Chimie générale et minérale
	MALINAS Yves	Clinique obstétricale
Mlle	MARIOTTE Anne-Marie	Pharmacognostie
MM.	MAYNARD Roger	Physique du solide
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	NEGRE Robert	Mécanique IUT 1
	MOZIERES Philippe	Spectrométrie physique
	OMONT Alain	Astrophysique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY PEYROULA Jean-Claude	Physique
	PERRET Jean	Sémeiologie médicale (neurologie)
	PERRIER Guy	Géophysique
	PIERRARD Jean-Marie	Mécanique
	RACHAIL Michel	Clinique médicale B
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
Mme	RENAUDET Jacqueline	Bactériologie
M.	REVOL Michel	Urologie
Mme	RINAUDO Marguerite	Chimie CERMAV
MM.	DE ROUGEMONT Jacques	Neuro-chirurgie
	SARRAZIN Roger	Clinique chirurgicale B
Mme	SEIGLE MURANDI Françoise	Botanique et crytogamie
MM.	SENGEL Philippe	Biologie animale
	SIBILLE Robert	Construction mécanique IUT 1
	SOUTIF Michel	Physique
	TANCHE Maurice	Physiologie
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire

.../...

MM.	VAN CUTSEM Bernard	Mathématiques appliquées
	VAUQUOIS Bernard	Mathématiques appliquées
	VERAIN Alice	Pharmacie galénique
	VERAIN André	Biophysique
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS DE 2ème CLASSE

MM.	ARNAUD Yves	Chimie IUT 1
	AURIAULT Jean-Louis	Mécanique IUT 1
	BEGUIN Claude	Chimie organique
	BOITET Christian	Mathématiques appliquées
	BOUTHINON Michel	E.E.A. IUT 1
	BRUGEL Lucien	Energétique IUT 1
	BUISSON Roger	Physique IUT 1
	CASTAING Bernard	Physique
	CHARDON Michel	Géographie
	CHEHIKIAN Alain	E.E.A. IUT 1
	COHEN Henri	Mathématiques pures
	COHENADDAD Jean-Pierre	Physique
	COLIN DE VERDIERE Yves	Mathématiques pures
	CONTE René	Physique IUT 1
	CYROT Michel	Physique du solide
	DEPASSEL Roger	Mécanique des fluides
	DOUCE Roland	Physiologie végétale
	DUFRESNOY Alain	Mathématiques pures
	GASPARD François	Physique
	GAUTRON René	Chimie
	GIDON Maurice	Géologie
	GIGNOUX Claude	Sciences nucléaires
	GLENAT René	Chimie organique
	GOSSE Jean-Pierre	E.E.A. IUT 1
	GROS Yves	Physique IUT 1
	GUITTON Jacques	Chimie
	HACQUES Gérard	Mathématiques appliquées
	HERBIN Jacky	Géographie
	HICTER Pierre	Chimie
	IDELMAN Simon	Physiologie animale
	JOSELEAU Jean-Paul	Biochimie
	JULLIEN Pierre	Mathématiques appliquées
	KERCKOVE Claude	Géologie

.../...

MM.	KRAKOWIACK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique IUT 1
	KUPKA Yvon	Mathématiques pures
	LUNA Domingo	Mathématiques pures
	MACHE Régis	Physiologie végétale
	MARECHAL Jean	Mécanique
	MICHOULIER Jean	Physique IUT 1
Mme	MINIER Colette	Physique IUT 1
MM.	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique IUT 1
	OUDET Bruno	Mathématiques appliquées
	PEFFEN René	Métallurgie IUT 1
	PELMONT Jean	Biochimie
	PERRAUD Robert	Chimie IUT 1
	PERRIAUX Jean-Jacques	Géologie minéralogie
	PERRIN Claude	Sciences nucléaires
	PFISTER Jean-Claude	Physique du solide
	PIERRE Jean-Louis	Chimie organique
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	RICHARD Lucien	Biologie végétale
	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
Mme	SOUTIF Jeanne	Physique
MM.	STUTZ Pierre	Mécanique
	VIALON Pierre	Géologie
	VIDAL Michel	Chimie organique
	VIVIAN Robert	Géographie

CHARGES D'ENSEIGNEMENT PHARMACIE

MM.	ROCHAS Jacques	Hygiène et hydrologie
	DEMENGE Pierre	Pharmacodynamie

PROFESSEURS SANS CHAIRE (médecine)

M.	BARGE Michel	Neuro-chirurgie

.../...

MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie
	CHAMBAZ Edmond	Biochimie (hormonologie)
	CHAMPETIER Jean	Anatomie
	COLOMB Maurice	Biochimie
	COULOMB Max	Radiologie
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GROULADE Joseph	Biochimie A
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Gérontologie
	JALBERT Pierre	Histologie
	MAGNIN Robert	Hygiène
	PHELIP Xavier	Rhumatologie
	REYMOND Jean-Charles	Chirurgie générale
	STIEGLITZ Paul	Anesthésiologie
	VROUSOS Constantin	Radiothérapie

MAITRES DE CONFERENCES AGREGES (médecine)

MM.	BACHELOT Yvan	Endocrinologie
	BENABID Alim Louis	Médecine et chirurgie
	BERNARD Pierre	Gynécologie obstétrique
	CONTAMIN Charles	Chirurgie thoracique
	CORDONNIER Daniel	Néphrologie
	CROUZET Guy	Radiologie
	DEBRU Jean-Luc	Médecine interne
	DYON Jean-François	Chirurgie infantile
	FAURE Claude	Anatomie et organogènèse
	FAURE Gilbert	Urologie
	FLOYRAC Roger	Biophysique
	FOURNET Jacques	Hépato-gastro-entérologie
	GAUTIER Robert	Chirurgie générale
	GIRARDET Pierre	Anesthésiologie
	GUIDICELLI Henri	Chirurgie générale
	GUIGNIER Michel	Thérapeutique (réanimation)
	JUNIEN-LAVILLAUROY Claude	Clinique O.R.L.
	KOLODIE Lucien	Hématologie biologique
	MALLION Jean-Michel	Médecine du travail
	MASSOT Christian	Médecine interne
	MOUILLON Michel	Ophtalmologie

.../...

MM.	PARAMELLE Bernard	Pneumologie
	RACINET Claude	Gynécologie-Obstétrique
	RAMBAUD Pierre	Pédiatrie
	RAPHAEL Bernard	Stomatologie
	SCHAEFER René	Cancérologie
	SEIGNEURIN Jean-Marie	Bactériologie-virologie
	SOTTO Jean-Jacques	Hématologie
	STOEBNER Pierre	Anatomie-pathologique

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

Vice-Présidents : M. Georges LESPINARD
M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioéléctricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahïm	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KAMARINOS Georges	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges — Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

E.N.S.E.R.G.

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

E.N.S.I.E.G.

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

E.N.S.I.M.A.G.

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

Je tiens à remercier :

Monsieur Claude DELOBEL, Professeur à l'Université de Grenoble, qui est à l'origine de cette thèse, pour l'intérêt qu'il a porté à ce travail. Je le remercie aussi d'avoir bien voulu me faire l'honneur de présider le jury de cette thèse.

Monsieur Georges GARDARIN, Professeur à l'université de Paris VI et chef de projet à l'INRIA-SIRIUS, qui a bien voulu faire partie de ce jury.

Monsieur Michel ADIBA, Professeur à l'Université de Grenoble, pour son amabilité, sa totale disposition à mon égard, ses critiques constructives et sa participation au jury.

Monsieur Jean-Marie NICOLAS, Ingénieur de recherche à l'ONERA-CERT, qui a accepté de juger cette thèse et dont les critiques m'ont été précieuses.

Monsieur Gia Toan NGUYEN, Ingénieur INRIA à l'IMAG et animateur de l'équipe MICROBE, pour les nombreuses discussions que j'ai eues avec lui et sa participation au jury.

Je suis également reconnaissant :

à Jean-Pierre GIRAUDIN, qui a eu la patience de lire et de critiquer ce manuscrit. Je le remercie aussi d'avoir guidé mes premiers pas d'enseignant à l'IUT de statistiques.

à Gilles BOGO, Ingénieur CII-HB pour les nombreuses discussions que j'ai eues avec lui sur le sujet.

à tous les membres de l'équipe MICROBE pour leur amical soutien, en particulier Fernando FERNANDEZ et Yoon-Joon LEE.

et enfin, au service de reprographie, en particulier Monsieur IGLESIAS, pour la réalisation matérielle de cet ouvrage.

Je salue aussi toutes les personnes de l'étage, qu'ils trouvent ici l'expression de ma vive sympathie.

Je ne saurais oublier,

tous ceux qui de près ou de loin, m'ont soutenu moralement pendant toute la préparation de cette thèse.

En souvenir de mon père,

à toute ma famille.

SOMMAIRE

1.0 INTRODUCTION

1.1 L'INTEGRITE DANS LES BASES DE DONNEES

L'intégrité des données est une des fonctions essentielles des SGBD. Les SGBD basés sur les modèles réseau et hiérarchique proposent des solutions limitées et très hétérogènes. En 1970 apparait le modèle relationnel <COD70> qui apporte des améliorations certaines relatives à ce problème. Quelques prototypes fondés sur ce modèle ont été réalisés <SYS76> et <ING76>, l'intégrité est traitée de façon plus ou moins concise. L'intégrité des données d'une base recouvrent plusieurs notions :

Contrôle de concurrence : Ce contrôle de l'accès a pour but de coordonner les actions des usagers d'une base de données, pour partager l'information, de façon que celle-ci reste valide.

Protection des données : C'est la possibilité de ne donner l'accès aux informations qu'à des personnes autorisées.

Sécurité de la base : Elle consiste à maintenir l'existence et la cohérence de la base en cas d'incidents techniques (erreur logicielle, panne machine, ...). Nous appelons les solutions apportées à ce problème "les mécanismes de reprise", c'est à dire l'ensemble des étapes effectuées pour revenir à un état cohérent.

Intégrité sémantique : On s'intéresse ici à la qualité de l'information, c'est à dire satisfaire à tout moment de la vie de la base les contraintes d'intégrité sémantiques (CIS) des données. En d'autres termes c'est la possibilité d'assurer que les informations stockées dans la base soient cohérentes par rapport à la "signification" qu'elles ont.

Dans le domaine plusieurs termes sont utilisés pour nommer les contraintes d'intégrité sémantiques (CIS). On trouve les termes "assertions", "prédicats", "règles de cohérence"... Pour notre part, nous utilisons indifféremment les vocables "CIS" ou "assertion" ou simplement "contrainte".

Pour notre étude, nous partons du modèle relationnel. Plusieurs publications traitent de ce problème (<STO75>, <ESW75>, <ESW76>, <McL76>, <BAD79>, <BLA81> et <NIC79>), mais très peu de SGBD existants ont implanté une méthode opérationnelle assurant la cohérence sémantique des données : le système INGRES <ING76> propose une méthode opérationnelle décrite dans <STO76>.

Nous avons élaboré un sous-système ISIS (Implantation d'un Sous-système d'Intégrité Sémantique) chargé des contrôles de cohérence sémantique dans l'environnement du SGBD MICROBE <FER80> : prototype de SGBD relationnel sur micro-ordinateur.

1.2 ENVIRONNEMENT DE L'ETUDE : LE PROJET MICROBE

En Septembre 1979 a été initialisé le projet MICROBE. Il consiste en la conception et la réalisation d'un SGBD relationnel, réparti sur un réseau local du type DANUBE (projet KAYAK <NAF79>).

L'utilisateur dispose d'une interface relationnelle de haut niveau MIQUEL <FRT80> qui lui permet de manipuler la base, les structures et les données.

Les processeurs étant des micro-ordinateurs LSI 11/23 de DEC avec une mémoire centrale de 128K octets (découpée en 2 partitions adressables de 64K octets maximum chacune). Le système d'exploitation est RSX-11M. La version centralisée de MICROBE tourne depuis Juin 81. L'environnement matériel et logiciel a imposé des contraintes d'implantation souvent draconniennes qui ont limitées les extensions du système, en particulier la version répartie.

Ce projet sera présenté en détails dans le chapitre 7.

1.3 POURQUOI UN SOUS-SYSTEME D'INTEGRITE ?

La nécessité d'un sous-système d'intégrité sémantique dans un SGBD Relationnel peut se justifier de plusieurs façons :

- Par rapport aux autres applications, les SGBD manipulent un volume de données très important.

- De plus, on peut noter que plusieurs utilisateurs ont accès simultanément ou non à ces mêmes données, d'où le risque plus grand d'avoir des données incohérentes.

- Et enfin, ces mêmes données sont en général sémantiquement plus riches.

1.4 PRESENTATION DE LA THESE : LE SOUS-SYSTEME ISIS

Cette thèse présente le sous-système ISIS (Implantation d'un Sous-système d'Intégrité Sémantique), méthode d'expression et de contrôle de l'intégrité sémantique dans une base de données relationnelles.

On s'intéresse d'abord à l'expression des contraintes, pour cela le langage MIQUEL a été étendu. Il prend en compte aussi bien les CIS mono-relationnelles que les CIS multi-relationnelles.

Pour bien comprendre la sémantique des requêtes et des contraintes, un rappel est donné. Nous avons déduit un ensemble de règles de traduction d'une expression MIQUEL en une expression algébrique.

Ensuite on présente une méthode de traduction de la requête ou de l'assertion en arborescence algébrique ‹FRT80›. Plusieurs avantages justifiant ce choix peuvent être cités, d'abord ces arborescences offrent des facilités de manipulation, d'évaluation, et d'optimisation, ensuite un même module pour l'évaluation des requêtes et des contraintes est utilisé moyennant quelques modifications dans le cas d'une contrainte. De plus dans le contexte réparti la même méthode de décomposition des arborescences peut être appliquée. Une preuve de l'équivalence sémantique des deux expressions est donnée.

Une fois la contrainte traduite en arborescence algébrique celle-ci est éventuellement restructurée ‹WIN82›, et ensuite stockée dans le catalogue des contraintes. Ce catalogue est intégré dans la micro-mémoire relationnelle ‹FER81b›, pour cela des primitives de manipulation des contraintes (insérer, détruire, obtenir et modifier une contrainte) ont été définies et réalisées.

Lors de l'éxécution d'une requête de mise-à-jour des données peuvent devenir incohérentes. Pour éviter cela une méthode de validation des opérations de mise-à-jour est présentée dans un contexte mono-relationnel. Lors de l'intégration d'un sous-système d'intégrité sémantique dans un SGBD il en résulte une dégradation de la performance des requêtes ‹BPO79› dûe à la validation des opérations de mise à jour, ainsi avons nous effectué une optimisation de la méthode au niveau de l'évaluation des CIS.

Nous proposons une synthèse du projet MICROBE. Ce projet est bâti en architecture modulaire, nous exposons les fonctions et la structure de chacun de ses modules.

Enfin, nous exposons la réalisation du système ISIS que nous avons développé en PASCAL OMSI.

2.0 RAPPELS THEORIQUES PRELIMINAIRES

Les différents paragraphes qui vont suivre définissent le cadre théorique dans lequel s'intègre cette thèse. On rappelle les définitions du modèle relationnel, des opérateurs de l'algèbre relationnelle et on décrit le schéma relationnel utilisé dans les différents exemples qui illustrent cette thèse.

2.1 LE MODELE RELATIONNEL

C'est à partir de 1970, qu'est introduit le modèle relationnel ‹COD70› ‹ADE82› : il marque une étape décisive dans l'évolution des bases de données. Ce modèle, par comparaison aux modèles hiérarchique et réseau, est muni d'un formalisme mathématique permettant de définir les objets et les opérations.

Dans ce modèle, une base de données est décrite par un schéma noté

$$S = ((R1,R2,...Rn), \Sigma)$$

qui consiste en un ensemble fini de relations (RI) et un ensemble Σ de propriétés inter-relations.

Chaque relation (ou intention) est composée d'un identificateur R, d'un ensemble fini d'attributs (A1,A2,...An) et de Σr : un ensemble de propriétés intra-relation. La relation est alors notée

$$R = \langle (A1,A2,...An), \Sigma r \rangle.$$

Si D1,D2,...Dn représentent des ensembles de valeurs (appelés domaines) respectivement de A1, A2 ... et An, l'extension de R (ou simplement la relation R) est définie comme un sous-ensemble du produit cartésien de ces domaines,

$$R(A1,A2,... An) \subseteq D1XD2 ...XDn.$$

Pour représenter au mieux la réalité modélisée, les deux ensembles Σ et Σr de règles d'intégrité complètent le schéma de la base. Ces règles permettent de réduire l'ensemble des états pris par la base.

Chaque relation est généralement représentée sous forme d'un tableau à n colonnes, dont chacune est appelée attribut. Chaque ligne de R, appelée tuple, est identifiée de manière unique par une clé primaire qui correspond à la valeur d'un (ou plusieurs) de ses attributs.

2.2 ALGEBRE RELATIONNELLE

Pour manipuler les relations définies dans le modèle relationnel, un ensemble d'opérateurs ont été définis <COD72>. Il existe deux types d'opérateurs, les opérateurs binaires et les opérateurs unaires. Les premiers sont issus de la théorie des ensembles et ont deux opérandes (deux relations) en entrée alors que les seconds n'en ont qu'un opérande (une relation).

Nous ne présenterons dans l'annexe 1 que les opérateurs utilisés dans les chapitres qui suivent, c'est-à-dire :

Opérateurs binaires :
θ-produit (θ-join), produit cartésien, intersection, union et différence.

Opérateurs unaires :
projection, sélection et anti-projection

Nous pouvons engendrer à partir du langage MIQUEL (annexe 2) les opérateurs produit cartésien, sélection, projection, union et différence, nous dirons alors que notre langage est "algébrique complet" <DEL80>, c'est à dire qu'on dispose de la puissance du calcul des prédicats du premier ordre.

2.3 SCHEMA RELATIONNEL UTILISE

Dans ce document, on utilise les deux base de données : la gestion des étudiants et des employés.

La première est composée de 3 relations :

La relation ETUDIANT qui décrit complètement un étudiant, son numéro de sécurité sociale (noss), son nom, son prénom, son adresse et sa date de naissance (datnais). La clé est noss.

La relation unité de valeur (UNITVAL) décrit les unités de valeur auxquelles les étudiants peuvent s'inscrire, on donne le numéro d'unité de valeur (nouv), le titre et le professeur responsable. La clé est nouv.

Et enfin la relation inscription (INSCRIT) décrit pour chaque étudiant les unités de valeurs auxquelles il est inscrit et la date d'inscription (datins). La clé de la relation est le couple (noss, nouv).

La seconde base est composée de 2 relations :

La relation EMPLOYE décrit pour chaque employé son nom, son prénom, son salaire et son numéro de département.

La seconde relation DEPTMENT (département) est composée des attributs suivants : le numéro de département (nodept), le libellé du département (libdept), le chef du département, le nombre d'employés (nbemp) et le budget.

Schéma relationnel de la base "gestion des étudiants" :

- ETUDIANT (noss, nom, prénom, adresse, datnais).
- INSCRIT (noss, nouv, datins, uer).
- UNITVAL (nouv, titre, responsable).

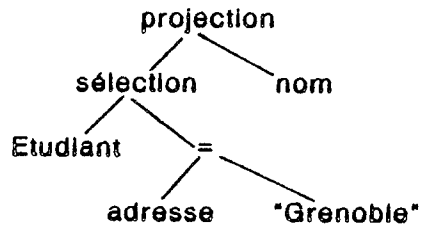
Schéma relationnel de la base "gestion des employés" :

- EMPLOYE (nom, prénom, sal, nodept).
- DEPTMENT (nodept, libdept, chef, nbemp, budget).

2.4 ARBORESCENCE ALGEBRIQUE

L'information que nous voulons obtenir d'une base de données peut s'exprimer sous forme d'une relation obtenue par applications successives d'opérateurs relationnels dont les opérandes sont les relations (intermédiaires ou non) de cette même base. Ce processus peut être représenté par une arborescence dite "algébrique" où les noeuds sont les opérateurs algébriques, des identificateurs de relations ou d'attributs, des opérateurs de liaison (and, or, =, >=, ...), des constantes, etc ...

Nous allons illustrer ceci par un exemple, supposons qu'on s'intéresse aux noms des étudiants habitant Grenoble. Cette requête peut être représentée par l'arborescence algébrique suivante :


projection
sélection
nom
Etudiant
=
adresse
"Grenoble"

3.0 DEFINITION, CLASSIFICATION ET EXPRESSION DES CONTRAINTES

3.1 INTRODUCTION

Les nombreuses études qui ont été faites sur l'intégrité sémantique restent très hétérogènes, sans doute celà est dû au fait que l'intégrité touche à tous les niveaux des bases de données : le niveau externe, le niveau conceptuel et le niveau interne <ANS75>. Nous allons d'abord étudier le problème au niveau externe, c'est à dire définir la notion d'intégrité sémantique, faire une classification des contraintes d'intégrité sémantiques (CIS) et voir leur expression dans un langage.

3.2 INTEGRITE ET CONTRAINTE SEMANTIQUES : DEFINITIONS

3.2.1 DEFINITIONS -

Plusieurs définitions de l'intégrité sémantique ont été données dans la littérature, nous en avons retenu deux qui nous semblent très significatives.

La première est celle de Florentin <FLO74> : "l'intégrité sémantique est la cohérence assurée entre le monde extérieur représenté par le modèle et le contenu de la base".

La seconde est celle d'Eswaran <ESW75> : "l'intégrité sémantique traite de la prévention des erreurs sémantiques, effectuées par les utilisateurs, dûes à leur négligence ou méconnaissance des données".

Dans les définitions du modèle relationnel exposées dans le chapitre 2, les deux ensembles de règles Σ et Σr introduits pour avoir une image fidèle du monde réel modélisé sont des contraintes d'intégrité sémantiques (CIS).

3.2.2 CAS PARTICULIER : L'INTEGRITE REFERENTIELLE -

Dans le modèle relationnel, chaque relation a une clé. Si elle est unique alors elle est dite primaire sinon secondaire. L'intégrité référentielle est un cas particulier des CIS qui s'attachent aux clés des relations.

Considérons notre base-exemple déja définie où noss est la clé primaire de la relation ETUDIANT, nouv est la clé primaire de la

relation UNITVAL et le couple d'attributs (noss , nouv) représente la clé composée primaire de la relation INSCRIT. Dans cette structure noss et nouv sont les deux seuls attributs primaires.

L'intégrité référentielle ‹DAT81› s'intéresse aux deux notions suivantes :

La première concerne l'existence des valeurs de la clé dans deux relations différentes. Pour chaque valeur de Noss dans la relation INSCRIT, il doit exister la même valeur pour Noss dans la relation ETUDIANT. Sans celà, les valeurs de Noss citées n'auraient pas de sens puisque l'étudiant n'existe pas. la contrainte associée peut s'exprimer de la manière suivante en langage MIQUEL (annexe 2) issu du langage SEQUEL ‹CHA76› :

```
( SELECT   nos:
  FROM     INSCRIT )  IS  IN   ( SELECT   noss
                                 FROM     ETUDIANT )
```

La seconde correspond à la destruction d'une valeur d'un attribut clé primaire. Si dans la relation ETUDIANT, on supprime l'étudiant de noss = 125009933, on devrait supprimer toutes les inscriptions correspondantes à cet étudiant ‹COD81› (tous les tuples de la relation INSCRIPTION dont le Noss = 125009933 doivent être détruits).

Pour exprimer l'intégrité référentielle, C. DATE a défini un langage dans ‹DAT81›, mais à l'heure actuelle aucun SGBD ne vérifie ce type de CIS. Signalons qu'elle figure dans les spécifications du système REBU ‹MIR82›.

3.2.3 QUELQUES EXEMPLES -

Sur les deux bases précédentes, les CIS suivantes sont définies :

a) noss est un entier compris entre 0 et 99999999.

b) une inscription correspond à un étudiant de la relation ETUDIANT.

c) le salaire d'un employé ne fait que croître.

d) la moyenne des salaires du département 25 est supérieur à 5000F.

3.3 CLASSIFICATION DES CONTRAINTES

Etant donné la grande variété des contraintes d'intégrité, beaucoup de classifications ont été proposées <BAD79>, <BOU79>, <STO75> et <JOU79>. Elles sont généralement classées suivant les critères de complexité, de coût ou de dépendance vis-à-vis du schéma ou des données. Nous proposons pour notre part une classification qui tient compte de tous ces critères présentée sous forme d'arborescence.

3.3.1 CRITERES DE CLASSIFICATION -

Les critères de classification les plus importants sont les suivants :

. La complexité de l'assertion : Prenons un exemple concret, si nous disposons de la relation EMPLOYE (nom, sal, dept, chef), et définissons deux contraintes :
La première, le salaire de tout employé du département D1 est supérieur au salaire de tout employé du département D2.
La seconde, le salaire de tout employé du département D1 est inférieur au salaire du chef du département D1.

De ces deux exemples, on déduit qu'il est beaucoup plus compliqué d'évaluer la première contrainte que la seconde, car la première est du "type *n-n*" alors que la seconde est du "type *n-1*".

. Le coût de l'assertion, c'est généralement le nombre d'accès disque qu'on effectue lors de son évaluation. Les assertions contenant des fonctions d'agrégation sont les plus coûteuses.

. La Syntaxe de l'assertion qui représente la partie logique de la CIS.

. Les CIS individuelles ou ensembllstes <ADE82> : On désigne par individuelle une CIS que doit vérifier un tuple, individuellement (exemple 4000 <= sal <= 20000), et ensembliste quand elle porte sur un ensemble de tuples. Badal <BAD79> est allé encore plus loin dans la classification en considérant les ensembles de données mis en jeu dans l'assertion. Ces ensembles de données peuvent être indépendants ou dépendants des données préexistantes dans la base.

. sal < 10000 est une CIS indépendante des données de la base lors de son évaluation.

. NEW.sal > OLD.sal est une CIS dépendante des données.

. Les CIS intra-relations : C'est quand les attributs qui apparaissent dans la CIS appartiennent à une même et seule relation. Cette CIS peut être intra-relation verticale si on contrôle la valeur d'un attribut d'un tuple en fonction des valeurs de ce même attribut pour les autres tuples, ou intra-relation horizontale si on veut contrôler la valeur d'un attribut en fonction des valeurs des autres attributs.

. Les CIS inter-relations : Quand les attributs mis en jeu appartiennent au moins à deux relations différentes. Ce genre de contrainte exprime une dépendance sémantique entre deux relations.

. Les CIS dynamiques ou statiques : dynamiques si elles considèrent le passage d'un état à un autre, et statiques sinon. De plus les CIS dynamiques peuvent porter sur une ou plusieurs opérations de mise-à-jour (insertion, Modification et Destruction).

. Les CIS différées ou immédiates : immédiate, si on valide la CIS à chaque opération élémentaire sur la base, et différée, si on la valide à un autre moment ou à la fin de la transaction. En effet, une transaction peut effectuer plusieurs opérations sur la base de données, avant la fin de la transaction la base peut être momentanément dans un état incohérent, mais globalement elle peut être cohérente.

3.3.2 PROPOSITION DE REPRESENTATION -

Nous considérons deux grandes classes de contraintes sémantiques :

_ les CIS structurelles s'intéressent aux structures de données et donc au schéma relationnel de référence. Il s'agit des dépendances fonctionnelles, multivaluées, hiérarchiques, produits et les types des objets. En effet, elles peuvent être vues comme des contraintes d'intégrité. Dans ‹NIC79›, toutes ces dépendances sont exprimées en termes de formules bien formées ("wff").

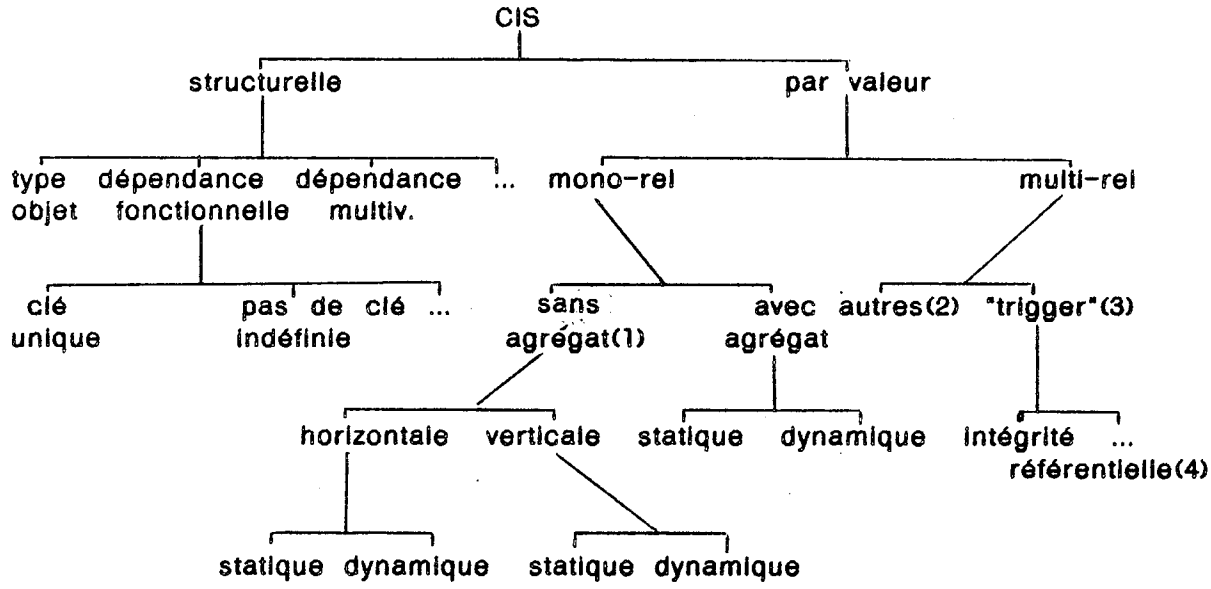
_ les CIS par valeur concernent les données (états de la base) et leurs transitions (passage d'un état à un autre). Elles s'expriment très facilement en terme d'algèbre booléenne.

Les contraintes de la première classe évoluent avec le schéma de la base de données alors que celles de la seconde peuvent être définies et détruites à tout moment, même si ce schéma reste stable.

Les CIS structurelles s'attachent à tous les tuples d'une relation alors que les CIS par "valeur" s'attachent à des

sous-ensembles de tuples d'une ou plusieurs relations.

De cette représentation ressortent les critères : de conceptualisation des CIS (structurelles, par valeur), de complexité quant à leur expression et leur vérification (mono-relationnelle, multi-relationnelle), ou de distinction entre CIS d'état ou de transition (statique, dynamique).



Classes et sous-classes des contraintes

(1) sans les fonctions d'agrégation (MOYenne, SOMme, MAXimum, MINimum et CARDinalité).
(2) tout autre CIS multi-relationnelle que les "trigger".
(3)"trigger": action spontanée (définie dans <ABR72>, équivalent au "on condition" de PL/1).
(4) intégrité référentielle (Cf. article de C. DATE <DAT81>).

On déduit 4 types de CIS mono-relationnelles :

- statique sans agrégat (ss)
- statique avec agrégat (sa)
- dynamique sans agrégat (ds)
- dynamique avec agrégat (da)

3.4 EXPRESSION DES CONTRAINTES

Dans le contexte relationnel, deux types de langages sont disponibles pour exprimer des CIS. Le premier type, très puissant, est issu de la logique du premier ordre. Il s'agit des formules bien formées <BER81>, <BER82>, <BLA81>, <HOM81> et <NIC79> similaire au calcul relationnel défini par E.F. CODD. Le second type, pour l'utilisateur d'une base de données, rassemble les langages QUEL dans <STO75>, SEQUEL dans <CHA76>.

Pour illustrer ceci, considérons la contrainte suivante : "pour tout étudiant inscrit à une unité de valeur, alors l'unité existe."

Elle s'exprime ainsi dans le premier type :

$\forall x \quad \forall y \quad (\quad \exists z \quad \exists w \quad$ INSCRIT(x,y,z,w) $\rightarrow \quad \exists t \quad \exists d \quad$ UNITVAL(y,t,d))

Et en SEQUEL :

```
(SELECT nouv
 FROM INSCRIT) IS IN   (SELECT nouv
                        FROM UNITVAL)
```

Pour notre part, on utilisera le langage MIQUEL, langage assez proche du langage SEQUEL. Initialement MIQUEL existe comme interface de définition et de manipulation de données et de structures dans MICROBE <FRT81a>. Il a été étendu pour tenir compte de l'intégrité sémantique dans un environnement micro-SGBD. La syntaxe complète de MIQUEL est présentée en annexe. L'interface utilisateur d'ISIS est composée de 4 commandes :

- ASSERT ---> Création de CIS.
- DROP ---> Supression de CIS.
- LISTCIS ---> Affichage des CIS et
- TESTCIS ---> Test de CIS.

Voyons comment exprimer dans MIQUEL les exemples de CIS présentées dans les pages précédentes (Cf. chapitre 5.2.3) :

a) noss IS IN (0, 99999999)

b)
```
( SELECT noss
    FROM INSCRIT ) IS IN ( SELECT noss
                           FROM ETUDIANT )
```

c) NEW.sal >= OLD.sal

d) (SELECT MOY (sal)
FROM EMPLOYE
WHERE nodept = 25) >= 5000.

4.0 TRADUCTION DES REQUETES ET DES CONTRAINTES : ASPECTS SEMANTIQUES

4.1 SYNTAXE ET SEMANTIQUE DES LANGAGES D'INTERROGATION

4.1.1 INTRODUCTION -

Avant d'exposer les concepts sémantiques associés aux langages d'interrogation de bases de données, il faut noter les deux niveaux généralement admis :

_ Le premier niveau s'appelle "terme".
_ Le second "formule".

Le "terme" décrit un sous-ensemble d'objets donc de tuples d'une relation tandis que la "formule" exprime l'état de la base (la réponse est vrai ou faux) ‹LIP79› et ‹NIC79›.

Ce qui correspond à la notion de "formule ouverte" (terme) et "formule fermée" (formule) dans la logique du premier ordre ‹ADE82›. Ces concepts permettent :

_ d'abord, de définir la notion de valeur d'une requête,

_ et ensuite, de proposer un ensemble de règles de passage d'une expression de requête à une autre tout en conservant la sémantique de celle-ci.

4.1.2 DEFINITION SYNTAXIQUE DES TERMES -

Les termes sont construits à partir des éléments suivants :
. le terme atomique (ou condition atomique)
. les opérateurs F , V , not , or , and , ->
. et les parenthèses) , (

Plus exactement l'ensemble T des termes est défini avec ces deux propriétés :

(1) F , V , terme atomique $\in$ T

(2) not t , (t or s) , (t and s) , (t->s) $\in$ T si t et s $\in$ T

un terme atomique est d'une des formes suivantes :

A ‹op› constante
A ‹op› B
A $\in$ ensemble
A $\notin$ ensemble

Où A , B sont des attributs de type compatible

<op> ∈ { =, <>, <, <=, >, >= }
constante est une valeur de même type que A,
ensemble est une collection de valeurs { a1 , a2 , ... an } ou
un intervalle (exemple dans les réels [1 , 2])

Remarque :

_ (t -> s) est équivalent à (not t) or s

Exemple de terme : soit l la longueur d'un objet tel que l prend ses valeurs dans R+ .
On veut sélectionner les objets dont la longueur est :
_ soit inférieure à 200
_ soit dans l'intervalle [300 , 400].
Le terme correspond à l'expression suivante :
(l < 200) or (l ∈ [300 , 400])
Supposons que R est une relation d'une base de données, un terme t correspond en SEQUEL à la requête suivante :

```
SELECT   *
FROM     R
WHERE    t
```

4.1.3 DEFINITIONS SEMANTIQUES -

La valeur d'une requête Q est notée ||Q|| : c'est la réponse du système à la requête Q.

Si nous supposons les correspondances suivantes entre opérateurs de l'algèbre booléenne et opérateurs de l'algèbre relationnelle :

_ F correspond à $\emptyset$
_ V " R
_ not " Complément (R)
_ or " union
_ and " intersection
_ -> " (complément (A)) union B

Nous avons la définition suivante :

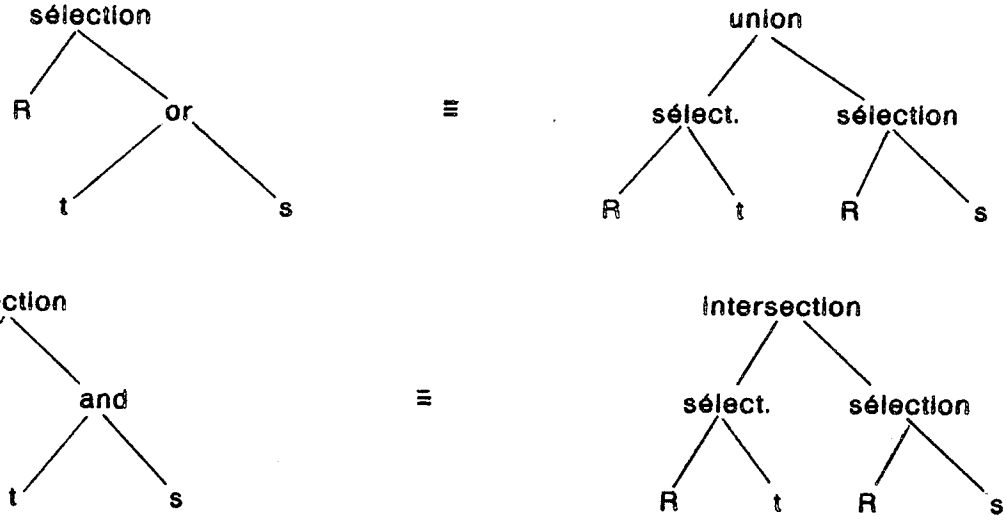
Définition : la valeur d'une requête dénotée ||Q|| est définie de façon inductive par :

les termes :

(1) ||condition atomique|| = { t ∈ R tel que la condition est vraie }

(2) ||F|| = ∅ , ||V|| = R
(3) ||not t|| = R − ||t||
(4) ||t or s|| = ||t|| ∪ ||s||
(5) ||t and s|| = ||t|| ∩ ||s||
(6) ||t -> s|| = (R − ||t||) ∪ ||s||

L'exemple (SELECT * FROM R WHERE t) correspondant au terme t, peut s'écrire sous la forme algébrique sélection (R , t), alors on en déduit que les règles (4) et (5) peuvent s'écrire sous la forme arborescente :



Nous terminons ce paragraphe par une définition sur l'équivalence de 2 requêtes.

Définition : Deux requêtes P et Q sont équivalentes si et seulement si ||P|| = ||Q||.

4.2 TRADUCTION DE MIQUEL EN ARBORESCENCES

Nous avons étudié dans le paragraphe précédent la sémantique d'une requête, nous en avons déduit en particulier des règles de transformation de requêtes. La méthode qui suit travaille avec plusieurs blocs SELECT liés entre eux. Nous compléterons les 2 règles proposées par un autre ensemble de règles qui permettent la traduction complète d'une requête MIQUEL. Précisons d'abord le fonctionnement de la méthode de traduction.

4.2.1 LA METHODE -

Bien que toute requête MIQUEL est traduite en arborescence algébrique avant son évaluation, seule la commande SELECT nous intéresse ici. Les autres commandes sont soit construites à partir de celle-ci (cas MODIFY, DELETE et CREER_RELATION), soit leur traduction n'est pas l'objet de ce document.

La méthode consiste, dans une première phase, à traduire chaque bloc SELECT en une arborescence algébrique à partir d'un certain nombre de règles prédéfinies (REGLES I) tout en gardant les liens existants entre blocs. Dans la seconde phase on effectue les liaisons inter-bloc (s'ils existent) en insérant des opérateus binaires tels que jointure, intersection (cas and) et union (cas or) tout en respectant un deuxième ensemble de règles prédéfinies (REGLES II).

4.2.2 PREMIERE PHASE -

La commande d'interrogation est structurée en trois éléments, les clauses SELECT, FROM et WHERE. La première (SELECT) définit la liste des attributs que l'on souhaite obtenir, la seconde indique la (les) relation (s) d'où sont extraits ces attributs et la dernière donne la condition de sélection des tuples (cette clause est optionnelle).

La commande a la forme suivante :

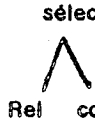
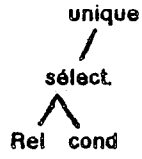
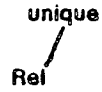
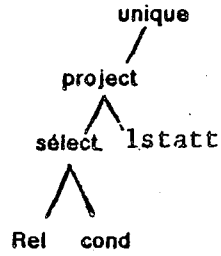
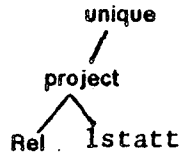
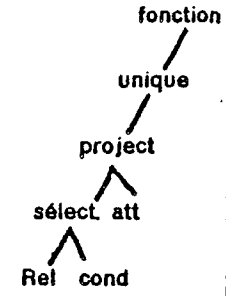
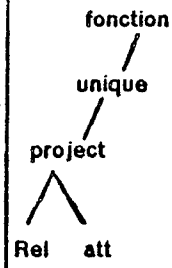
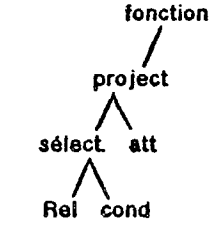
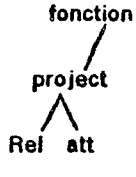
SELECT ‹liste-exp-select›
FROM ‹liste-relation›
[WHERE ‹booléen›]

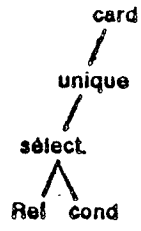
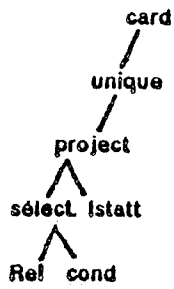
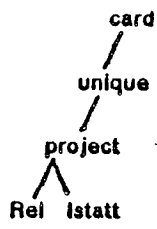
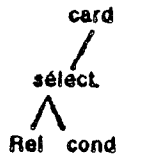
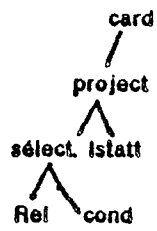
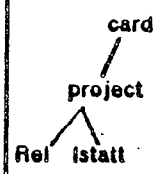
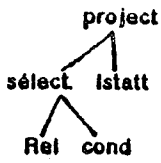
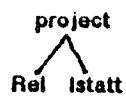
Dans cette phase on effectue les différentes analyses : lexicale, syntaxique et sémantique de la requête, avant de construire les sous-arbres en utilisant une première catégorie de règles (REGLES I, présentées dans la section suivante). Celles-ci donnent, pour chaque cas de ‹liste-exp-select›, le sous-arbre formé des opérateurs projection et sélection et des fonctions d'agrégation (MAX, MIN, MOY, SOM, CARD, UNIQUE).

4.2.2.1 REGLES I -

On présente pour chaque cas de bloc SELECT l'arborescence algébrique construite correspondante. De plus on donne le type du résultat du bloc (R = relation et N = numérique).

TABLEAU DES REGLES I :

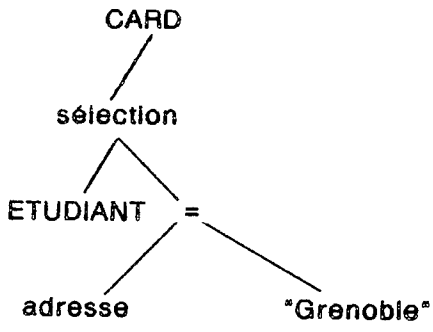
nO	niveau langage <lst-exp-sel>	arbre obtenue avec clause WHERE	arbre obtenue sans WHERE	résu-ltat.
1	*		Rel	R
2	unique *			R
3	unique <lstatt>			R
4	fonction (unique <attribut>)			N
5	fonction (<attribut>)			N

6	card (unique *)			N
7	card (unique ‹lstatt›)			N
8	card (*)			N
9	card (‹lstatt›)			N
10	‹lstatt›			R

4.2.2.2 EXEMPLE DANS LE CAS D'UN SEUL BLOC -

```
SELECT CARD ( * )
FROM   ETUDIANT
WHERE  adresse = "Grenoble" ;
```

On obtient l'arbre :



4.2.2.3 EXEMPLES DANS LE CAS DE PLUSIEURS BLOCS -

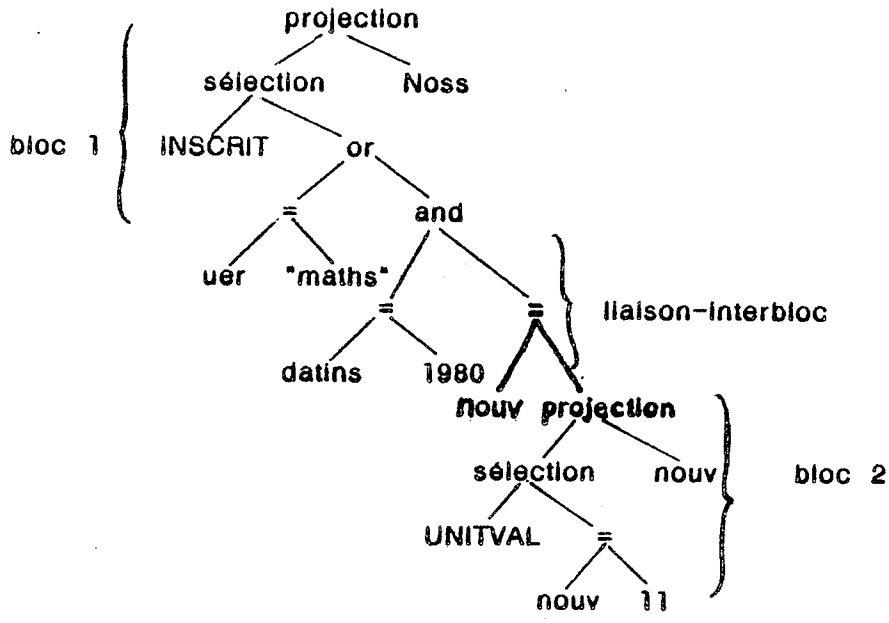
Si on veut connaitre les numéros de sécurité sociale (noss) des étudiants inscrits :

- en 1980 à l'unité de valeur numéro 11
- ou à l'uer de maths.

On peut l'exprimer en MIQUEL :

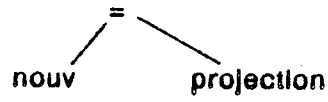
```
bloc 1 { SELECT   noss
       { FROM     INSCRIT
       { WHERE    ( uer = "maths" )
                   or
                ( ( nouv =    SELECT   nouv       }
                              FROM     UNITVAL    } bloc 2
                              WHERE    nouv = 11 )}
                  and
                  ( datins = 1980 ) ) ;
```

Une fois la première phase effectuée, on obtient l'arborescence suivante :



Remarque sur l'arborescence :

l'arborescence produite suit la structure des blocs 1 et 2 de la requête. La liaison entre les 2 blocs est assurée par :



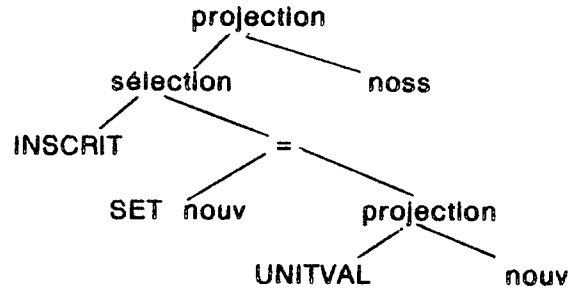
c'est une situation incorrecte, mais elle garde la liaison entre les 2 blocs. Sémantiquement ça veut dire que nouv ∈ l'ensemble des tuples de la relation intermédiaire produite par la projection. Les règles II du paragraphe suivant permettent de supprimer ce genre d'incorrection en insérant à la place des opérateurs algébriques de liaison tout en conservant la sémantique de la requête. Dans ce cas, l'opérateur inséré est la jointure de relations.

Exemple avec le mot clé "SET" de MIQUEL :

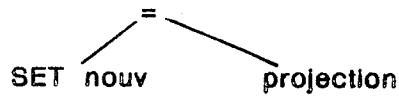
On veut connaitre les noss des étudiants inscrits à toutes les unités de valeur. Avec MIQUEL on obtient :

```
SELECT noss
FROM   INSCRIT
WHERE  SET nouv = SELECT nouv
                  FROM   UNITVAL ;
```

On obtient l'arborescence suivante :



Comme dans le cas précédent, l'arbre contient une sitation de liaison incorrecte :

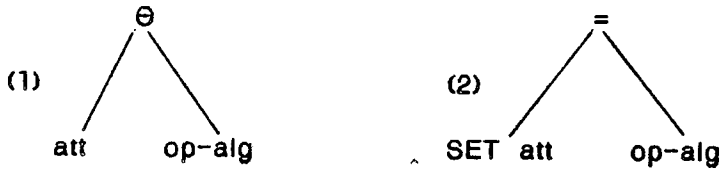


sémantiquement, ça veut dire que l'ensemble des "nouv" pour un "noss" donné est équivalent à l'ensemble des "nouv" obtenus via la projection. Dans ce cas on remplace cette liaison par une jointure suivie d'une anti-projection.

4.2.3 SECONDE PHASE -

C'est la phase la plus importante, en effet c'est dans cette phase que l'on construit l'arborescence algébrique complète. Elle utilise comme entrée l'arborescence fournie par la première phase et un noyau de règles prédéfinies appelées REGLES II.

Ces règles permettent de faire une liaison inter-bloc (correcte) chaque fois qu'on détecte dans l'arborescence en cours les deux situations de liaison (1) et (2) (incorrectes) suivantes :

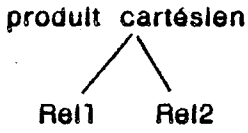


où "op-alg" est un des opérateurs de l'algèbre relationnelle.

4.2.3.1 REGLES II : LIAISON INTER-BLOC -

Dans le tableau des règles qui vont suivre nous avons utilisé quelques abréviations que nous allons définir :

_ Rel-from représente soit la relation du bloc SELECT en cours si elle est unique dans <liste-from> soit le sous-arbre suivant dans le cas contraire :



_ Σdom représente la liste des domaines de Rel-from.

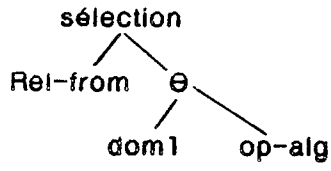
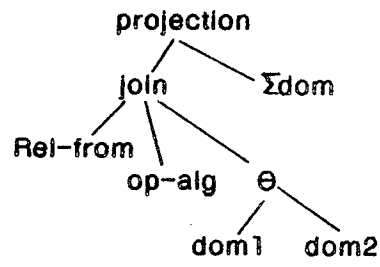
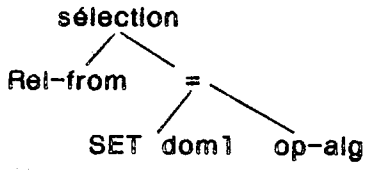
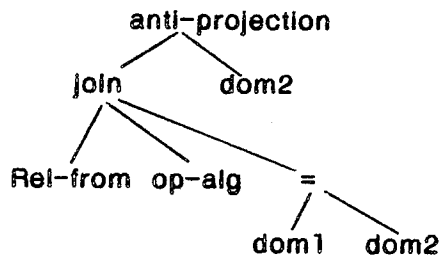
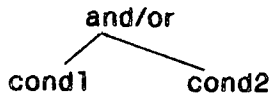
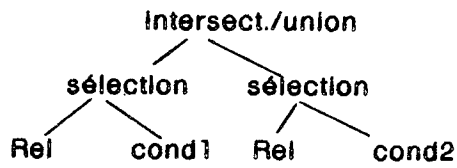
_ dom1 est le domaine du bloc SELECT en cours. Il permet de faire la liaison avec le bloc SELECT inférieur.

_ dom2 représente le domaine contenu dans <liste-exp-select> du bloc inférieur.

_ op-alg signifie opérateur algébrique, représente ici la racine du sous-arbre du bloc directement inférieur.

_ θ est une des connectives suivantes : =, >, >=, etc ...

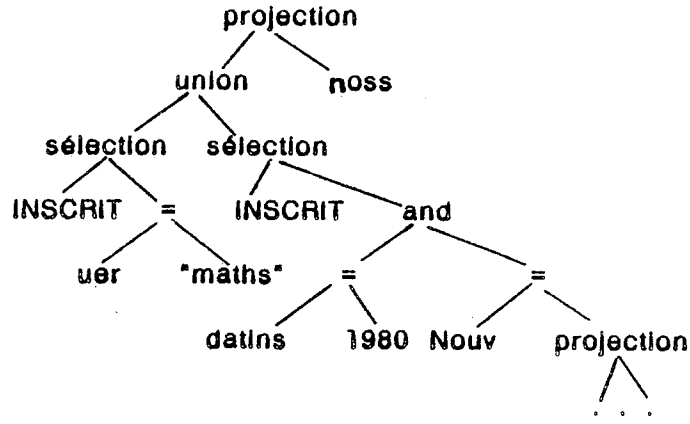
TABLEAU DES REGLES II

no	situation	transformation
(R1)		
(R2)		
(R3)		

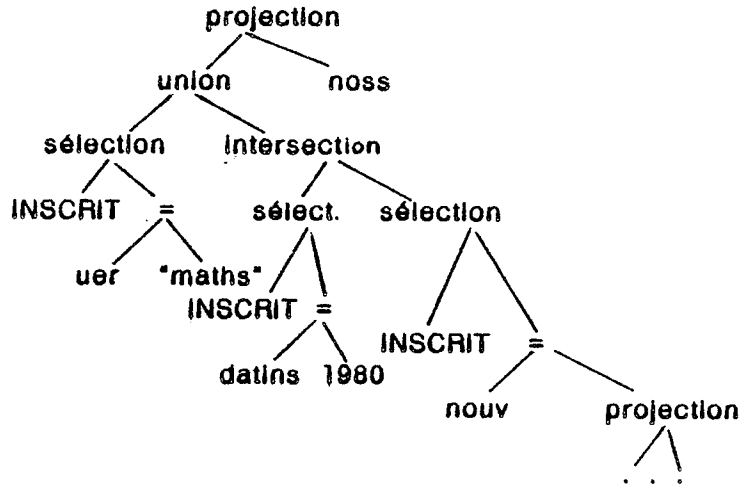
4.2.3.2 EXEMPLE -

La figure de l'exemple précédent (cf. 4.2.2.3) montre l'arborescence de la requête telle qu'elle a été fournie par la première phase. Passons maintenant à la seconde phase.

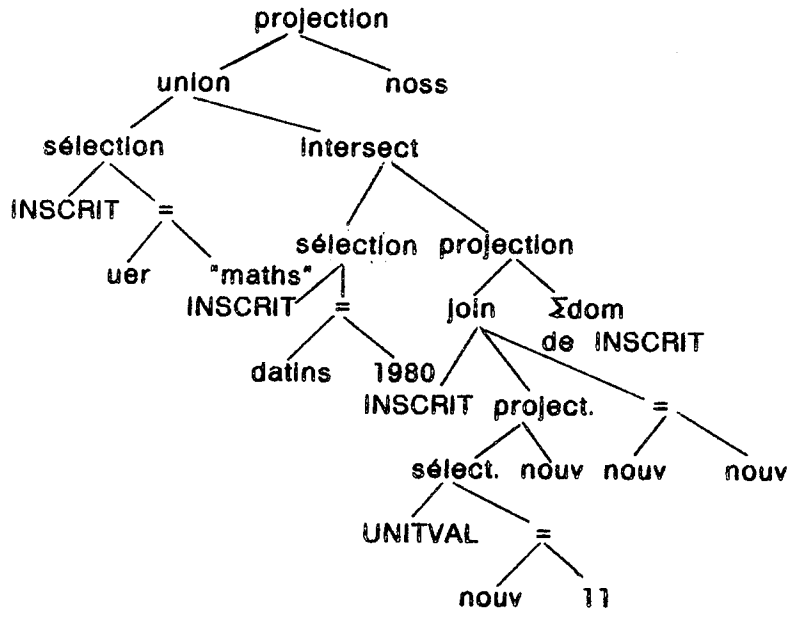
_ Appliquons la règle R3 sur l'opérateur logique "or", on obtient après modification l'arborescence suivante :



_ Ensuite appliquons toujours la règle R3 sur l'arbre ainsi produit :



_ Enfin appliquons la règle R1, on obtient l'arbre final suivant :



4.2.4 PREUVE DES REGLES II -

Pour prouver les règles II de traduction qui sont déjà présentées, on utilisera des transformations au niveau des opérateurs algébriques qui maintiennent l'équivalence sémantique.

4.2.4.1 TRANSFORMATIONS UTILISEES -

_ Première transformation :

D'après 4.1.2 ||t or s|| = ||t|| U ||s|| . Ce qui est équivalent au niveau algébrique à :

sélection(R: t or s) = union (sélection(R:t),sélection(R:s))

_ Seconde transformation :

De même ||t and s|| = ||t|| ∩ ||s||, ce qui est équivalent à :

sélection(R: t and s)
=
intersection(sélection(R:t),sélection(R:s))

_ Troisième transformation :

Soient 2 relations R(X,Y) et S(U,V). La division se définit ainsi.

$$\|R(X,Y)\ [X \div Y]\ S(U,V)\| = \forall\ a,\ \|R(a,Y)\|,\ a \in [U]S$$

L'opérateur division s'interprète aussi en fonction du θ-produit et de l'anti-projection :

$$R(X,Y)\ [X \div U]\ S(U,V) =]Y[\ (\ [X,Y](R(X,Y)\ |X=U|\ S(U,V)))$$

Dans le cas où la relation S n'est composée que du constituant U, on déduit la troisième transformation :

R (X,Y) [X ÷ U] S (U) =]Y[(R (X,Y) [X = U] S (U)) ou bien
Division (R (X,Y) , S (U) , X ÷ U) =
anti-projection(θ-join(R(X,Y),S(U), X=U) , Y).
Cette formule sera utilisée dans le paragraphe suivant.

4.2.4.2 PREUVE DES REGLES -

Le tableau exposé en 4.2.3.1 nous donne les règles de transformations R1, R2 et R3. Nous allons montrer que ces règles respectent l'équivalence sémantique de la requête.

REGLE R1

Dans la démonstration, pour faciliter la lecture on prend : θ = "=", la démonstration est rigoureusement la même pour θ quelconque. La situation (1) incorrecte syntaxiquement, obtenue en phase 1, peut s'écrire :
sélection (Rel : dom1 ∈ op-alg) (a)
On effectue la transformation suivante :
θ-join (Rel.op-alg.dom1=dom2) [tous dom de Rel] (b)
puisque
(b) = (((r^s) : r ∈ Rel , s ∈ op-alg et r[dom1] = s[dom2])

[tous dom de Rel]

=((r^s) :r ∈ Rel ,r|dom1| ∈ (s1,s2,...)) [tous dom de Rel]
puisque op-alg est composée d'un seul domaine dom2 et prenant les valeurs s1,s2,...

= (r : r ∈ Rel et r[dom1] ∈ (s1,s2,...))

= (a)

REGLE R2

On obtient dans la première phase select (Rel : SET dom1 = op-alg), ce qui est équivalent sémantiquement à :
division (Rel(X.dom1),op-alg(dom2),dom1 ÷ dom2). (c)

En utilisant la 3ème transformation citée plus haut on obtient :
anti-projection(θ-join(Rel,op-alg,dom1=dom2),dom2). (d)

REGLE R3

La démonstration de cette règle est très simple, il suffit d'appliquer l'une des deux transformations citées en 4.1.2.3.

4.2.4.3 ALGORITHMES DE TRADUCTION -

Il existe deux algorithmes qui effectuent la liaison interbloc :

_ le premier est "ascendant", c'est à dire qu'une fois détecté le cas (1) ou (2) (voir 4.2.3), on remonte l'arborescence conditionnelle jusqu'à l'opérateur sélection et on effectue les transformations nécessaires suivant les trois règles R1, R2 et R3 (Cf. Chap. 4.2.3).

_ le second, contrairement au premier, détecte les deux situations de liaison, et recherche un chemin en remontant l'arborescence conditionnelle jusqu'à l'opérateur sélection, ensuite il effectue un parcours de ce chemin de haut en bas (la fin étant l'une des deux situations de liaison) et insère les opérateurs nécessaires suivant les trois règles.

Le second algorithme est le meilleur, le nombre de règles est réduit à trois. Mais on est obligé de faire un parcours de l'arbre pour la recherche du chemin. Dans MICROBE, nous avons opté pour le premier pour des raisons "historiques". Dans la démonstration précedente nous avons opté pour le second cas car il suffit de prouver un nombre minimum de règles (trois).

4.3 TRADUCTION DES CONTRAINTES EN ARBORESCENCES

C'est la partie booléenne de la commande ASSERT qui est traduite . En effet, cette partie peut avoir des blocs SELECT et leur traduction en arboresences est donnée en détails dans les paragraphes

précédents.

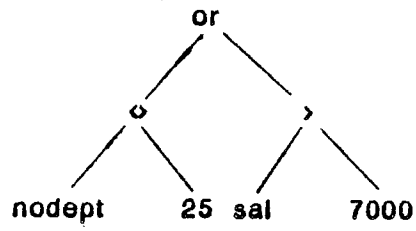
1er cas : l'assertion ne contient pas de bloc SELECT

Prenons l'assertion suivante :

Les salaires des employés du département 25 sont supérieurs à 7000 :

ASSERT C1 ON EMPLOYE : nodept <> 25 or sal > 7000

La partie booléenne sera traduite comme suit :

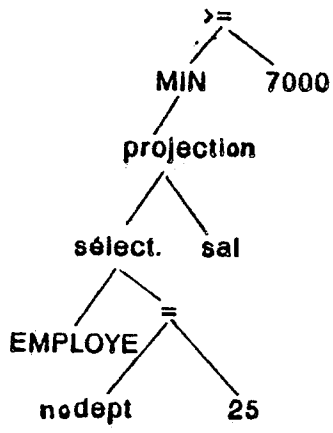


2ème cas : l'assertion contient un bloc SELECT

Supposons que l'on veuille exprimer dans la relation EMPLOYE. que le minimum des salaires des employés du département 25 est supérieur ou égal à 7000.
Dans notre système Isis. on écrit :

```
ASSERT C1 ON EMPLOYE :    ( SELECT   MIN ( sal )
                          FROM       EMPLOYE
                          WHERE      nodept = 25 )   >= 7000
```

On produit donc l'arbre suivant :

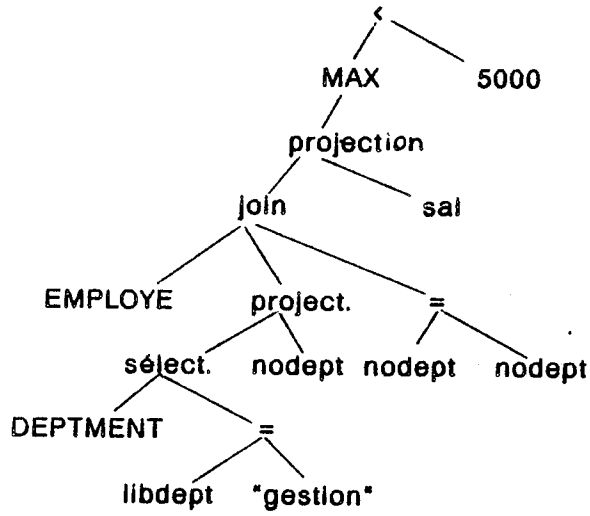


3ème cas : l'assertion contient plusieurs blocs SELECT

Soit la CIS suivante : "le maximum des salaires des employés du département gestion est inférieur à 5000F".
En MIQUEL on l'exprime ainsi :

```
( SELECT    MAX (sal)
  FROM      EMPLOYE
  WHERE     nodept      =    SELECT     nodept
                             FROM       DEPTMENT
                             WHERE      libdept = "gestion") < 5000
```

Après traduction on obtient l'arborescence suivante :



4ème cas : L'assertion contient BETWEEN ou IS IN

Dans le cas où on a l'assertion suivante:
<att> BETWEEN <param1> AND <param2>, avant la traduction on effectue la transformation suivante :
(<att> >= <param1>) AND (<att> <= <param2>).

De même l'assertion du type :
<att> IS IN (val1, val2 , ..., valn) est équivalente à :
(<att> = val1) OR (<att> = val2) OR ... OR (<att> = valn).

5.0 GESTION DES CONTRAINTES

5.1 INTRODUCTION

Une fois la contrainte traduite en arborescence algébrique celle-ci va être optimisée en vue d'une réduction de son temps d'évaluation. C'est une optimisation algébrique basée sur un ensemble de règles prédéfinies. Par application de celles-ci on élimine les sous-arbres homogènes (i.e. les opérations du sous-arbre font référence à une seule relation) et on effectue la distribution des opérateurs de sélection et de projection <WIN82>. Cette méthode sera détaillée au moment de la présentation du projet MICROBE, car elle s'applique de la même façon aux requêtes de MIQUEL.

Ensuite deux testssont effectués :

Le premier concerne la consistance des CIS, cette notion sera développée en détail dans ce chapitre. Il s'agit du problème des "conflits logiques" qu'elles peuvent induire. Elle concerne aussi bien les CIS statiques que dynamiques. Nous nous limitons dans un premier temps au problème de consistance pour les CIS statiques.

Le second permet de tester les CIS sur les données préexistantes dans la base.

Si ces deux tests donnent des résultats corrects, la CIS est enregistrée dans un catalogue INTEGRITE, sinon elle est rejetée.

5.2 CONFLITS LOGIQUES ENTRE CONTRAINTES

5.2.1 DEFINITIONS -

Un attribut est dit contraint, s'il apparait dans la partie booléenne d'une assertion.

Exemple : ASSERT CS1 ON EMPLOYE : sal < 10000 AND sal >=0
L'attribut contraint ici est sal, d'ailleurs son domaine est réduit à l'ensemble [0, 10000 [

Considérons l'assertion suivante :
ASSERT CS2 ON EMPLOYE : sal < 10000 AND sal > 11000

Le domaine de l'attribut "sal" est réduit à l'ensemble vide. Dans ce cas, aucune insertion dans la relation EMPLOYE ne pourrait être faite, car aucun tuple ne pourrait vérifier la contrainte.

Nous dirons qu'une contrainte est en conflit avec elle-même si le domaine d'un des attributs contraints est réduit à l'ensemble vide.
Exemple : CS2 est en conflit avec elle-même.

Plus généralement, nous dirons qu'une contrainte du type statique est en conflit avec elle-même, ou avec une ou plusieurs CIS statiques si en tenant compte de l'ensemble des arborescences conditionnelles reliées par l'opérateur logique "and", il existe un attribut contraint dont le domaine est réduit à l'ensemble vide.

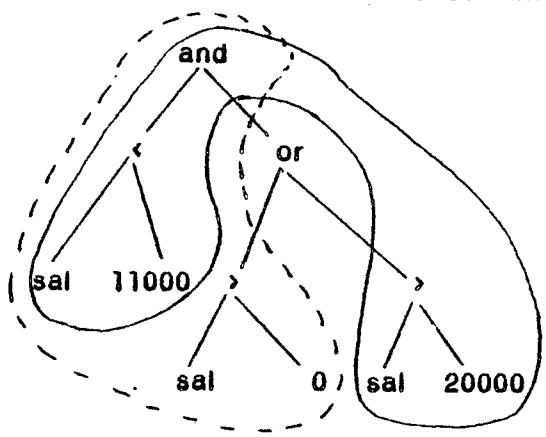
Lors de la définition d'une contrainte, on effectue un test pour éviter les conflits entre contraintes. Si c'est le cas, la contrainte est automatiquement rejetée.

5.2.2 PRINCIPES DE SOLUTION AU PROBLEME DES CONFLITS -

Pour bien comprendre cette résolution, nous allons procéder par un exemple. Soient deux contraintes C1 et C2 :

- ASSERT C1 ON EMPLOYE : sal < 11000
- ASSERT C2 ON EMPLOYE : (sal > 0) OR (sal > 20000).

En supposant, que seules ces deux contraintes contiennent "sal" comme attribut contraint. Pour tester si C1 et C2 ne sont pas en conflit, il faut considérer l'arborescence suivante :



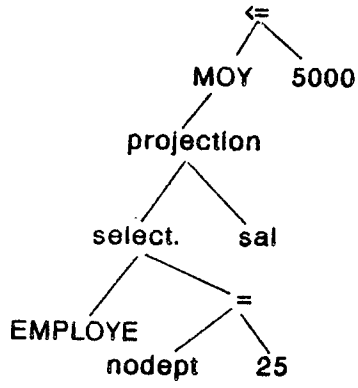
Dans cet exemple, l'évaluation de l'arborescence nous donne deux possibilités, la 1ère en pointillés la seconde en trait plein.
L'évaluation des deux cas nous donne :

- 1ère possibilité : sal $\in$] 0 , 11000 [
- 2ème possibilité : sal $\in$ ensemble vide

On en déduit, puisqu'au moins une possibilité (la 1ère) nous donne un salaire appartenant à un ensemble non vide, les deux contraintes C1 et C2 ne sont pas en conflit. Nous remarquons d'après cet exemple, que le nombre de possibilités d'évaluation de l'arbre dépend du nombre d'opérateurs logiques "or" qui y figurent.

5.2.3 CONTRAINTES AVEC OPERATEURS ALGEBRIQUES -

Soit la CIS suivante, "la moyenne des salaires des employés du département 25 est inférieure ou égale à 5000F" traduite par cette arborescence.



Dans ce cas précis, "sal" est bien l'attribut contraint. On ne sait pas exactement dans quelle zone du segment] - oo , + oo ["sal" prend ses valeurs. Mais si on suppose que "sal" est un nombre positif, on peut déduire de l'arbre précédent au moins la contrainte suivante :

sal <= CARD (RI) * 5000 où CARD (RI) est la cardinalité de la relation intermédiaire entre la fonction d'agrégation et la projection.

Dans le cas où sal est positif et C une constante entière ou réelle, on peut déduire le tableau suivant :

Tableau des assertions agrégatives et de leurs conséquences sur le domaine : cas sal >=0.

assertion algébrique	contrainte sur le domaine de l'attribut.
MOY (sal) <= C	sal <= Card (RI) * C
MOY (sal) = C	sal <= Card (RI) * C
MOY (sal) < C	sal < Card (RI) * C
MOY (sal) <> C	?
MOY (sal) > C	?
MOY (sal) >= C	?
MAX (sal) <= C	sal <= C
MAX (sal) < C	sal < C
MAX (sal) = C	sal <= C
MAX (sal) <> C	?
MAX (sal) > C	?
MAX (sal) >= C	?
MIN (sal) <= C	?
MIN (sal) < C	?
MIN (sal) = C	sal >= C
MIN (sal) <> C	?
MIN (sal) > C	sal > C
MIN (sal) >= C	sal >= C
SOM (sal) <= C	sal <= C
SOM (sal) < C	sal < C
SOM (sal) = C	sal <= C
SOM (sal) <> C	?
SOM (sal) > C	?
SOM (sal) >= C	?
CARD(sal) op C	?

Pour efectuer un test de consistance sur les asertions agrégatives. Il faut remplacer d'abord l'arbre correspondant par la partie droite du tableau dans notre cas (Exemple : MAX (sal) <= C sera remplacé par sal<=C).

5.2.4 METHODE DE RESOLUTION DES CONFLITS -

Supposons une contrainte définie sur la base, on voudrait tester si elle n'est pas en conflit avec elle-même ou avec celles déjà existantes dans la base. Alors, pour chaque attribut contraint dans la CIS à tester, on recherche d'abord toutes les assertions qui ont ce même attribut contraint. On relie toutes ces arborescences par un opérateur logique "and". A partir de l'arborescence ainsi produite, on recherche toutes les possibilités d'évaluation de l'arbre.

COMMENT ?

Pour chaque opérateur "or" de l'arbre, on construit deux arborescences :

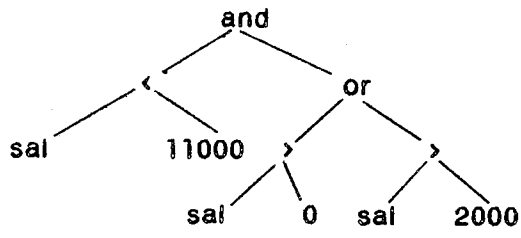
Si l'opérateur "or" est une racine de l'arbre, on produit deux arbres, l'arbre gauche et l'arbre droit du "or" auxquels on supprime ce même opérateur.

Dans le cas où l'opérateur "or" n'est pas une racine, on produit un premier arbre composé de l'arbre traité à qui on enlève la partie gauche en partant du "or" et l'opérateur "or" lui-même, et un 2ème inverse du 1èr. On répète l'opération jusqu'à ce que les opérateurs logiques "or" disparaissent de tous les arbres produits.

Lors de l'évaluation des conditions atomiques, certaines ont des valeurs indéfines "?" (voir tableau précédent). Dans ce cas on ne tient pas compte de cette valeur.

Exemple : Reprenons le cas de deux CIS C1 et C2 du chapitre 5.2.2.

1) Ici l'attribut contraint est "sal", on relie les 2 arbres par un "and", on obtient l'arbre suivant :



2) Il existe un "or" dans l'arborescence, lors de sa suppression on obtient 2 arbres A1 et A2 :

3) Le premier arbre A1 donne sal $\in$] 0, 11000[.
le second donne sal $\in \varnothing$.

On en déduit que les 2 CIS sont consistantes, puisque au moins un arbre final donne un salaire n'appartenant pas à l'ensemble vide.

Exemple non solvable :

Considérons la relation ENSEIGNANT (nom, profession, classe, grade) avec les trois CIS suivantes :

CIS1 : Tout enseignant assistant est de classe B.
CIS2 : Tout enseignant de classe B est ingénieur.
CIS3 : Tout ingénieur n'est pas assistant.

Ces 3 CIS sont inconsistantes, car de CIS1 et CIS2 on déduit que "tout enseignant assistant est ingénieur", ce qui est en contradiction avec CIS3.

Notre méthode appliquée à ce cas ne déterminera pas cette contradiction, car elle n'est pas munie d'un moyen déductif.

5.2.5 LIMITES ET CONCLUSION -

Notre méthode, sans mécanisme déductif, est limitée quant aux CIS qu'elle traite. Néanmoins, elle détecte un grand nombre de conflits dans le cas d'une base de données reélle.

Le problème de consistance des CIS est très connu en démonstration automatique ("theorem proving"). Pour l'ensemble des CIS exprimées dans la théorie des prédicats du premier ordre, ce problème n'est pas décidable, mais semi-décidable. Mais les "wff" qui expriment les énoncés de dépendances fonctionnelle et multivaluée ont la propriété d'être toujours consistantes.

5.3 CATALOGUE "INTEGRITE"

Le catalogue INTEGRITE stocke les assertions. Celui-ci sera géré directement par le système ISIS. Il s'ajoutera au catalogue MIMER déja existant. Ce catalogue INTEGRITE est composé de deux relations

internes.

5.3.1 LES RELATIONS CATALOGUES -

Relation CONTRAINTE

Cette relation stocke les corps des CIS, chacun de ses tuples est un descripteur d'une CIS. Ce descripteur contient les informations suivantes : le témoin d'existence, le nom de l'assertion, le mode; donc les opérations sur lesquelles porte l'assertion (I,M,D,IM,ID,MD et IMD : où IMD par exemple signifie que les 3 opérations de mise à jour insertion, modification et destruction portent sur l'assertion), le nom interne de la relation, le type de la contrainte (ss = statique sans agrégat, ds=dynamique sans agrégat, sa = statique avec agrégat et da = dynamique avec agrégat), la valeur de l'agrégat si le type de l'assertion est sa ou da et enfin un pointeur vers la racine de l'arbre prédicat dans la relation catalogue ARBRE.

Relation ARBRE

Chaque tuple décrit un noeud de l'arbre associé à l'assertion. Il sera composé de 4 valeurs : type de l'opérateur (logique ou algébrique), le numéro de l'opérateur, le pointeur fils et le pointeur frère.

5.3.2 SCHEMA DU CATALOGUE INTEGRITE -

RELATION "CONTRAINTES"

témoin (0/1)	nomcis	mode op.	nom interne relation	type CIS	valeur agrégat	ptr
1	CIS1	IMD	2	ds	nil*	2
1	CIS2	M	4	sa	462	6
.						
.						
.						

RELATION "ARBRE-CIS"

type noeud	numéro	ptr-fils	ptr-frère
'C'	9	3	nil
'D'	6	nil	4
'K'	3	nil	nil
.			
.			
.			

* toute cis sans agrégat aura pour valeur nil.

5.4 L'INTERFACE DE BAS NIVEAU

Sur le catalogue INTEGRITE sont définies les quatre opérations de base suivantes : (Elles sont utilisées par le concepteur de base de données, ici au niveau de MIOPE en vue de la validation des opérations de mise à jour par exemple).

Créer-cis : Création d'une CIS
Détruire-cis : Destruction de CIS sous condition
Obtenir-cis : Obtention de CIS suivant une condition
Modifier-cis : Modification des paramètres d'une CIS

L'architecture fonctionnelle d'ISIS avec enchainement de ces opérations est présentée dans chapitre 8. section 8.1.

6.0 VALIDATION DES OPERATIONS DE MISE A JOUR

6.1 INTRODUCTION

Une base de données peut être mise à jour ou consultée. C'est à la mise à jour que la base évolue (i.e. les données de la base évoluent). Il existe trois opérations de mise à jour d'une base de données, l'insertion de nouvelles données (I), la suppression de données (D) et leur modification (M). Toute évolution des données doit satisfaire les contraintes d'intégrité sémantiques. C'est pour celà que chaque opération de mise à jour (I, M ou D) doit être validée. Valider, vis-à-vis des CIS, une opération de mise à jour, dont l'ensemble des attributs contraints et l'ensemble des attributs mis à jour ont une intersection non vide, consiste à vérifier que l'opération de mise à jour laisse la base dans un état cohérent (i.e. les données mises à jour satisfont les CIS associées). La méthode de validation présentée ici se place dans un contexte algébrique, elle ne traite pas la reprise en cas de violation de CIS et s'effectue en phase d'exécution au moindre coût (Cf. article de Badal et Popek <BPO79>).

La méthode de validation que nous présentons travaille avec des CIS mono-relationnelles (statiques ou dynamiques) avec les fonctions agrégats (MAX, MIN, ...) sous forme d'arborescences algébriques. Elle optimise l'évaluation des CIS en évitant, premièrement de tester des CIS qui ne le devront pas et deuxièmement de recalculer les valeurs des fonctions agrégats en assurant constamment leur maintenance.

6.2 METHODES DE VALIDATION EXISTANTES

6.2.1 INTRODUCTION -

Trois méthodes originales sont proposées. La première est celle du système INGRES <STO75> implémentée et appelée "query modification". La seconde est celle d'ESWARAN <ESW75> et <ESW76> du system-R. Cette deuxième méthode est basée sur la notion de "trigger" (déclencheur) qui est une généralisation de la notion d'assertion. C'est un ensemble d'instructions constituées d'un corps ("body") et d'une condition. Le corps du trigger est exécuté une fois la condition vérifiée. Enfin la troisième, est celle de HAMMER <HAM78>. L'idée principale de cette méthode de validation des transactions est d'analyser l'effet de chaque opération de mise à jour sur chaque contrainte. Viennent ensuite les méthodes optimisées <NIC79> et <BLA81>. Elles proposent des mécanismes d'optimisation du temps de vérification des CIS.

Toutes ces méthodes seront détaillées dans les paragraphes suivants.

6.2.2 LA METHODE INGRES -

Elle consiste à modifier la requête ("query modification") pour tenir compte des assertions mises en cause. Avec QUEL on définit des contraintes, celles-ci seront enregistrées dans le catalogue "INTEGRITY" sous forme de chaines de caractères et seront donc disponibles pour consultation.

6.2.2.1 ASSERTIONS SANS AGREGATS -

La syntaxe de l'insertion est de la forme suivante :

```
RANGE   OF   X1   IS   R1
  . . .
RANGE OF     Xn   IS   Rn

APPEND R ( D1 = f1, . . . . Dn = fn )
WHERE   Q
```

où :
X1, . . . ,Xn sont des variables n-uple sur les relations R1, . . . ,Rn.
R est le nom de la relation résultat.
D1, ...Dn sont les dommaines de R
f1, ...fn sont des fonctions valides de QUEL
Q est une assertion sur les variables X(1), ...X(N)
Q = Q (X(1), ..., X(N)).

Supposons que nous avons les assertions suivantes Q(1)(Y), ...Q(k)(Y) où Y est la variable n-uplet de R.

L'algorithme "query modification" pour l'opération APPEND est le suivant :

1) Obtenir toutes les assertions (sans agrégats) Q(1)(Y), ...Q(n)(Y) sur la relation R de l'opération APPEND.

2) Modifier l'assertion (Q) dans l'opération APPEND par (Q and Q*) où :

```
Q* = Q(1)(Y) (f(1), ...., f(n))
     and Q(2)(f(1), ...., f(n) )
     . . .
     and
        Q(k)(f(1), ...., f(n))
```

où : Q(j)(f(1), ..., f(n)) résulte de Q(j)(Y) en remplaçant Y.D(k) par f(k) quand Y.D(k) apparait dans Q(j).

Exemple : Considérons la requête d'insertion suivante :
RANGE OF X IS EMPLOYE
APPEND (nom='Dupont',prénom='Pierre',sal=13000, nodept=25)

Supposant que l'assertion suivante existe déja dans le catalogue :

RANGE OF X IS EMPLOYE
INTEGRITY X.sal < 10000

L'algorithme "query modification" nous donne :
RANGE OF X IS EMPLOYE
APPEND (nom='Dupont',prénom='Pierre', sal=13000, nodept =25)
WHERE 13000<10000

On remarque tout de suite que l'insertion ne sera pas effectuée car la condition 13000 < 10000 est fausse.

Nous avons donné ici un exemple sur l'insertion, l'opération de modification REPLACE est traitée de la même manière.

6.2.2.2 ASSERTIONS AVEC AGREGATS -

Pour expliciter la méthode utilisée, nous allons procéder par un exemple. Dans le système INGRES, l'opération QUEL d'insertion (APPEND) se décompose en 2 parties :

1) Une relation W est obtenue à partir des autres relations (opération RETRIEVE)

2) On fait une union entre W et la relation dans laquelle on veut insérer des tuples.

Supposons que l'on dispose de la CIS suivante sur la même relation que précédemment :
RANGE OF X IS EMPLOYE
INTEGRITY AVG (X.sal) < 14000 (1)
et que W est une relation union-compatible avec EMPLOYE. Nous voulons insérer tous les tuples de W dans EMPLOYE, alors nous écrivons :
RANGE OF X IS W
APPEND INTO EMPLOYE (Y.ALL)

Si on veut tenir compte de l'assertion (1) on écrit :
RANGE OF Y IS W
RANGE OF E IS EMPLOYE

```
APPEND INTO   EMPLOYE   (Y.ALL)

                    SUM (E.sal) + SUM (Y.sal)
WHERE               ------------------------- < 14000
                    COUNT(E.TID) + COUNT(Y.TID)
```

où TID est un identificateur de tuple garanti unique.

6.2.3 LA METHODE SYSTEM-R -

Pour le système R (system-R), une méthode de validation des transactions vis-à-vis des assertions, basée sur le concept de "trigger" a été proposée <ESW76>. Pour donner une idée générale, procédons par un exemple.

Exemple :

Chaque fois qu'un nouvel employé est inséré dans la relation EMPLOYE, on incrémente nbemp de 1 dans le tuple du département correspondant au nouvel employé. Soit alors le trigger suivant :

```
DEFINE TRIGGER TRIGEMP
ON INSERTION   OF   EMPLOYE :
( UPDATE DEPTMENT
  SET nbemp=nbemp+1
  WHERE nodept = NEW EMPLOYE.nodept)
```

Deux implantations ont été envisagées, l'une consiste à considérer les procédures "trigger" comme une partie du SGBD et de créer un nouveau module chaque fois qu'un trigger est défini. Cette approche est acceptable si la définition et la supression des trigger ne sont pas fréquentes. L'autre approche est de considérer les procédures trigger comme des objets de la base et de les traiter comme des ressources. Quand le sous-système d'intégrité s'aperçoit que la condition du trigger est satisfaite, il charge une copie de la procédure trigger, passe les paramètres, change le mode de verrouillage et donne le contrôle à la procédure. La procédure trigger terminée, le sous-système déverrouille et rend le contrôle à la procédure appelante.

6.2.4 LA METHODE DE HAMMER -

L'idée principale de la méthode décrite dans <HAM78> est d'analyser l'effet de l'opération de mise à jour sur chaque assertion. Cette approche est appliquée dans le cas où les opérations sur la base et les assertions sont prédéfinies a priori. La reprise en cas de violation de l'assertion coûte cher, c'est pourquoi HAMMER s'est

orienté vers une méthode où la validation s'effectue avant l'exécution de la transaction.

Pour celà, HAMMER a défini un processeur qui effectue une analyse perturbatrice de l'assertion par la transaction (voir exemple suivant). Celui-ci génère plusieurs tests candidats et le plus efficace est choisi.

Exemple :

Considérons l'assertion : "Pour chaque employé, le salaire doit être plus grand que 1/10 du budget du département où il travaille.

Ce qui se traduit par :
e ∈ EMPLOYE
d ∈ DEPT :
(e.nodept = d.nodept) -> (e.sal > 0.1 * d.budget)

Le processeur génère l'ensemble erreur ("error-set") qui est le complément logique de l'assertion :

e ∈ EMPLOYE
d ∈ DEPT :
(e.nodept = d.nodept) and (e.sal <= 01. * d.budget)

Supposons maintenant qu'on veut effectuer l'opération de mise à jour suivante : "mettre à jour le salaire de l'employé e1 à la valeur S". le processeur génère 2 tests T1 et T2 tels que :

T1 : e ∈ EMPLOYE, d ∈ DEPT : (e = e1) ->
(e1.nodept = d.nodept) and (S <= 0.1 * d.budget)

et

T2 : e ∈ EMPLOYE, d ∈ DEPT : ((e=e1) -> (S > e1.sal)) ->
(e1.nodept=d.nodept) and (S <= 0.1 * d.budget)

Le test le plus efficace est T2, car si le salaire de e1 augmente par exemple on peut déduire qu'on ne peut violer l'assertion donc on s'arrête de tester.

6.2.5 LES METHODES OPTIMISEES -

Deux méthodes basées sur le principe de réduction du temps de vérification des CIS ont été proposées. Pour ce faire, elles reposent sur les stratégies suivantes :

_ éliminer les CIS qui ne sont pas perturbées par l'opération de mise à jour.

_ éviter la reprise en cas de violation de la CIS, c'est-à-dire n'exécuter l'opération de mise à jour qu'une fois qu'on a déterminé que l'ensemble des données restent dans un état cohérent après mise à jour. Pour cela, on cherche à construire des pré-tests permettant de définir l'effet d'une mise à jour sur une CIS et de l'évaluer.

_ tirer profit du fait que la CIS est satisfaite avant mise à jour.

La première est de NICOLAS <NIC79>. Une démonstration formelle de la validité de la méthode est effectuée dans le cadre de la logique des prédicats. En simplifiant, elle consiste à remplacer certaines variables de la CIS par les valeurs mises à jour tout en préservant une certaine équivalence. L'un des points forts de la méthode, réside dans le fait que la CIS peut être une expression quelconque du calcul des prédicats.

La seconde, rapportée dans <BLA81>, <BER81> et <BER82> est une des plus complète à ce jour. Elle du même type que celle de NICOLAS, à ceci près :

_ Dans cette méthode, certaines assertions ne sont pas testées car elles ne sont pas perturbées par l'opération de mise à jour (tests dits triviaux).

_ Une optimisation est effectuée à la vérification de la CIS en s'appuyant sur la maintenance de certaines informations redondantes pour les assertions avec les fonctions agrégats.

_ La classe des CIS vérifiées est plus restreinte dans ce cas.

6.3 NOTRE METHODE DE VALIDATION

6.3.1 INTRODUCTION -

Généralement, on peut valider une requête à trois moments différents :

_ soit à la compilation de la requête, dans ce cas la requête n'est autorisée à s'exécuter qu'une fois toutes les assertions évaluées et ayant un résultat vrai.

_ soit à l'exécution de celle-ci, ce qui veut dire que la validation de la requête est concurrente à son exécution. Par exemple, dans le cas d'une modification, à chaque modification de tuple en mémoire on teste la contrainte.

_ ou après exécution, c'est à dire exécuter la requête d'abord et la valider ensuite. Ce qui nécessite une reprise en cas de violation des assertions.

Dans ‹BPO79›, Badal a montré que le coût de la méthode de validation à l'exécution est minimum par rapport aux deux autres méthodes dans le cas d'une base de données réelle. C'est pour celà que la méthode présentée ici se fonde sur cette conclusion.

6.3.2 EVALUATION CONDITIONNELLE DES ASSERTIONS -

Pour optimiser le coût de validation lors de l'évaluation des assertions, certaines ne seront évaluées que si elles vérifient des conditions particulières. C'est une des stratégies des méthodes optimisées déja présentées.

Exemple : "Les salaires des employés du département 25 sont tous >= 7000F"
La condition est la suivante : (dept = 25) -> (sal >= 7000).

Elle s'exprime ainsi en ISIS :

```
ASSERT CIS1 ON EMPLOYE : (dept <> 25) OR (sal >= 7000).
```

Evidemment cette assertion ne concerne que les employés du département 25. Donc on ne testera cette contrainte que si la première condition est satisfaite (dept = 25).

Ce que l'on vient de voir ne concerne que les assertions sans agrégats, ni blocs SELECT. Considérons maintenant l'exemple suivant, " Le maximum des salaires du département 25 est <= 7000F".

Elle s'exprime ainsi en ISIS :

```
ASSERT CIS2 ON EMPLOYE : ( SELECT MAX(sal)
                           FROM   EMPLOYE
                           WHERE  DEPT = 25 ) <= 7000.
```

De même cette assertion ne sera testée que si l'employé dont le salaire est mis à jour, vérifie la condition "dept = 25", c'est la condition de l'opérateur de sélection de l'arbre engendré pour cette assertion.

Regardons maintenant, les conditions qui portent directement sur les valeurs des expressions arguments des comparaisons. Pour celà prenons l'assertion suivante :

expr1 < expr2.

Si expr1 et expr2 sont des expressions très complexes, on ne testera pas cette assertion dans les cas suivants :

- C1 : expr1 diminue
- C2 : expr2 augmente
- C3 : expr1 diminue et expr2 augmente
- C4 : expr1 et expr2 diminuent ou augmentent de la même valeur.

On en déduit que si (C1 ou C2 ou C3 ou C4) est vrai pour une requête donnée, alors cette requête ne peut pas violer l'assertion.

6.3.3 VALIDATION DES OPERATIONS DE MISE A JOUR -

6.3.3.1 INTERFERENCE OPERATION-ASSERTION -

On ne testera pas chaque opération de mise à jour sur toutes les CIS portant sur la même relation. Il faudrait tenir compte de la sémantique de l'opération et de l'ensemble des objets mis à jour.

Exemple : l'insertion ne modifiant pas de valeurs ne porte pas sur les assertions dynamiques puisque ces contraintes s'intéressent aux transitions des valeurs.

Tableau interférence opération-types d'assertions mono-relationnelles

	statique sans agrégat	statique avec agrégat	dynamique sans agrégat	dynamique avec agrégat
INSERTION	x	x		
MODIFICATION	x	x	x	x
DESTRUCTION		x		x

6.3.3.2 EVALUATION DES CIS -

Il existe deux types de contraintes si on s'intéresse au coût de vérification de celles-ci : les contraintes sans agrégats (ss et ds) et, les contraintes avec agrégats (sa et da). Les dernières coûtent généralement plus cher en temps d'évaluation car la mise à jour d'un attribut nécessite le calcul de la fonction agrégat donc une lecture de l'ensemble des tuples sur lesquelles porte celle-ci.

Pour pallier à cet inconvénient, nous avons utilisé une méthode qui permet de maintenir constamment à jour les fonctions agrégats au fur et à mesure de l'évolution des données (à chaque maj d'un tuple). La valeur de la fonction agrégat (VALAG) est stockée dans le catalogue INTEGRITE à la case associée à la CIS correspondante. Nous nous posons maintenant 2 questions : à quel moment mettre à jour VALAG ? et comment ?

Pour la 1ère question, reprenons l'exemple 2 du paragraphe 6.3.2. La fonction agrégat MAX est mise à jour chaque fois qu'un tuple de la relation EMPLOYE est mis à jour, à la condition que ce tuple vérifie dept = 25 (condition associée à l'opérateur algébrique de sélection).

Nous déduisons tout simplement, que la mise à jour d'une fonction agrégat se fait chaque fois qu'un tuple de la relation correspondante à la CIS est mis à jour, et que ce tuple vérifie la condition de l'opérateur de sélection.

Pour la seconde, le tableau suivant donne pour chaque opération de mise à jour la formule de modification de VALAG (appelée ici Vg).

Tableau de mise à jour des fonctions agrégats suivant les différentes requêtes (I,M ou D).

Vg : valeur de l'agrégat
Vo : valeur de l'attribut avant mise à jour
Vn : valeur de l'attribut après mise à jour
Co : cardinalité de la relation avant mise à jour
Cn : cardinalité après mise à jour.
C'n : cardinalité de la relation intermédiaire juste en dessous de l'opérateur MOY.

Note : Cn = Co + 1 en insertion
= Co - 1 en destruction
= Co en modification

	INSERTION	MODIFICATION	DESTRUCTION
SOM	Vg:=Vg+Vn	Vg:=Vg-Vo+Vn	Vg:=Vg-Vo
MOY	Vg := ((Vg*Co)+Vn)/C'n	Vg := Vg+(Vn-Vo)/Co	Vg := ((Vg*Co)-Vn)/C'n
MAX	si Vg < Vn alors Vg:=Vn	si Vg < Vn alors Vg:=Vn	si Vg = Vn alors calcul Vg
MIN	si Vg > Vn alors Vg:=Vn	si Vg > Vn alors Vg:=Vn	si Vg = Vn alors calcul Vg
CARD	Vg:=Vg+1	Vg:=Vg	Vg:=Vg-1

6.3.3.3 GENERATION DU TEST ASSOCIE A LA CIS -

La validation s'effectue sur chaque tuple mis à jour. Nous allons voir ici comment se fait la génération du test pour chaque opération de mise à jour et pour chaque type de CIS. A partir de l'arborescence de la CIS, et de l'opération de mise à jour on produit l'arborescence "test". Si ce test donne un résultat vrai lors de son évaluation alors la CIS n'est pas violée.

INSERTION :

_ CIS du type "ss" :

. substitution des constantes aux attributs correspondants dans le tuple.

_ CIS du type "sa" :

. substitution des constantes aux attributs.

. calcul de la valeur agrégat (valag).

. substitution de valag courant à valag ancien.

_ CIS du type "ds" et "da" :

. pas de test (l'insertion ne peut pas violer ce type de CIS).

MODIFICATION :

_ CIS du type "ss" et "sa" :

. comme en insertion

_ CIS du type "ds" :

. substitution des nouvelles valeurs aux attributs avec le mot clé NEW.

. substitution des anciennes valeurs aux attributs avec le mot clé OLD.

_ CIS du type "da" :

. évaluer les opérateurs algébriques en substituant les attributs avec NEW (respectivement OLD) à la nouvelle valeur (respectivement ancienne).

. calculer la valeur des fonctions

. substituer les fonctions à leur valeur dans l'arbre CIS.

DESTRUCTION :

_ CIS du type "ss" et "ds" :

. pas de test (CIS non violée).

_ CIS du type "sa" :

. comme en insertion

_ CIS du type "da" :

. comme en modification (sauf qu'il ne faut pas compter les anciennes valeurs).

6.3.4 OPTIMISATION DE L'EVALUATION DES CONTRAINTES -

Un des problèmes du contrôle de l'intégrité sémantique réside dans le coût de l'évaluation des contraintes. Pour celà ISIS optimise en 4 points cette évaluation :

_ D'abord toutes les assertions sont traduites en arborescences algébriques pour des facilités de manipulation et d'évaluation.

_ L'évaluation de certaines assertions complexes ne se fera que sous certaines conditions.

_ L'évaluation des contraintes sera concurrente à l'exécution de la requête. Une étude ‹BPO79› a montré que le coût est moindre dans ce cas.

_ Enfin pour évaluer les contraintes avec agrégats, il n'est pas nécessaire, dans la plupart des cas, de recalculer la fonction agrégat. De plus les attributs sont remplacés par des constantes. Intuitivement, plus il y a d'attributs remplacés par des constantes, plus le test est "sélectif", donc plus facile à évaluer.

6.3.5 CONTRAINTES SUR LES CLES -

Il existe 3 types de contraintes sur les clés :

_ unicité de la clé primaire d'une relation,
_ la valeur d'une clé ne peut pas être indéfinie et
_ la valeur d'une clé ne peut pas être modifiée.

Ces trois contraintes du type structurel ne sont pas incluses dans le catalogue mais sont testées à chaque fois qu'une clé est modifiée ou insérée. Ce test sera effectué grâce à la structure B*-arbre intégrée dans MIMER (Cf. chapitre 7). Comment effectuer ce contrôle ?

Chaque fois qu'on définit un attribut comme une clé dans une relation, on lui associe un arbre B*-index qui facilitera l'accès aux

tuples de la relation. Il existe deux primitives qui modifient ce B*-index : INSERER-CLE et SUPPRIMER-CLE. Par l'intermédiare de ces deux primitives on teste les deux premières contraintes ci-dessus. La modification de clé est refusée systématiquement, car il n'existe pas de primitive MODIFIER-CLE.

6.3.6 CONCLUSION ET COMPARAISON -

Notre méthode se place sur la même ligne que les méthodes optimisées, mais se place dans le contexte SGBD relationnel sur micro-ordinateur. Comme dans la méthode de BLAUSTEIN <BLA81>, cette optimisation s'appuie sur la maintenance d'informations redondantes qui ne sont autres que les valeurs des fonctions agrégats. Sans cela, le coût de validation risque d'être très élevé.

Néanmoins, les CIS traitées dans notre premier prototype sont mono-relationnelles dynamiques ou statiques.

L'une des particularités de la méthode réside dans le fait qu'on traite les CIS sur les clés via la méthode d'accès B*-arbre.

6.4 TEST DES ASSERTIONS

En cas d'erreur logicielle et/ou matérielle (erreur d'enregistrement, de lecture, ...) l'utilisateur voudra tester si les contraintes existantes sur la base ne sont pas violées. Evidemment cela ne concerne que les contraintes statiques donc du type "ss" ou "sa". Pour celà l'utilisateur (administrateur de la base) dispose de la commande TESTCIS. Par l'intermédiaire de cette commande on peut tester de façon sélective des contraintes d'intégrité. On accède au catalogue pour sélectionner les contraintes et les relations mises en cause. Ensuite on effectue l'évaluation de ces contraintes sur les données de la base après reprise. Dans le cas où une contrainte est violée alors les données de la relation ne sont pas cohérentes, une méthode de reprise par l'intermédiaire du journal est nécessaire.

6.5 LES POSSIBILITES D'EXTENSION D'ISIS

Bien que l'étude du sous-système se soit limitée aux CIS mono-relationnelles, il est possible de l'étendre aux multi-relationnelles. Au niveau de l'expression et de la traduction

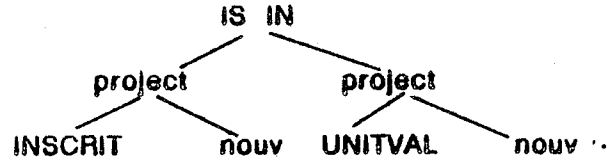
en arborescence. le problème est déja résolu puisque MIQUEL peut exprimer des CIS multi-relationnelles. de même la méthode de traduction est prévue pour des CIS multi-relationnelles. Il subsiste un problème au niveau de la validation des opérations de mise à jour.

Quelques cas peuvent se résoudre sans difficultés.

Exemple 1 : Reprenons l'exemple sur l'intégrité référentielle

```
( SELECT    nouv
  FROM      INSCRIT  )  IS IN ( SELECT nouv
                                FROM   UNITVAL )
```

Nous aboutissons à l'arborescence suivante :



Cette CIS peut être remise en cause dans 4 cas :

a) insertion d'un tuple dans INSCRIT
b) destruction d'un tuple de UNITVAL
c) modification de l'attribut "nouv" de INSCRIT
d) modification de l'attribut "nouv" de UNITVAL

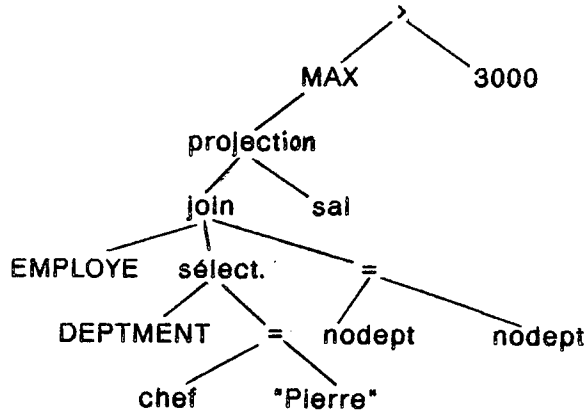
Par contre la CIS n'est pas touchée dans 2 cas :

e) insertion d'un tuple dans UNITVAL
f) destruction d'un tuple de INSCRIT

Ce type de CIS (R1 IS IN R2) est basée sur un opérateur ensembliste "IS IN" (opérateur d'inclusion). Il n'est pas nécessaire de montrer chaque fois qu'une relation est modifiée. que R1 est inclus dans R2. Mais il faut utiliser le fait que l'inclusion est vérifiée l'état d'avant. Il suffit de montrer. par exemple dans le cas d'une insertion dans R1. que ce tuple appartient à R2 et seulement ce tuple.

Exemple 2 : Considérons la CIS avec 2 relations : "le maximum des salaires des employés du département ayant comme chef Pierre est supérieur à 3000".

On obtient l'arborescence suivante :

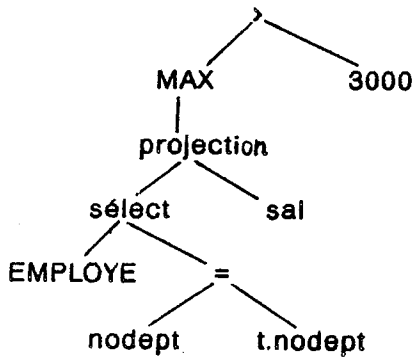


Etudions l'incidence de l'insertion d'un tuple dans EMPLOYE ou dans DEPTMENT.

a) Insertion d'un tuple t dans DEPTMENT :

. si ce tuple ne vérifie pas la condition t.chef = "Pierre" alors la CIS n'est pas violée.

. si la condition t.chef = "Pierre" est vérifiée alors il faut tester la CIS suivante :



Ces 2 cas peuvent être regroupés en un seul test qui est :

```
( t.chef ≠ "Pierre" )  or ( ( SELECT MAX (sal)
                              FROM   EMPLOYE
                              WHERE  nodept = t.nodept)  > 3000 )
```

b) Insertion d'un tuple e dans EMPLOYE :

Il faut tester la condition (c) suivante :

```
( SELECT chef
  FROM   DEPTMENT
  WHERE nodept = e.nodept ) IS IN ("Pierre").
```

. si la condition (c) est vérifiée alors tester e.sal > 3000

. sinon la CIS n'est pas violée.

Comme le cas précédent les 2 tests peuvent se réduire en un seul :

```
( ( SELECT chef
    FROM   DEPTMENT
    WHERE  nodept = e.nodept ) IS NOT IN ("Pierre") )
 or
    ( e.sal > 3000 )
```

Ces 2 exemples montrent que des transformations algébriques sont nécessaires, pour produire un test associé à chaque opération de mise à jour et à chaque relation mise à jour. Remarquons que les opérateurs qui manipulent les tuples de la relation mise à jour sont supprimés de la CIS.

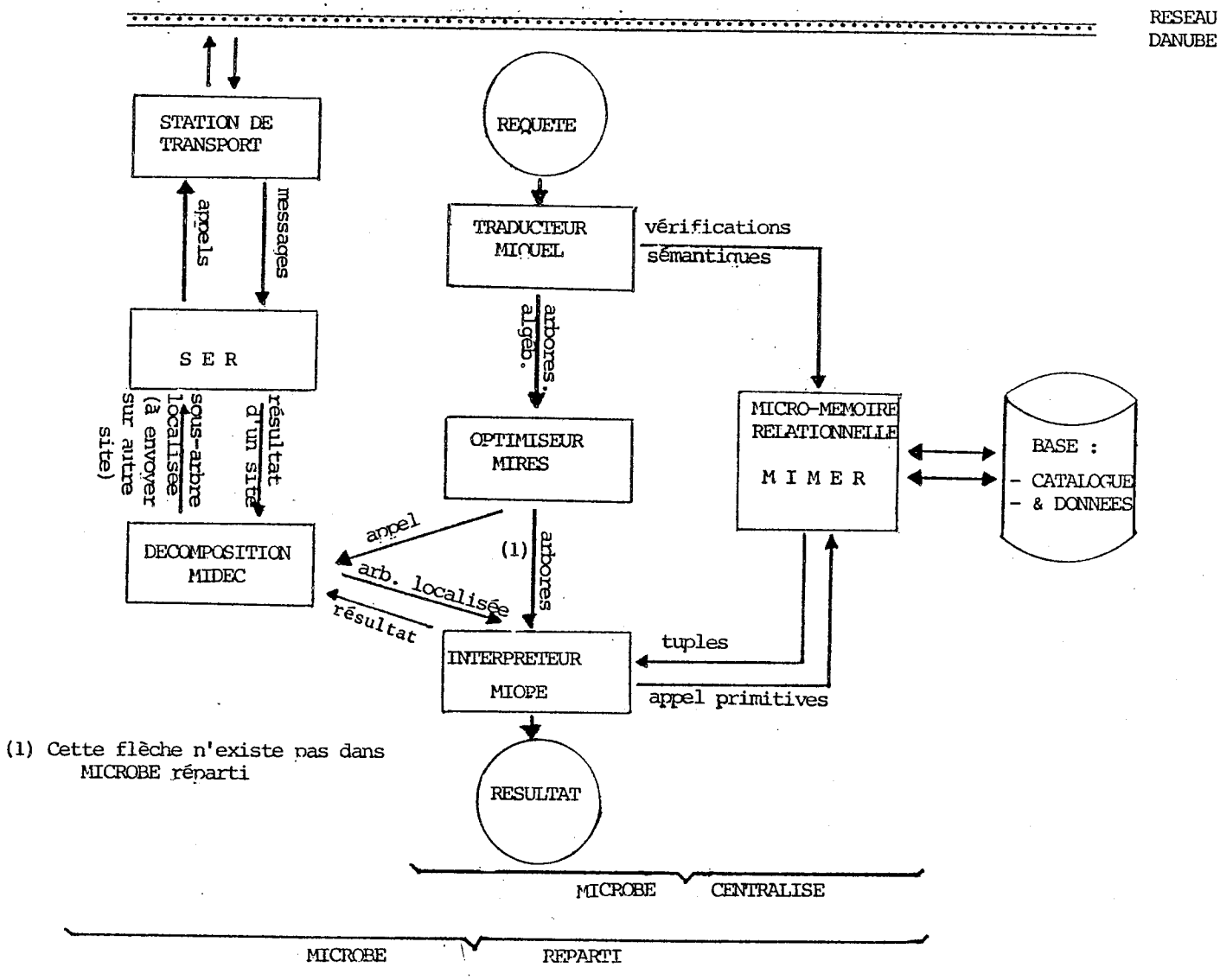
· Les 2 exemples de CIS que nous venons d'analyser s'intéressent à 2 relations. Nous pensons que le problème se compliquerait dans le cas à plusieurs relations. Mais une étude dans la voie des transformations algébriques combinées avec la redondance des fonctions agrégats, peut être riche de retombées.

7.0 PRESENTATION DU PROJET MICROBE

7.1 INTRODUCTION

Le projet MICROBE a pour objectif la conception et la réalisation d'un SGBD relationnel, réparti sur le réseau local DANUBE <NAF79>. Une version centralisée de MICROBE tourne depuis Juin 1981. La version répartie nécessite la mise en oeuvre d'un algorithme de décomposition de requêtes et d'un système d'exécution répartie. Nous présenterons d'abord MICROBE centralisé composé de quatre modules : Le traducteur de requêtes MIQUEL, l'optimiseur MIRES, le module d'évaluation des opérateurs algébriques MIOPE et enfin la micro-mémoire relationnelle MIMER. Ensuite nous exposerons les deux modules, le système d'exécution répartie SER et la décomposition des requêtes MIDEC qui associés aux premiers forment la version répartie.

7.2 ARCHITECTURE FONCTIONNELLE



(1) Cette flèche n'existe pas dans MICROBE réparti

MICROBE est implanté en architecture modulaire. Nous allons voir brièvement les fonctionnalités de chaque module. Une requête soumise par un utilisateur est analysée syntaxiquement et sémantiquement par MIQUEL. Celui-ci produit une arborescence algébrique associée à cette requête ‹FRT80›. Cette arborescence est alors modifiée à l'aide des règles algébriques de l'optimiseur MIRES ‹WIN82› qui garantissent une évaluation "optimale" de la requête. Ensuite elle est interprétée par le module MIOPE ‹LEE81› via des primitives MIMER et le résultat est communiqué à l'utilisateur.

7.3 MICROBE CENTRALISE

7.3.1 L'INTERFACE MIQUEL -

Toute requête MIQUEL est traduite en arborescence algébrique. L'intérêt de cette approche est de produire une arborescence qui peut être ensuite optimisée et décomposée de façon dynamique en sous-arbres locaux. Ceci est détaillé dans les paragraphes suivants.

Nous allons présenter d'abord l'ensemble des commandes utilisateur disponibles dans MIQUEL, ensuite nous présenterons brièvement le module traducteur de requêtes.

Interface utilisateur :

L'utilisateur accède à MICROBE par cette interface. Il dispose des commandes suivantes :

- Interrogation de la base : SELECT
  - . l'utilisateur peut selectionner des données de la base par cette commande. Les requêtes très complexes nécessitent souvent des imbrications de blocs SELECT.
- Mise à jour de la base :
  - . INSERT_INTO : insertion de nouveaux tuples dans une relation.
  - . MODIFY : modification de la valeur d'un attribut donné.
  - . DELETE_FROM : destruction de tuples

déja existants.

_ Manipulation des structures

. CREATE_RELATION : Création dynamique de relation.

. KILL_RELATION : Destruction dynamique de relation.

. CREATE_ATTRIBUT : Création dynamique d'attribut.

. KILL_ATTRIBUT : Destruction dynamique d'attribut.

. CREATE_INDEX : Création d'un index sur un ou plusieurs attributs.

_ Manipulation de la base

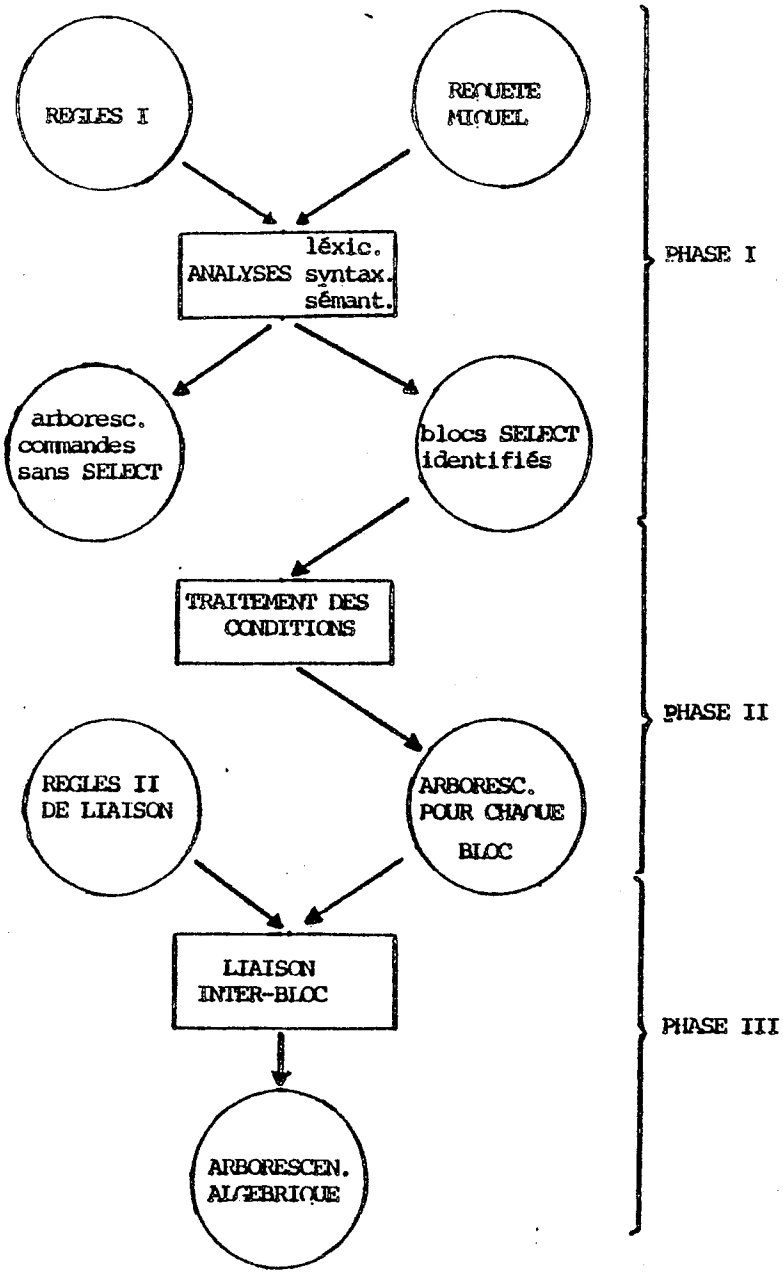
. On peut Créer, détruire, ouvrir et fermer une base de données.

_ Information sur la base

. On peut lister les relations, les attributs et leurs caractéristiques.

L'ensemble des commandes MIQUEL peuvent être classées en 2 types, celles qui contiennent des blocs SELECT (SELECT, MODIFY, DELETE_FROM et CREATE_RELATION) et les autres.

Le traducteur MIQUEL :



ORGANIGRAMME FONCTIONNEL DE MIQUEL

Le traducteur implémenté dans MICROBE fonctionne en 3 phases :

_ La 1ère phase, effectue les analyses lexicale, syntaxique et sémantique (existence des relations et attributs) de la requête et construit, une arborescence finale pour le second type de commandes, et des sous-arbres associés à chaque bloc via les règles I (Cf. chapitre 4) pour le premier type.

_ La 2ème phase, produit une arborescence pour chaque condition de bloc SELECT déjà identifié en phase 1 tout en maintenant les liaisons inter-bloc existantes.

_ La 3ème phase utilise en entrée les arborescences de la phase 2 et l'ensemble des règles II (Cf. chapitre 4) pour effectuer les liaisons inter-bloc en insèrant des opérateurs de liaison tels que join, union et intersection. Cette phase est présentée en détails au chapitre 4.

7.3.2 L'OPTIMISEUR D'ARBORESCENCES (MIRES) -

Une fois la requête traduite par MIQUEL, l'arborescence produite est prise en compte par MIRES qui effectue une optimisation algébrique via des règles prédéfinies basées sur les opérateurs de l'algèbre relationnelle.

Il existe plusieurs transformations algébriques possibles, mais MIRES se limite aux règles les plus plausibles : MICROBE est implémenté sur micro-ordinateurs, un optimiseur complet, loin d'améliorer les performances, les alourdirait de par sa complexité.

Parmi les transformations possibles on peut citer :

_ la distribution des opérateurs de sélection et de projection <WIN82> (ce sont ces transformations qui sont implémentées dans MIRES),

_ la réduction des sous-arbres homogènes,

_ l'élimination des relations vides,

_ et l'application des règles d'idempotence.

MIRES travaille en 3 phases dont chacune effectue un parcours récursif de l'arbre correspondant à la requête.

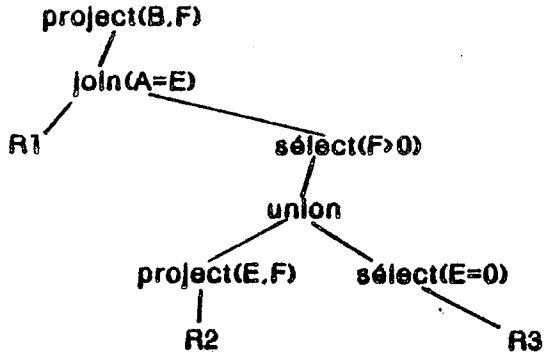
_ La première détermine pour chaque noeud (opérateur ou relation) les attributs qui composent la relation (produite dans le cas d'un opérateur),

_ La seconde, effectue la distribution de la sélection sur la projection, l'union, l'intersection ou la différence et regroupe 2 sélections consécutives.

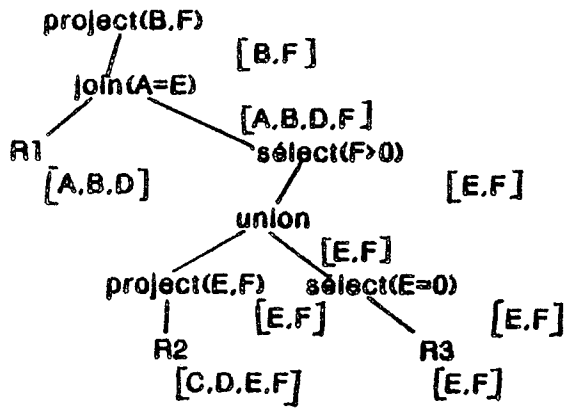
_ La dernière, effectue la distribution de la projection sur la sélection, l'union, la différence, le produit cartésien ou la jointure et regroupe 2 projections consécutives en éliminant une projection si ses attributs sont les mêmes que ceux de son fils.

Pour illustrer l'ensemble de ces 3 phases, procédons par un exemple, soit la base suivante composée de 3 relations R1 (A,B,D), R2 (C,D,E,F) et R3 (E,F).

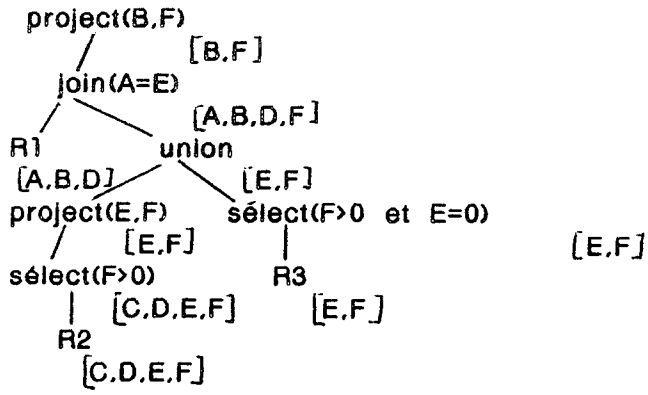
Soit l'arborescence suivante,



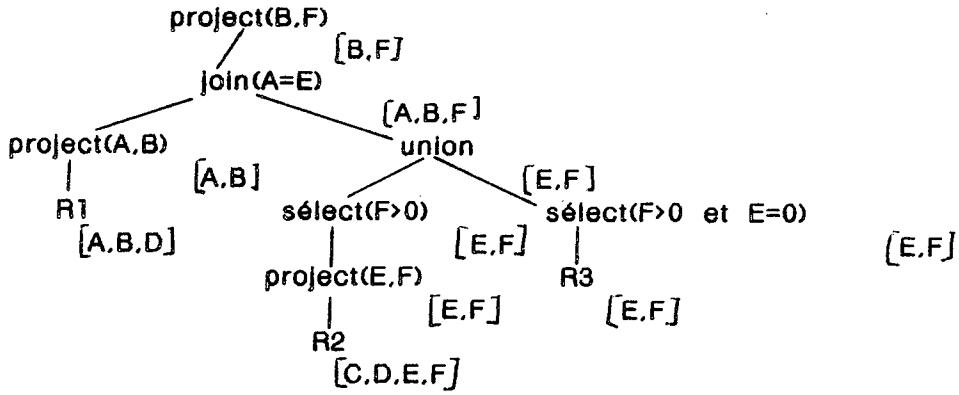
La première phase détermine les attributs pour chaque opérateur ou relation noeud :



Passons maintenant à la phase 2. Il faut distribuer la sélection sur l'union et regrouper 2 sélections. On obtient l'arbre suivant :

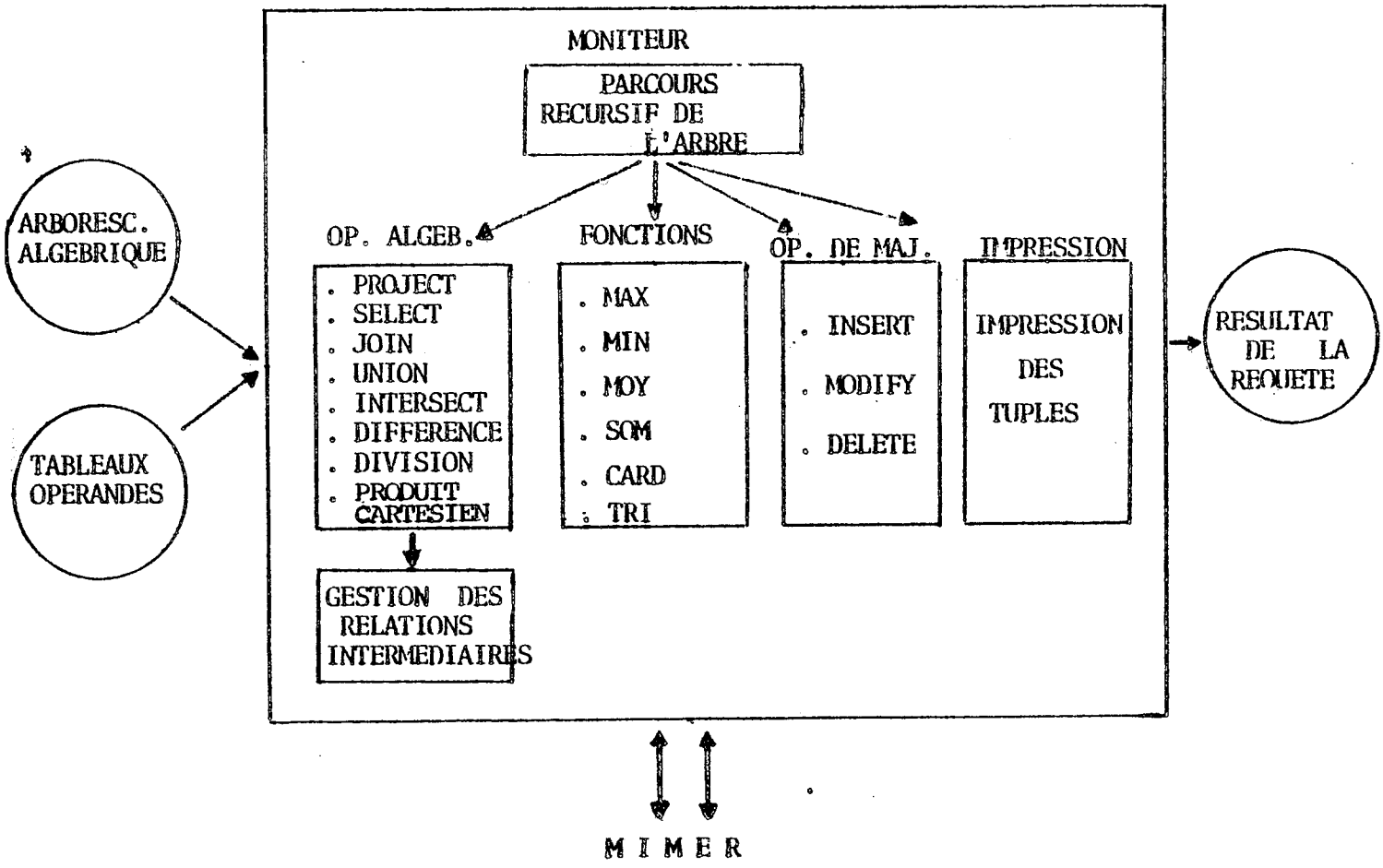


Appliquons enfin la phase 3. C'est la distribution de la projection sur la jointure puis sur la sélection, on obtient alors l'arbre suivant :



7.3.3 L'EVALUATION DES OPERATEURS ALGEBRIQUES (MIOPE) -

MIOPE a pour objectif l'évaluation de l'arborescences optimisée par MIRES et le renvoi de résultats à l'utilisateur. Cette évaluation se fait en mode "pipe-line", c'est à dire qu'à chaque fois qu'un tuple est retenu en un noeud opérateur donné, il est directement envoyé au noeud père. Cette technique a deux avantages : elle évite le stockage de relations temporaires (cas des opérateurs unaires) et minimise le temps d'attente du premier tuple. De plus, elle permet d'évaluer de façon parallèle des sous-arbres en environnement réparti.



ARCHITECTURE DE MIOPE

Pour évaluer l'arbre, l'interpréteur fait un parcours récursif et à chaque opérateur il appelle la procédure adéquate. Dans le cas d'un opérateur unaire, la procédure prend en compte le tuple émis par l'opérateur fils et l'envoie au noeud père dans le cas où il est retenu à ce niveau là, sinon (cas d'un opérateur binaire) la procédure se met en attente au moins jusqu'à la fin de l'exécution d'un des opérateurs fils (cas du "join" par exemple).

Dans MIOPE plusieurs opérateurs sont implémentés, on peut citer : les opérateurs ensemblistes UNION, INTERSECTION, DIFFERENCE et PRODUIT CARTESIEN, les opérateurs de manipulation de relation n-aires PROJECTION, SELECTION, JOIN, DIVISION et l'ensemble des fonctions agrégats et de tri.

7.3.4 LA MICRO-MEMOIRE RELATIONNELLE (MIMER) -

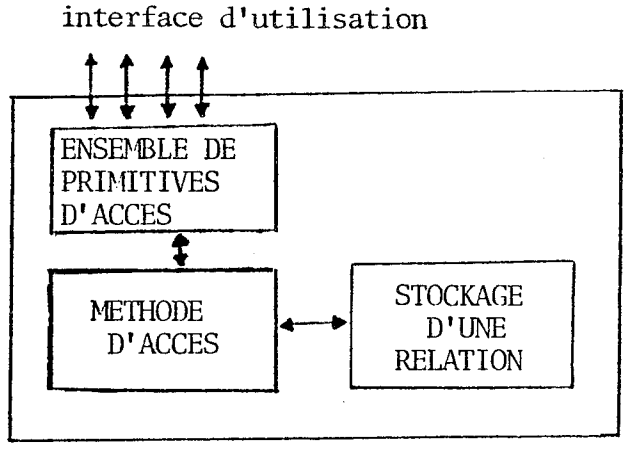
MIMER (Micro-MEmoire Relationnelle) <FER81> est un logiciel permettant :

_ une "vision relationnelle" du fichier physique de la base de données,

_ une gestion de pages de données par un mécanisme de mémoire virtuelle,

_ une gestion de différents types de relations de la base de données (relations internes, inverses et de base),

_ l'établissement de méthodes d'accès (séquentiel, B*-arbre, ...).



ARCHITECTURE FONCTIONNELLE DE MIMER

Les catalogues :

La base de données est gérée par un ensemble de catalogues qui sont des relations internes. Il en existe quatre. Ils sont gérés directement par MIMER. Ils décrivent toutes les relations de la base et établissent une liaison avec la mémoire secondaire :

1) La relation MAITRE : chaque tuple est un descripteur de l'une des relations de la base. Ce descripteur contient les informations suivantes : le nom externe, le type, la cardinalité, le degré, la longueur d'un tuple, le type de stockage, et des liens vers les relations INDEX et ATTRIBUTS, et l'adresse de la première et de la dernière page de la relation. Dans le cas d'un SGBD distribué, on rajoute le site de localisation de la relation. (tous les tuples d'une même relation sont sur un même site).

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

3) La relation INDEX : ici un tuple décrit une relation inverse créée sur une relation de base.

4) La relation ATTRIBUT-CLE : elle détermine l'ensemble des attributs constituant les clés des relations inverses.

Types des relations :

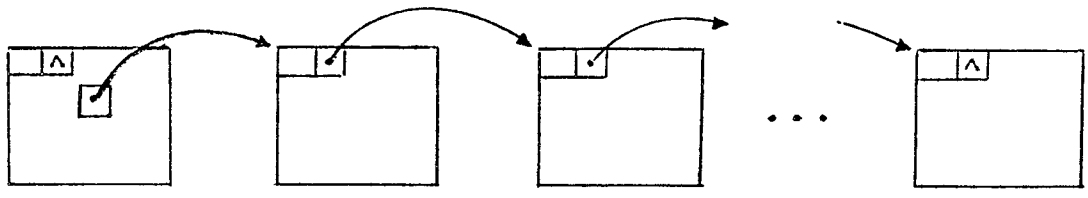
Il existe 3 types de relations dans MIMER :

- les relations internes qui ne sont autres que les catalogues.

- les relations de base qui contiennent les données des utilisateurs.

- et les relations inverses qui sont des index sur les relations de base, ceci permet un accès rapide aux données des relations de base suivant un adressage associatif. Logiquement, une relation inverse contient 2 attributs : l'attribut CLE qui n'est autre que la clé primaire ou secondaire de la même relation et l'attribut POINTEUR qui stocke les Idt (adresse du type page et déplacement) des tuples correspondants dans la relation de base pointée.

Liste d'espace disponible :



Descripteur de la B.D.

Pour stocker les données, nous utilisons un support physique (disque magnétique) permettant l'accès direct. Physiquement le fichier est constitué d'un certain nombre de blocs qui constituent un espace d'adressage. Chaque bloc est identifié par une adresse et correspond à un enregistrement qui est accédé et recopié en mémoire centrale par la méthode d'accès direct aux fichiers.

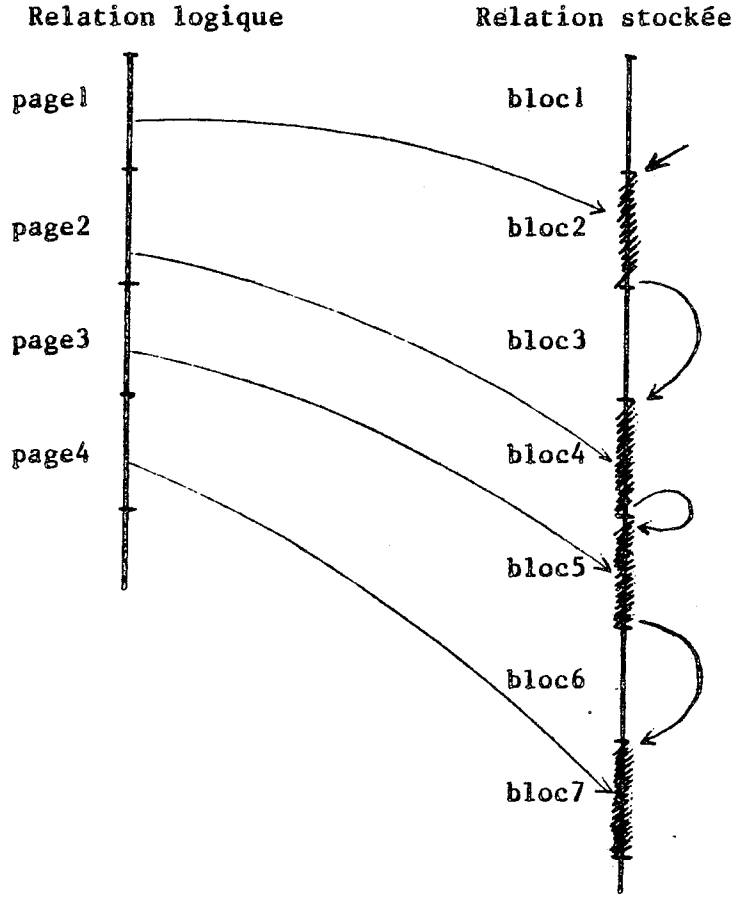
Le premier bloc du fichier contient l'identification de la base de données, c'est-à-dire son nom, le nom de l'administrateur, le nom logique du disque, le code d'exploitation et sa taille en blocs. De plus ce bloc contient un pointeur vers le premier bloc de la liste d'espace disponible.

Cette liste est constituée par la chaîne de blocs libres de la base. Celle-ci est gérée avec une politique LIFO ("Last In First Out"). A la création de la base de données tous les blocs appartiennent à cette liste.

Stockage d'une relation :

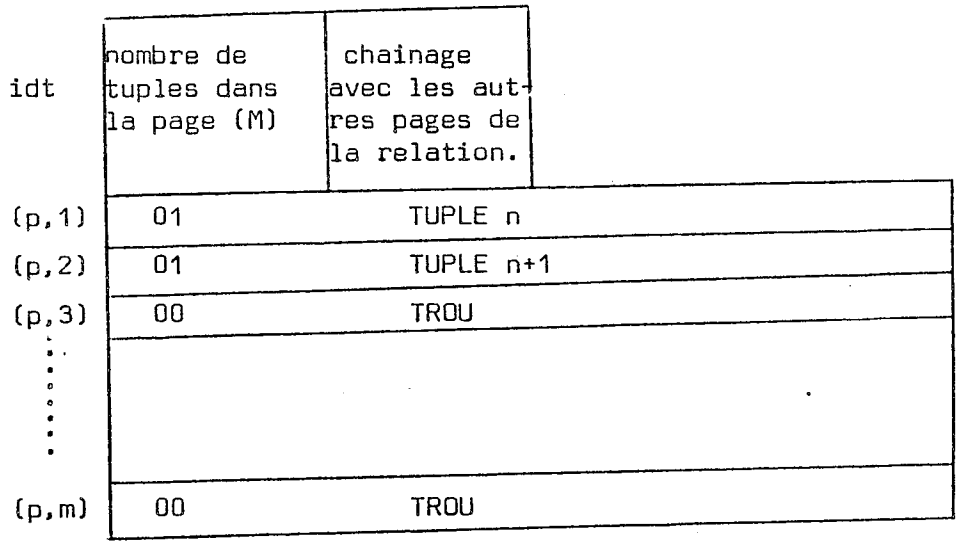
Une relation est découpée en pages logiques de taille fixe, chaque page correspond à un bloc en mémoire secondaire.

Chaque tuple d'une relation est distingué des autres par un idt (identificateur de tuple) qui correspond au couple (p,d) où p est l'adresse de la page et d le déplacement par rapport au début de cette page.



les blocs physiques ne sont pas forcément séquentiels sur le disque mais un chainage permet de retrouver tous les blocs associés à une relation.

Chaque page physique (ou bloc) a le format suivant :



Format de la page d'adresse P d'une relation.

Au début de la page 4 octets sont réservés pour :

_ le nombre maximum de tuples dans la page dans les 2 premiers octets,

_ et le chainage avec la page suivante (si elle existe) dans les 2 octets suivants.

Les tuples sont stockés les uns après les autres. Chacun est précédé d'un témoin d'existence qui aura la valeur '00' s'il s'agit d'une place disponible ("trou") et '10' si la place est effectivement occupée par un tuple.

Un tuple ne peut pas être fragmenté entre plusieurs pages. Donc la taille d'un tuple est limitée à la taille d'une page.

La mémoire virtuelle :

Les pages constituant la base de données sont amenées en mémoire centrale dans une zone de pagination à la demande. Le nombre de pages résidentes en mémoire est un paramètre de génération de MICROBE. Elles sont remplacées dans cette zone avec une politique LRU ("Last Recently Used").

Les méthodes d'accès :

Deux méthodes d'accès ont été implémentées dans MIMER :

_ la méthode séquentielle, qui consiste en un parcours de la relation telle qu'elle est stockée physiquement au moyen d'un balayage simple (de page en page).

_ les B*-arbres :

Un B*-arbre d'ordre d et de profondeur p est défini comme une arborescence ayant les propriétés suivantes :

i) chaque noeud a au plus d fils.

ii) chaque noeud, excepté la racine et les feuilles, a au moins [d/2] fils.

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

iv) toutes les feuilles apparaissent au même niveau (soit p ce niveau ici).

v) un noeud ayant k fils (k<=d) contient k-1 clés.

Un B*-arbre permet d'accéder un tuple d'une relation dont on connait la clé par une méthode de recherche de la racine du B*-arbre jusqu'à la feuille tuple. Un B*-arbre est stocké sous la forme de 2 relations qui sont traitées comme n'importe quelle relation de la base de données.

Deux types d'accès sont fournis par les B*-arbres :

_ l'accès séquentiel ordonné de toute la relation utilisateur grâce à un parcours séquentiel du B*-arbre qui ne sont autres que les tuples de la relation.

_ l'accès associatif, qui consiste à accéder à un tuple d'une relation de base dont la clé est connue sans avoir à parcourir séquentiellement toute la relation.

7.4 MICROBE REPARTI

7.4.1 SYSTEME D'EXECUTION REPARTIE (SER) -

SER (Système d'Exécution Répartie) fait suite aux travaux effectués dans le domaine des applications réparties notamment dans le projet SIRIUS (SIGOR et MERE). Il a été conçu pour répondre en particulier aux besoins des applications de gestion de bases de données réparties. Il présuppose l'existence :
_ d'un médium de communication entre les différents sites fournissant un service de transport classique,
_ sur chacun des sites, un CATALOGUE GENERAL de programmes pouvant intervenir dans la construction de l'application répartie (par exemple, une bibliothèque d'opérateurs relationnels dans le cadre de MICROBE).

A partir des différents catalogues de programmes existants sur les sites connectés, SER fournit les moyens pour :

_ activer un programme à distance (sur un processeur quelconque),

_ enchainer les différents programmes locaux entre eux pour le compte de l'application pour un même utilisateur,

_ assurer les transferts de données nécessaires entre 2 programmes s'exécutant pour le même utilisateur,

_ exprimer sous forme logique les conditions d'activation d'un programme en fonction éventuellement des résultats d'autres programmes,

_ signaler les fins normales ou anormales des programmes.

SER est caractérisé par un contrôle d'exécution entièrement réparti qui s'appuie sur trois entités de base :

_ l'action locale (AL)
_ les variables de synchronisation (VS)
_ et les fichiers de messages temporaires (FMT).

L'Action Locale :

l'action locale est l'unité logique d'exécution de SER. Une application locale est définie par un programme à exécuter pré-existant sur le processeur concerné, des flots de données en entrée et en sortie et des conditions de synchronisation à l'intérieur de l'application.

Une AL correspond physiquement à l'exécution d'un programme sur

un processeur. Ce peut être, par exemple un opérateur relationnel (projection). Les seules actions qu'une AL peut avoir sur l'environnement d'autres AL sont la mise à jour des VS et la consommation ou la production de FMT.

Variable de Synchronisation :

C'est une variable associée à une action locale, qui permet d'assurer le contrôle d'exécution réparti ou d'exprimer les relations reliant dynamiquement à l'exécution plusieurs AL participantes à l'application répartie.

Pour chaque AL, une condition d'activation, définissant une certaine configuration de valeur des VS associées, permet de fixer le moment de son activation. De même, une expression de terminaison permet, sur la fin d'exécution d'une AL, de définir des mises à jour de VS "distantes"; c'est-à-dire associées à d'autres AL du même site ou des sites distants.

Fichiers de Messages Temporaires :

Une AL produit et consomme des FMT. SER gère automatiquement une zone réseau utilisée pour stocker de manière temporaire ces FMT. Ils se présentent comme des fichiers séquentiels à articles de taille variable. Ils sont produits article par article sans possibilité de retour et sont détruits automatiquement après consommation. SER assure le transfert des FMT de l'AL productrice vers l'AL consommatrice en mode pipe-line.

Fonctions de SER :

Sur chaque site on dispose d'un SER local assurant l'activation et le contrôle des AL de cette station pour le compte de l'application répartie. Notons que SER est par lui-même un système réparti.

SER est composé de deux sous-modules : SER-client qui assure la prise en compte de chaque commande de création de contexte d'AL et sa transmission vers le sous-module SER-serveur qui assure sa préparation, son lancement et son contrôle.

7.4.2 DECOMPOSITION DES REQUETES (MIDEC) -

Le problème de la décomposition :

Décomposer une requête, consiste à remplacer cette requête complexe par une séquence de sous-requêtes de complexité moindre en vue d'une optimisation. Ceci s'applique en environnement centralisé, mais en réparti la question est un peu plus délicate, car non seulement il faut décomposer la requête en sous-requêtes, mais il est nécessaire de localiser ces dernières (déterminer leurs sites d'exécution), effectuer et contrôler les transmissions de données inter-sites et finalement regrouper les résultats en vue d'une réponse finale.

Parmi les critères d'optimisation retenus dans la décomposition, on peut citer :

- le temps de réponse d'une requête
- et la charge globale du système.

Ces critères dépendent de plusieurs facteurs :

- ceux propres au réseau (taille des paquets transmis, délai de transmission, ...),
- ceux propres aux machines (temps d'accès disque, ...),
- ceux propres aux données (volume des données, existence d'index, ...),
- ceux propres à la méthode d'évaluation de la requête (pipe-line, ...).

En environnement réparti, supposons que la requête est traduite en arborescence algébrique, alors il existe deux méthodes de décomposition :

- statique : les sites d'exécution des opérateurs sont prédéterminés avant le lancement de la requête.
- dynamique : contrairement au mode statique, les sites d'exécution sont déterminés au fur et à mesure de l'exécution de la requête en fonction principalement des volumes de données à transférer.

Les décomposeurs dynamiques peuvent être centralisés (il existe un site maitre qui supervise l'exécution de la requête) ou décentralisés (ceci consiste à donner le même statut à tous les sites concernés).

Le décomposeur MIDEC :

MIDEC est du type dynamique décentralisé. Il est composé de 3 phases :

_ Phase 1 : c'est la phase de localisation initiale de l'arbre sur le site d'émission de la requête et de son envoi à tous les sites.

_ Phase 2 : au niveau de chaque site, on effectue l'évaluation des opérateurs localisés sur ce site.

_ Phase 3 : au niveau de chaque site, on poursuit la localisation de l'arbre en tenant compte des résultats locaux et distants (sur d'autres sites).

Ces 2 dernières phases s'exécutent en parallèle et se terminent à la localisation et l'exécution de la racine de l'arbre.

MIDEC, pour s'exécuter s'appuie sur SER, par des appels à des ALs en vue des transmissions de données (FMT) ou des informations de contrôle (VS).

7.5 LES EXTENSIONS DE MICROBE

7.5.1 L'INTERFACE GRAPHIQUE (INGRID) -

L'interface INGRID «CHA82» implémenté dans MICROBE s'inspire du langage QBE développé par ZLOFF. Le langage INGRID répond aux trois caractéristiques suivantes : relationnnel, graphique et intéractif.

L'ensemble des commandes peut se résumer ainsi :

_ des commandes de manipulation de la base (ouverture, fermeture, création et destruction),

_ des commandes de manipulation de la structure (création, destruction et affichage des attributs et des relations),

_ des requêtes de manipulation de tuples (interrogation, modification, insertion et destruction).

Le traitement des requêtes dans INGRID se fait comme suit :

_ Une première phase de saisie des informations à l'écran.

_ une seconde phase d'analyses lexicale, syntaxique et sémantique.

_ suivie de l'interprétation via des primitives de la micro-mémoire relationnelle MIMER.

_ et enfin une phase d'affichage. En effet un module graphique a été mis au point dans le but d'une souplesse d'utilisation tant en saisie de la requête qu'à la visualisation de son résultat (programmation de touches spécialisées pour manipulation de squelettes de relations sur un terminal VT100).

7.5.2 CHOIX DES CHEMINS D'ACCES DANS MIMER -

Trois chemins d'accès sont offerts par MIMER, le séquentiel, le séquentiel ordonné suivant l'index et l'associatif; les 2 derniers se font avec les B*-arbres.

Supposons qu'on veut effectuer un balayage conditionnel sur une relation et qu'il existe plusieurs relations inversées dont les clés apparaissent dans la condition de balayage. Nous avons alors, le choix entre plusieurs chemins d'accès. La solution optimale consiste à choisir le chemin dont le coût d'accès global de la requête est minimum. Nous avons opté dans MICROBE, à une minimisation d'accès partiel <LEE83>, c'est à dire réduit à une relation.

Exemple :

Pour illustrer ceci, prenons un exemple. Soit une relation COMMANDE composée des attributs NOMF (nom du fournisseur), NOMP (nom du produit commandé) et QTE (la quantité commandée). La clé de la relation étant le couple (NOMF,NOMP).

On crée 2 relations inverses sur cette relation, un index primaire INDEX1 sur les 2 attributs NOMF et NOMP et un index secondaire INDEX2 sur NOMP.

relation et index

COMMANDE

Idt	nomf	nomp	qte
t1	f1	p1	300
t2	f4	p5	400
t3	f1	p2	200
t4	f4	p4	300
t5	f1	p6	100
t6	f2	p1	300

INDEX1

Idt	X	Y
t7	f1	p1	t1
t8	f1	p2	t3
t9	f1	p6	t5
t10	f2	p1	t6
t11	f4	p4	t4
t12	f4	p5	t2

INDEX2

Idt	Z	T
t13	p1	t1
t14	p1	t6
t15	p2	t3
t16	p4	t4
t17	p5	t2
t18	p6	t5

Question : On s'intéresse à l'ensembles des commandes ayant (nomf = 'f4') et (nomp = 'p5')
On peut répondre à cette question par trois chemins différents :

_ 1er cas : parcours séquentiel de la relation COMMANDE pour sélection.

_ 2ème cas : utilisation de l'INDEX1

_ 3ème cas : utilisation de l'INDEX2 et sélection.

Evidemment le cas 3 est le plus intéressant ici, car le nombre d'accès est minimum via le B*-arbre.

La méthode de coût mise au point ‹LEE83› ne sera pas discutée ici, nous dirons seulement qu'elle tient compte de plusieurs facteurs dont nous citons :

_ pour le cas séquentiel, la longueur du tuple, la taille de la page, la cardinalité de la relation, le coût d'accès à un bloc physique et le nombre de conditions atomiques.

_ avec les B*-arbres (séquentiel ordonné et associatif), il faut rajouter la profondeur de l'arbre, le nombre d'attributs clés, le nombre moyen de clés stockées dans un noeud, ...

7.5.3 SECURITE ET JOURNALISATION -

Un SGBD lors de son fonctionnement est sujet à des défaillances matérielles et logicielles. Des données peuvent être perdues ou mises dans un état incohérent. Pour pallier à cet inconvénient, on apporte des solutions appelées "mécanismes de reprise".

Il existe plusieurs mécanismes pour pallier à la sécurité physique de la base dont "le fichier différentiel". Dans cette méthode, les mises à jour effectuées sur la base sont localisées dans un espace relativement petit, appelé fichier différentiel. De cette façon il est possible de réduire les coûts de reprise et d'augmenter la vitesse de recouvrement de la base. Toute consultation de la base passe par le fichier différentiel pour avoir la plus récente valeur. Quand le fichier est suffisamment long, on effectue une réorganisation de la base et on le remet à "vide".

Quant à la sécurité logique, elle peut être bien assurée par la méthode de journalisation. Dans cette méthode, chaque opération de mise à jour est enregistrée dans le journal. En cas d'incident, il existe 2 opérations qui refont et défont la base en tenant compte du journal.

Dans MICROBE, une étude a été effectuée <PRA82>. Il en ressort que les 2 méthodes s'intègrent bien dans l'architecture de MIMER.

8.0 REALISATION D'ISIS DANS MICROBE

8.1 INTRODUCTION

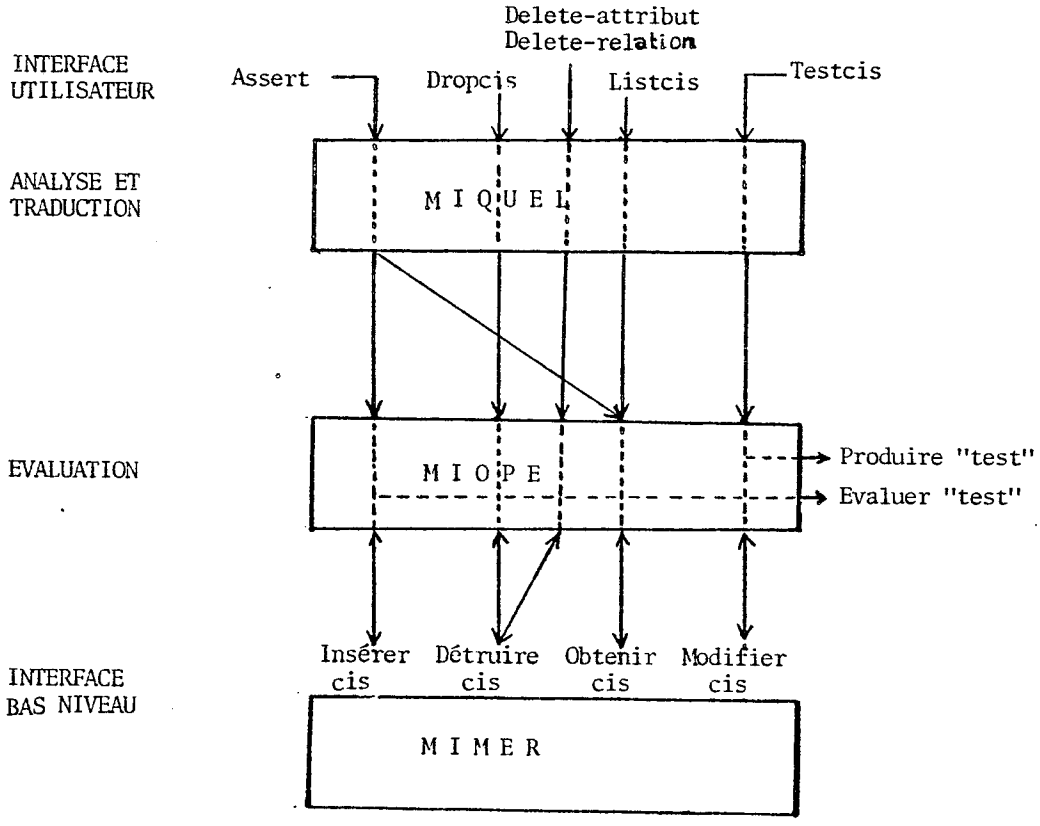
ISIS touche à tous les modules de la version centralisée de MICROBE : le traducteur pour la gestion de l'interface ISIS et la génération d'arborescences, l'interpréteur pour la validation des opérations de mise à jour et MIMER pour la gestion du catalogue INTEGRITE.

Seuls les modules de gestion de l'interface et du catalogue INTEGRITE sont actuellement opérationnels. Pour la partie validation des opérations de mise à jour, les spécifications sont achevées, mais la réalisation est bloqueé par des contraintes de place mémoire même en utilisant une politique "d'overlay".

Nous n'avons pas traité ici les CIS multi-relationnelles, d'abord pour leur compléxité (difficiles à tester, méthode de validation complexe <BLA81> et <NIC79>, temps d'évaluation long, ...) et ensuite pour les limitations imposées par les micro-ordinateurs.

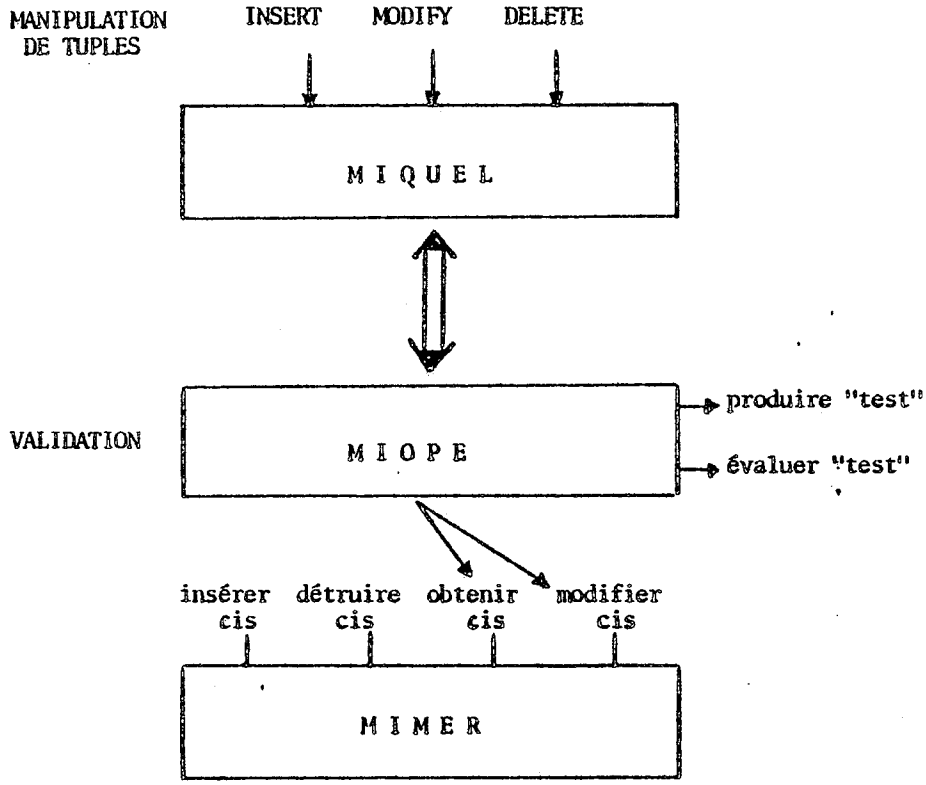
MICROBE ET ISIS :

Ci-joint une architecture d'ISIS dans les 3 modules de MICROBE. Remarquons que les commandes de destruction de relation et d'attributs, détruisent aussi toutes les CIS associées à la relation ou aux attributs détruits.



STRUCTURE D'ISIS DANS MICROBE

VALIDATION DANS MICROBE :



VALIDATION DES OPERATIONS
DE MISE A JOUR DANS ISIS

8.2 L'ANALYSE ET LA TRADUCTION DES CIS

Ce module tourne déjà depuis Juin 81 pour les commandes de manipulation de tuples. Il a été adapté pour la traduction des CIS dans <ACO82>. La réalisation est présentée en détails (structures de données, programmes d'analyses lexicale, syntaxique et sémantique, programmes de génération de l'arbre, etc ...) dans <FRT80>.

Etapes de création d'une assertion

Ci-dessous l'algorithme de création d'une assertion.

DEBUT
  arrivée d'une CIS.

```
analyses lexicale, syntaxique et sémantique.
traduction de la CIS en arborescence algébrique.
optimisation et restructuration de l'arborescence.
test de consistance de la CIS.
si (pas de conflit)
alors test de la CIS sur les données préexistantes.
        si (test correct)
        alors insertion de la CIS dans le catalogue.
        sinon rejet de la CIS.
sinon rejet de la CIS.
FIN-ALGORITHME
```

8.3 LA VALIDATION DES OPERATIONS DE MISE A JOUR

8.3.1 ALGORITHME GENERAL DE VALIDATION -

L'algorithme général de validation est le suivant :

```
DEBUT
    arrivée d'une opération de maj sur la base
    test si op=I ou op=M ou op=D
    SELON      co. type de l'opération de mise à jour co.
      cas op = I
            recherche des assertions satisfaisant
            les 3 conditions suivantes :
              1) relation de l'assertion = relon maj
              2) type de l'assertion € I ds, da, ss I
              3) mode de l'assertion € I I, IM, ID, IMD I
            Appel de la procédure INSERTION

      cas op = M

            recherche des assertions satisfaisant
            les 2 conditions suivantes :
              1) ces assertions portent sur la relation maj
              2) mode € I M,IM,DM, IMD I
            appel procédure MODIFICATION

      cas op = D

            recherche des assertions satifaisant les
            3 conditions suivantes :
              1) les assertions portent sur la relation maj.
              2) mode € I D,ID,MD, IMD I
              3) type de l'assertion € I sa, da I
            appel de la procédure DESTRUCTION

    FIN-SELON
FIN-ALGORITHME
```

8.3.2 VALIDATION DE L'INSERTION -

Nous présentons ici l'algorithme d'insertion de tuples en tenant compte des différentes contraintes s'associant à celle-ci. L'insertion du tuple ne se fera que si toutes les contraintes mises en cause sont vérifiées.

```
PROCEDURE INSERTION

        vérifier l'existence de la relation R et de ses attributs.

        tantque non (fin insertion) faire
        |
        |       tantque non (fin attribut) faire
        |       |       lire attribut
        |       |       convertir chaine en REEL ou ENTIER
        |       |       suivant le type de l'attribut.
        |       |       affectation de la valeur au tuple
        |       |
        |       fin tantque
        |
        |       tantque non (fin contrainte) faire
        |       |       Appel procedure EVALCIS co. Cf chap. 8.3.5 co.
        |       |       si test faux
        |       |       alors sortir (message et contrainte violée)
        |       |             goto pas-d'insertion
        |       |       finsi
        |       fintque
        |
        |       insertion du tuple
        |
    pas-d'insertion:
        |
    fintque

FIN-INSERTION
```

8.3.3 VALIDATION DE LA MODIFICATION -

Dans cet algorithme on effectue deux lectures sur les données (dont une est éventuellement conditionnée par le test vrai de toutes les contraintes associées). La première consiste donc à parcourir les tuples de la relation en question et à tester les contraintes. Une fois toutes les contraintes testées, si une contrainte est violée alors la modification est rejetée, sinon (aucune CIS n'est violée) on effectue la seconde lecture des tuples et on effectue la modification réellement sur la base.

```
PROCEDURE MODIFICATION

        vérifier l'existence de la relation et de ses attributs.

        tantque   non  (fin des tuples de R)  faire
                  lire tuple de R
                  modifier attribut au niveau mémoire

                  tantque non (fin des contraintes) faire

                      Appel procedure EVALCIS co. Cf chap. 8.3.5 co.

                      si test = faux
                      alors    sortir (message et contrainte)
                               goto fin-modification
                      finsi
                  fintque
        fintque

        tantque   non  (fin des tuples de R)  faire

                  lire tuple de R
                  tester condition de modification
                  si test = vrai
                  alors    remplacer attribut
                           réecrire tuple dans R
                  finsi
        fintque
        fin-modification:

FIN-MODIFICATION
```

8.3.4 VALIDATION DE LA SUPPRESSION -

Dans cet algorithme, comme dans celui de la modification on effectue deux parcours de la relation.

```
PROCEDURE DESTRUCTION

    vérifier l'existence de la relation et de ses attributs etc...

    tantque  non  (fin des tuples de R) faire
    |       lire tuple de R
    |       tantque non (fin des contraintes) faire
    |       |       Appel procédure EVALCIS
    |       |       si test = faux
    |       |       alors  sortir (message et contrainte)
    |       |              goto fin-destruction
    |       |       finsi
    |       fintque
    fintque

    tantque  non  (fin des tuples de R)  faire
    |       lire tuple de R
    |       tester condition de destruction
    |       si      test = vrai
    |       alors détruire tuple
    |       finsi
    fintque
    fin-destruction:

FIN-DESTRUCTION
```

8.3.5 EVALUATION DES CIS -

Il existe deux types de contraintes si on s'intéresse au coût de vérification : d'une part les contraintes sans agrégats (ss et ds) et d'autre part, les contraintes avec agrégats (sa et da). Les dernières coûtent généralement plus cher en temps d'évaluation car la mise à jour d'un attribut nécessite la lecture de tous les tuples interférant avec l'assertion et son évaluation.

Ci-dessous l'algorithme d'évaluation d'une contrainte quels que soient son type et l'opération de mise à jour.

```
PROCEDURE EVALCIS

    SELON  co. type de la contrainte co.
```

cas contrainte = ss co. statique sans agrégat co.
co. traitement de cis statique sans agrégat co.

```
debut
    si opération = destruction
    alors    erreur
    finsi
    évaluer l'assertion sur le tuple mis à jour.
fin
```

cas contrainte = sa co. statique avec agrégat co.
co. traitement de contrainte statique avec agrégat co.

```
debut
    évaluer l'agrégat avec le tuple modifié
    en tenant compte du tableau du chapitre 6.3.3
    modifier l'assertion : mettre à la place
    de l'arbre associé à l'agrégat la valeur
    calculée de celui-ci.

    mettre à jour la valeur de l'agrégat (valag)
    dans le catalogue CONTRAINTES.

    Enfin, évaluer la contrainte avec la nouvelle
    valeur de l'agrégat.
fin
```

cas contrainte = ds co. dynamique sans agrégat co.
co. traitement de cis dynamique sans agrégat co.

```
debut
    si opération ◇ modification
    alors    erreur
    sinon
        modifier la valeur du tuple en conservant
        l'ancienne.

        évaluer l'assertion en tenant compte du
        tuple modifié et de son ancienne valeur.
    finsi
fin
```

cas contrainte = da co. dynamique avec agrégat co.
co. traitement de cis dynamique avec agrégat co.

```
debut
    si      opération = insertion
    alors   erreur
```

sinon

modifier le tuple et garder l'ancienne valeur.

évaluer l'agrégat avec le tuple modifié en tenant compte du tableau du chapitre 6.3.3.

modifier l'assertion : remplacer l'arbre agrégat par sa valeur.

mettre à jour valag dans le catalogue CONTRAINTES.

évaluer la contrainte

finsi

fin

FIN-EVALCIS

8.4 LES PRIMITIVES DE GESTION DES CIS

Les primitives d'ISIS sont construites à partir de celles de MIMER <FER81b>. Elles sont de 2 types :

- les primitives utilisateurs (le niveau C de MIMER),
- et les autres (le niveau B de MIMER).

Nous présentons ici l'ensemble des primitives du premier type :

Paramètres des primitives :

<tuple-cis> est composé des champs,

. témoin d'existence
. identificateur de la CIS
. opération sur laquelle elle s'applique (I/M/IM ...)
. type de la CIS
. nom interne de la relation
. pointeur sur l'arbre de la CIS
. nombre de noeud de l'arbre
. et valeur de l'agrégat (sinon valeur = nil)

‹arbre› est l'arborescence prédicat dont chaque noeud est composé :

. témoin d'existence
. type du noeud (opérateur, connecteur, ...)
. numéro (de l'opérateur, du connecteur, ...)
. pointeur fils
. pointeur frère
. indication sur le noeud

CE est le code d'erreur des procédures

IR1 (et IA1) est le nom interne de la relation (respectivement de l'attribut).

Primitives :

INSERER-CIS (‹tuple-cis›, ‹arbre›, CE) ;

- paramètre en entrée : ‹tuple-cis›, ‹arbre›
- paramètre en sortie : CE
- cette primitive permet d'insérer une contrainte définie par ‹tuple-cis› pour le type, l'opération, .. et par ‹arbre› pour l'arbre associé.

DELETE-CIS (‹nom-cis›, CE) ;

- paramètre en entrée : ‹nom-cis›
- paramètre en sortie : CE
- détruit, si elle existe, la contrainte portant le nom ‹nom-cis›.

DELETE-CIS-REL (IR1, CE) ;

- paramètre en entrée : IR1
- paramètre en sortie : CE
- détruit toutes les contraintes associées à la relation IR1.

DELETE-CIS-DOM (IR1, IA1, CE) ;

- paramètre en entrée : IR1, IA1
- paramètre en sortie : CE
- détruit toutes les contraintes associées à la relation IR1 et portant sur l'attribut IA1.

OBTENIR-CARACTERISTIQUES-CIS (‹nom-cis›, ‹tuple-cis›, ‹arbre-cis›, ‹tableau-attributs›, CE) ;

- paramètre en entrée : ‹nom-cis›

- paramètre en sortie : <tuple-cis>, <arbre-cis>,
  <tableau-attributs>, CE
- obtenir toutes les caractéristiques d'une cis en partant de <nom-cis>.

OBTENIR-CARACTERISTIQUES-CIS-REL (IR1, tableau-de-tuple-cis, CE) ;

- paramètre en entrée : IR1
- paramètre en sortie : <tableau-de-tuple-cis>, CE
- obtenir toutes les caractéristiques de toutes les cis associées à la relation IR1.

OBTENIR-CARACTERISTIQUES-CIS-ATTRIBUT (IR1, IA1,
<tableau-de-tuple-cis>, CE) ;

- paramètre en entrée : IR1, IA1
- paramètre en sortie : <tableau-de-tuple-cis>, CE
- obtenir les caractéristiques des cis associées à la relation IR1 et portant sur l'attribut IA1.

MODIFIER-CIS (<tuple-cis>, <nouvelle-valeur>, CE)

- paramètre en entrée : <tuple-cis>, <nouvelle-valeur>
- paramètre en sortie : CE
- Modifie le champ VALAG de tuple-cis par <nouvelle-valeur>.

Pour les primitives de niveau plus bas citons :

_ 2 primitives de création des 2 catalogues, intégrées dans la primitive de création de la base de MIMER,

_ et 2 primitives d'insertion de tuples dans ces catalogues.

8.5 SESSION MICROBE : LE LANGAGE MIQUEL

LANCEMENT DE MICROBE

```
>RUN MICROBE

     -- Bienvenue chez MICROBE --

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

CREATION DE LA BASE : EX1.DAT

```
MIQ>CB;.


?...nom de la base : EX1
?...votre nom : FERRAT
?...nom de disque ou la base se trouve : DK2:
?...votre UIC : °100,2$
?..la taille de la base : 40
.......La base EX1           est creee.

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

OUVERTURE DE LA BASE EX1

```
MIQ>OB;


?...nom de la base : EX1
?...votre nom : FERRAT
?...nom de disque ou la base se trouve : DK2:
?...votre UIC : °100,2$
.......La base EX1           est ouverte.

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

CREATION DE 2 RELATIONS : EMLPOY ET DEPT

```
MIQ>CR EMPLOY :  NOM (CHAR(10)), PRENOM(CHAR(10)), SAL (ENTIER),
MIQ>NODEPT ( ENTIER ) ;

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


.......La relation EMPLOY est cree.
.......L'attribut NOM     est cree.
.......L'attribut PRENOM  est cree.
.......L'attribut SAL     est cree.
```

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

MIQ>CR DEPT :  NODEPT (ENTIER), CHEF(CHAR(10)), NBEMP (ENTIER),
MIQ>BUDGET(ENTIER) ;

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


.......La relation DEPT   est cree.
.......L'attribut NODEPT est cree.
.......L'attribut CHEF   est cree.
.......L'attribut NBEMP  est cree.
.......L'attribut BUDGET est cree.

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

AFFICHAGE DU CATALOGUE DES RELATIONS

```
MIQ>EB;

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

caracteristiques des relations de la base :

nom	type	cardinalite	degre	longueur	type de stockage
EMPLOY	B	0	4	24	
DEPT	B	0	4	16	

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

AFFICHAGE DES ATTRIBUTS DE EMPLOY

```
MIQ>EA EMPLOY : * ;

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

caracteristiques des attributs de la relation EMPLOY :

nom	type	longueur	position
NOM	C	10	1
PRENOM	C	10	11
SAL	E	2	21
NODEPT	E	2	23

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

INSERTION DE TUPLES DANS EMPLOY

```
MIQ>INSERT_INTO EMPLOY ;

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


  Pour chaque attribut, inserez sa valeur (ou '/' si indéfinie)

        1 eme tuple?
 NOMATT:NOM    , TYPE:C , LONGUEUR: 10, VALEUR:DUPONT
 NOMATT:PRENOM, TYPE:C , LONGUEUR: 10, VALEUR:PIERRE
 NOMATT:SAL    , TYPE:E , LONGUEUR:  2, VALEUR:4000
 NOMATT:NODEPT, TYPE:E , LONGUEUR:  2, VALEUR:2
...Le      1 eme tuple a ete insere.
?...Insertion d'un autre tuple (O/N) O
        2 eme tuple?
 NOMATT:NOM    , TYPE:C , LONGUEUR: 10, VALEUR:DURANT
 NOMATT:PRENOM, TYPE:C , LONGUEUR: 10, VALEUR:MICHEL
 NOMATT:SAL    , TYPE:E , LONGUEUR:  2, VALEUR:5000
 NOMATT:NODEPT, TYPE:E , LONGUEUR:  2, VALEUR:1
...Le      2 eme tuple a ete insere.
?...Insertion d'un autre tuple (O/N) O
        3 eme tuple?
 NOMATT:NOM    , TYPE:C , LONGUEUR: 10, VALEUR:ARMAND
 NOMATT:PRENOM, TYPE:C , LONGUEUR: 10, VALEUR:JACOB
 NOMATT:SAL    , TYPE:E , LONGUEUR:  2, VALEUR:6000
 NOMATT:NODEPT, TYPE:E , LONGUEUR:  2, VALEUR:3
...Le      3 eme tuple a ete insere.
?...Insertion d'un autre tuple (O/N) N

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

UNE FOIS TOUTES LES INSERTIONS TERMINES

LISTER TOUS LES TUPLES DE EMPLOY

```
MIQ>SELECT *
MIQ>FROM EMPLOY ;

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

.....Le resultat de votre requete.

NOM	PRENOM	SAL	NODEPT
DUPONT	PIERRE	4000	2
DURANT	MICHEL	5000	1
ARMAND	JACOB	6000	3
DONTON	CLAUDE	4500	2
GARAUD	PASCAL	4500	1
RANDON	ALEXIS	7000	1

nombre de tuples resultat : 6

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

LISTER TOUS LES DEPARTEMENTS

```
MIQ>SELECT *
MIQ>FROM DEPT ;

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

.....Le resultat de votre requete.

NODEPT	CHEF	NBEMP	BUDGET
1	DURANT	56	9000
2	DURANT	25	1000
3	DUPONT	54	4000
5	DURANTON	25	7000

nombre de tuples resultat : 4

QUELS SONT LES EMPLOYES DU DEPARTEMENT 1 ?

```
MIQ>SELECT *
MIQ>FROM EMPLOY
MIQ>WHERE NODEPT = 1 ;

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


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

 +=======================================+
 ! NOM          PRENOM       SAL     NODEPT !
 +=======================================+
 ! DURANT       MICHEL        5000        1 !
 +---------------------------------------+
 ! GARAUD       PASCAL        4500        1 !
 +---------------------------------------+
 ! RANDON       ALEXIS        7000        1 !
 +---------------------------------------+
 nombre de tuples resultat :      3

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

QUELS SONT LES NOMS DES EMPLOYES TRAVAILLANT DANS LES DEPARTEMENTS AYANT UN BUDGET SUPERIEUR A 4000 ?

```
MIQ>SELECT NOM
MIQ>FROM EMPLOY
MIQ>WHERE NODEPT = SELECT NODEPT
MIQ>                  FROM DEPT
MIQ>             WHERE BUDGET > 4000 ;

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

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

indice	noeud	type	fils	frer
1----->	project	operateur	2	0
2----->	join	operateur	15	3
15----->	EMPLOY	relation	0	5
5----->	project	operateur	6	16
6----->	select	operateur	8	7
8----->	DEPT	relation	0	14
14----->	sup	connecteur	12	0
12----->	BUDGET	domaine	0	13
13----->	4000	constante	0	0
7----->	NODEPT	domaine	0	0
16----->	egal	connecteur	17	0
17----->	NODEPT	domaine	0	18
18----->	NODEPT	domaine	0	0
3----->	NOM	domaine	0	0

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


 +============+
 ! NOM        !
 +============+
 ! DURANT     !
 +------------+
 ! GARAUD     !
 +------------+
 ! RANDON     !
 +------------+
 nombre de tuples resultat :       3

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

QUELS SONT LES NOMS ET PRENOMS DES EMPLOYES DU DEPT 1 GAGNANT PLUS 6000F ?

```
MIQ>SELECT NOM, PRENOM
MIQ>FROM EMPLOY
MIQ>WHERE NODEPT =1 AND SAL > 6000 ;

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

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

     indice        noeud              type          fils      frer

      1----->      project            operateur      2         0
      2----->      select             operateur      5         3
      5----->      EMPLOY             relation       0        12
     12----->      and                connecteur     8         0
      8----->      egal               connecteur     6        11
      6----->      NODEPT             domaine        0         7
      7----->      1                  constante      0         0
     11----->      sup                connecteur     9         0
      9----->      SAL                domaine        0        10
     10----->      6000               constante      0         0
      3----->      NOM                domaine        0         4
      4----->      PRENOM             domaine        0         0


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


 +=======================+
 ! NOM        PRENOM     !
 +=======================+
 ! RANDON     ALEXIS     !
 +-----------------------+
 nombre de tuples resultat :       1

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

CREER L'ATTRIBUT LIBELLE DEPT (LIBDPT) DANS LA RELATION DEPT

```
MIQ>CA DEPT :  LIBDPT (CHAR(10)) ;
?...Voulez-vous l'affichage de l'arbre (O/N):
.......L'attribut LIBDPT est cree.
>>> Nouvelle requete MIQUEL ou fermer la base <<<
```

LISTER TOUS LES TUPLES DE LA RELATION DEPT

```
MIQ>SELECT *
MIQ>FROM DEPT ;
?...Voulez-vous l'affichage de l'arbre (O/N):
```

.....Le resultat de votre requete.

NODEPT	CHEF	NBEMP	BUDGET	LIBDPT
1	DURANT	56	9000	??????????
2	DURANT	25	1000	??????????
3	DUPONT	54	4000	??????????
5	DURANTON	25	7000	??????????

nombre de tuples resultat : 4

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

LE DEPT 1 EST LE DEPT VENTES METTRE A JOUR

```
MIQ>MODIFY DEPT SET LIBDPT ='VENTES'  WHERE NODEPT = 1 ;
?...Voulez-vous l'affichage de l'arbre (O/N):
>>> Nouvelle requete MIQUEL ou fermer la base <<<
```

LISTER TOUS LES TUPLES DE DEPT

```
MIQ>SELECT *
```

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

.....Le resultat de votre requete.

NODEPT	CHEF	NBEMP	BUDGET	LIBDPT
1	DURANT	56	9000	VENTES
2	DURANT	25	1000	??????????
3	DUPONT	54	4000	??????????
5	DURANTON	25	7000	??????????

nombre de tuples resultat : 4

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

DETRUIRE LA RELATION EMPLOY

MIQ>KILL_RELATION EMPLOY ;

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

.......La relation EMPLOY a ete supprimee.

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

FERMETURE DE LA BASE

MIQ>FB;

OPE> La base est fermee.

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

FIN DE SESSION MICROBE : SORTIE

MIQ>FIN;

...MICROBE vous remercie et à très bientot ! ...

8.6 SESSIONS ISIS

8.6.1 L'INTERFACE MIQUEL D'ISIS -

INTERFACE UTILISATEUR D'ISIS

LANCEMENT DU PROGRAMME

```
>RUN MIQTSK

    __ Bienvenue chez MICROBE __

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

MIQ>OB;

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

CREATION D'UNE CONTRAINTE

```
MIQ>ASSERT CIS2  ON REL2 ;
MIQ>SAL <= SELECT  MOY ( SAL)
MIQ>      FROM REL2
MIQ>     WHERE NODEPT = 25 ;



NOMCIS =   CIS2
OPERAT = IMD
RELAT  = (   0,    0)
ARBRE  = (   0,    0)
help --->0

      0 :  Parcours recurcif ARBRE
      1 :  Imp. ARBRE
      2 :  Imp. TABCOND
      3 :  Imp. TDOM
      4 :  Imp. TRELAT
      5 :  Imp. TCONST
      6 :  FIN .
```

choix -->0

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

indice	noeud	type	fils	frere
1----->	assert	operateur	2	0
2----->	REL2	relation	0	10
10----->	inf-esal	connecteur	8	0
8----->	SAL	domaine	0	3
3----->	moyenne	fonction	4	0
4----->	project	operateur	5	0
5----->	select	operateur	7	6
7----->	REL2	relation	0	13
13----->	esal	connecteur	11	0
11----->	NODEPT	domaine	0	12
12----->	25	constante	0	0
6----->	SAL	domaine	0	0

DETRUIRE LA CONTRAINTE CIS1

MIQ>DROPCIS REL1 CIS1 ;

help --->0

0 : Parcours recurcif ARBRE
1 : Imp. ARBRE
2 : Imp. TABCOND
3 : Imp. TDOM
4 : Imp. TRELAT
5 : Imp. TCONST
6 : FIN .

choix -->0

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

indice	noeud	type	fils	frere
1----->	dropcis	operateur	2	0
2----->	REL1	relation	0	3
3----->	CIS1	contrainte	0	0

LISTER TOUTES LES CIS

```
MIQ>LISTCIS  REL1  * ;

help --->0

       0 :  Parcours recurcif ARBRE
       1 :  Imp. ARBRE
       2 :  Imp. TABCOND
       3 :  Imp. TDOM
       4 :  Imp. TRELAT
       5 :  Imp. TCONST
       6 :  FIN .

 choix -->0

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

indice	noeud	type	fils	frere
1----->	listcis	operateur	2	0
2----->	REL1	relation	0	0

TESTER SUR REL1 LA CIS "CIS1"

```
MIQ>TESTCIS REL1  CIS1 ;

help --->0

       0 :  Parcours recurcif ARBRE
       1 :  Imp. ARBRE
       2 :  Imp. TABCOND
       3 :  Imp. TDOM
       4 :  Imp. TRELAT
       5 :  Imp. TCONST
       6 :  FIN .

 choix -->0

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

indice	noeud	type	fils	frere
1----->	testcis	operateur	2	0
2----->	REL1	relation	0	3
3----->	CIS1	contrainte	0	0

8.6.2 PRIMITIVES AU NIVEAU MIMER -

```
MENU (O/N) ->
```

```
C:c-cis
D:d-cis
M:m-cis
F:fb
```
MENU

OUVERTURE DE LA BASE
```
---->
OK
```
VISUALISATION DE TOUTES LES CIS
```
---->
1:all/2:rel/3:att
 -->1
***** debut OCC *****  CE =                0
***** fin OCC *****  CE =                 0

 NOMCIS      MODE  NBNOD      VALAG      TYPECIS

 CIS1        IM       3      0.000000E+00     SS

                tableau arborescence

          indice    ttype     tnum      tfils     tfrere    tind     valnod

            1-----> C         1         1         1         1
            2-----> D         1         0         0         0
            3-----> F         2         0         9         0

MENU (O/N) ->
```

```
O:ob
V:v-cis
D:d-cis
M:m-cis
F:fb
```
CREATION D'UNE CIS "CIS2"
```
? ==>

temoin ?1
nomcis ?CIS2
mode ?IMD
type ?SA
ps ?2
depl ?2
valag ?6.8
```

```
New tuple (Y/N):Y
type:C
tnum:2
fils:1
frere:0
tind:0
New tuple (Y/N):Y
type:D
tnum:2
fils:0
frere:0
tind:0
New tuple (Y/N):Y
type:K
tnum:1
fils:0
frere:0
tind:0
New tuple (Y/N):N
```

VISUALISATION DE TOUTES LES CIS

```
---->
1:all/2:rel/3:att
 -->1
***** debut OCC *****  CE =              0
***** fin OCC *****  CE =               0
```

NOMCIS	MODE	NBNOD	VALAG	TYPECIS
CIS2	IMD	3	6.800000E+00	SA

tableau arborescence

indice	ttype	tnum	tfils	tfrere	tind	valnod
1----->	C	2	1	0	0	
2----->	D	2	0	0	0	
3----->	K	1	0	0	0	

NOMCIS	MODE	NBNOD	VALAG	TYPECIS
CIS1	IM	3	0.000000E+00	SS

tableau arborescence

indice	ttype	tnum	tfils	tfrere	tind	valnod
1----->	C	1	1	1	1	
2----->	D	1	0	0	0	
3----->	F	2	0	9	0	

MODIFICATION DE LA VALEUR "VALAG" D'UNE CIS

```
? ==>
cis à maj?
new val ?
in MVA
out MVA
OK
```

VISUALISATION DE TOUTES LES CIS

```
---->
1:all/2:rel/3:att
 -->
***** debut OCC *****  CE =                0
***** fin OCC *****  CE =                  0

 NOMCIS      MODE  NBNOD      VALAG      TYPECIS

 CIS2        IMD      3     9.100000E+00     SA

                tableau arborescence

          indice    ttype     tnum      tfils     tfrere    tind     valnod

           1-----> C         2         1         0         0
           2-----> D         2         0         0         0
           3-----> K         1         0         0         0

 NOMCIS      MODE  NBNOD      VALAG      TYPECIS

 CIS1        IM       3     0.000000E+00     SS

                tableau arborescence

          indice    ttype     tnum      tfils     tfrere    tind     valnod

           1-----> C         1         1         1         1
           2-----> D         1         0         0         0
           3-----> F         2         0         9         0

MENU (O/N) ->
```

DESTRUCTION DE LA CIS "CIS2"

```
----->
C :cis / R : cis-rel / A : cis-att
  -->
cis ?
 ***** debut DC *****
--- --
 cis détruite
***** fin DC
```

VISUALISATION DE TOUTES LES CIS

```
----->
1:all/2:rel/3:att
  -->
***** debut OCC *****  CE =            216
***** fin OCC *****  CE =              0

 NOMCIS      MODE  NBNOD      VALAG      TYPECIS

 CIS1        IM      3      0.000000E+00     SS

                tableau arborescence

          indice    ttype     tnum     tfils     tfrere     tind     valnod

            1------>  C         1         1         1         1
            2------>  D         1         0         0         0
            3------>  F         2         0         9         0
```

FERMETURE DE LA BASE

```
----->F
```

9.0 CONCLUSIONS ET PERSPECTIVES

Le problème d'un sous-système d'intégrité sémantique est important pour 3 raisons :

_ d'abord, les CIS enrichissent la sémantique de la base de données, surtout quand on se trouve en face d'un modèle pauvre en sémantique, c'est le cas du modèle relationnel.

_ de plus, il y a suppression de nombreuses erreurs dues aux opérations de mise à jour.

_ et enfin, ce problème est directement lié aux traitements des requêtes et à la maintenance des vues.

Cette thèse traite de la conception et de la réalisation (partielle) d'un sous-système d'intégrité sémantique. Notre contribution porte :

_ Sur le plan méthodologique :

. D'abord, au niveau de la classification des CIS, nous avons énuméré une liste non exhaustive des critères de classification. Nous avons ensuite redéfini l'interface SEQUEL (langage MIQUEL) pour l'expression des CIS.

. Pour manipuler, stocker, évaluer les CIS, nous les avons traduites en arborescences d'opérateurs algébriques et logiques. Ceci permet d'homogénéiser requêtes et CIS dans un contexte algébrique. Une méthode de traduction basée sur un ensemble de règles prédéfinies a été proposée. En se basant sur la sémantique des requêtes, nous avons donné aussi une preuve de ces règles.

. Les CIS en tant que telles peuvent engendrer des problèmes de consistance lors de leur définition ou évaluation. Nous avons proposé une méthode de résolution du problème dans un contexte réduit.

. En se limitant à une classe de CIS mono-relationnelles avec agrégats, nous avons défini une méthode de validation optimisée.

_ Sur le plan réalisation :

. 2 modules sont opérationnels dans le cadre du projet MICROBE, celui de la traduction des CIS en arborescences et celui de la création et la gestion du catalogue des CIS grâce aux primitives de la micro-mémoire relationnelle MIMER.

Néanmoins, ISIS comporte des imperfections au niveau de la validation des opérations de mise à jour.

_ D'abord sur le plan conception, nous n'avons pas traité les CIS multi-relationnelles. Ceci est parfaitement justifié dans un contexte micro-SGBD tel que MICROBE.

_ Les algorithmes de validation ont été définis (Cf. Chapitre 8), mais n'ont pas été réalisés. Avec une extension de mémoire du micro-ordinateur LSI 11/23, ceci peut se faire dans un délai relativement court.

En intégrité sémantique, plusieurs problèmes restent ouverts, citons :

_ actuellement, lors de l'optimisation de la validation des opérations de maj, seule une classe de CIS a été prise en compte ‹NIC79›, ‹BBC80› et ‹BLA81›. Un des axes de recherche consiste à étendre celle-ci.

_ Le concept de valeur indéfinie ("null value") a été introduit en 1979 par CODD ‹COD79› et LIPSKI ‹LIP79›. Pour évaluer les CIS ou les requêtes dans un système tolérant les valeurs indéfinies, il a fallu une logique tri-valuée V (vrai), F (faux) et ? (indéfini). Dans ‹COD79›, une redéfinition des opérateurs algébriques a été faite. Il s'avère nécessaire de lancer une étude concernant l'impact des valeurs indéfinies sur les contraintes d'intégrité.

_ concernant les CIS "trigger", nous savons qu'une modification d'une relation R1 peut engendrer une modification d'une autre relation R2. Supposant qu'en plus la modification de R2 engendre de même la modification de R1 et ainsi de suite. Nous remarquons, qu'à un certain moment on risque de perdre le contrôle des maj effectuées. Ce cas peut être généralisé à plusieurs relations. C'est un des sujets intéressants dans le cadre des bases de données relationnelles.

_ Les sous-systèmes d'intégrité nécessitent aussi des mécanismes de verrouillage et de confidentialité. Une étude a été faite par ESWARAN ‹ESW76› dans le system-R, cependant beaucoup reste à faire.

_ Dans un environnement réparti, le problème est un peu plus complexe, très peu d'études ont été menées dans ce cadre. Citons, une étude des CIS au niveau de l'équivalence de schémas de base de données suivant la conception "ascendante" ou "descendante" de la base ‹ADI78› et ‹AND80›, une étude de coût basée sur le temps de vérification des CIS et celui dû à la transmission des données d'un site à un autre ‹BAD78› et ‹BAD79›.

L'étude du sous-système ISIS a débuté alors qu'une version de MICROBE tournait déjà. Notre expérience a montré qu'il faut tenir compte du problème des CIS dès la conception initiale du SGBD. Dans

le cas contraire beaucoup de problèmes risquent de se poser du fait que le sous-système d'intégrité concerne toutes les couches du SGBD.

10.0 LES ANNEXES

10.1 ANNEXE 1 : DEFINTION DES OPERATEURS RELATIONNELS

Nous utilisons le calcul relationnel défini dans <COD72> et la notion de concaténation de tuples.

soient d et e deux tuples avec

d = (d1,d2, ...dn)

e = (e1,e2, ...em)

la concaténation de d et e est dénotée

d^e = (d1,d2, ...dn,e1,e2, ...em)

Opérateurs binaires:

- Produit cartésien : *

  R * S = ((r^s) : r ∈ R , s ∈ S)

- Intersection : ∩

  R ∩ S = (t : t ∈ R et t ∈ S)

- Union : U

  R U S = (t : t ∈ R ou t ∈ S) - (R ∩ S)

Opérateurs unaires :

- Projection : R [<liste-attribut>]

  R [A] = (r [A] : r ∈ R)

- Sélection : R [A θ B]

  R [A θ B] = (r : r ∈ R et (r[A] θ r[B]))

- θ-Produit : R [A θ B] S

  R [A θ B] S = ((r^s) : r ∈ R, s ∈ S et (r [A] θ s[B]))

- Anti-projection :] <attribut>[R

  ]B[R(A,B) = (y : a ∈ A , y ∈ B ∀ a ∃ (a,y) ∈ R)

10.2 ANNEXE 2 : SYNTAXE COMPLETE DU LANGAGE MIQUEL

Symboles utilisés :

[ ] : signifie que le contenu est optionnel.
| : est équivalent au "ou" logique.

Syntaxe :

MIQUEL ::= <requete> ;

<requete> ::= <interrogation>

| <creer-rel> | <kill-rel> | <creer-att> | <kill-att>

| <insert> | <modify> | <delete>

| <lister-rel> | <caract-rel> | <caract-att>

| <creer-base> | <kill-base> | <open-base> | <close-base>

| <creer-cls> | <drop-cls> | <lister-cls> | <test-cls>

Manipulation de structures :

<creer-rel> ::= CREATE_RELATION <def-rel>
| CR <def-rel>
<def-rel> ::= <identrel> := <rel-affect>
| <identrel> : <list-att-def>
<rel-affect> ::= <identrel>
| <select1>
| <identrel> UNION <identrel>
| <identrel> INTERSECT <identrel>
| <identrel> DIFFERENCE <identrel>
| <identrel> JOIN <identrel> WHERE <att> <op> <att>

<list-att-def> ::= <att-def> | <att-def> , <list-att-def>
<att-def> ::= <att> (<type>)
<type> ::= ENTIER | REEL | CHAR (<entier>)
<kill-rel> ::= KILL_RELATION <identrel>

<creer-att> ::= CREATE_ATTRIBUT <identrel> : <list-att-def>
| CA <identrel> : <list-att-def>

<kill-att> ::= KILL_ATTRIBUT <identrel> : <list-att>
<list-att> ::= <att> | <att> , <list-att>

Manipulation de tuples :

<insert> ::= INSERT_INTO <identrel>

```
<modify>    ::=   MODIFY <identrel> SET <att> := <operande>
                              [ WHERE <booleen> ]

<operande>::=   <constante> | <att>
```

Affichage du catalogue :

```
<lister-rel> ::= EDIT_BASE | EB

<caract-rel>          ::= EDIT_RELATION   <identrel>
                        | ER   <identrel>

<caract-att>          ::= EDIT_ATTRIBUT   <identrel>   : <attsel>
                        | EA          <identrel>   :   <attsel>

<attsel>     ::=   * | <att>
```

Manipulation de la base :

Toutes ces commandes sont interactives. Le système demandera les informations nécessaires pour les accomplir.

```
<creer-base>          ::=   CREATE_BASE   |   CB
<kill-base> ::=   KILL_BASE
<open-base>           ::=   OPEN_BASE   | OB
<close-base>          ::=   CLOSE_BASE   |   FB
```

Manipulation de CIS :

```
<drop-cis> ::=   DROPCIS   <identrel>     <list-nomcis>
                        | DC      <identrel>     <list-nomcis>
<list-nomcis>         ::=   *   |   <nomcis>

<lister-cis> ::=   LISTCIS     <rel-cis>
                        | LC            <rel-cis>
<rel-cis>    ::=   *     | <identrel>   <list-nomcis>

<test-cis>   ::=   TESTCIS     <rel-cis>
                        | TC            <rel-cis>

<creer-cis>::=    ASSERT <nomcis> ON <op-rel>   :   <assertion>
                        | AC        <nomcis> ON <op-rel>   :   <assertion>
<op-rel>     ::=   <identrel> | <op-maj> OF <identrel>
<op-maj>     ::=   (   <op1>   [ , <suite-op>   ] )
<suite-op>   ::=   <op1> , <suite-op>
<op1>                 ::=   I | M | D

<assertion> ::=       <fact-bool2>
```

```
                    | <fact-bool2>   <op-bool> <assertion>

<op-bool> ::=  AND | OR

<fact-bool2>          ::=   <predicat>
                          | ( <predicat> )

<predicat> ::=  [ OLD | NEW ] <att> <op-arithm>
                        [ OLD | NEW ]   <expr1>
                      | <att> IS [ NOT ] IN <select1>
                      | <att> DOES [ NOT ] CONTAINS <select1>
                      | <att> BETWEEN <param1> AND <param2>

<expr1>               ::=   <att>
                          | <constante>
                          | SELECT   <fonction> ( [ UNIQUE ] <att> )
                            FROM     <identrel>
                            [ WHERE     <booleen> ]

<constante>           ::=       <entier>
                          |     <reel>
                          |     <chaine>

<fonction> ::=    MAX | MIN | MOY | SOM | CARD

<list-val>  ::=   <constante> [ , <suite-const> ]

<suite-const>         ::=   <constante> , <suite-const>

<param1> ::=  <entier>  |  <reel>
<param2> ::=  <entier>  |  <reel>
```

Consultation de la base :

```
<interrogation>       ::= <select1>
                          [ ORDERBY <ordre-att>  ]

<select1>  ::= SELECT <liste-exp-att>
                      FROM     <liste-rel>
                      [ WHERE <booleen>  ]

<select2>  ::= SELECT [ UNIQUE ] <att>
                         FROM     <liste-rel>
                      [ WHERE <booleen>  ]

<liste-exp-att>       ::=   [ UNIQUE  ]  *
                          | [ UNIQUE  ] <list-att>
                          | MAX ( [ UNIQUE ]   <att>   )
                          | MIN ( [ UNIQUE ]   <att>   )
```

```
                    | MOY ( [ UNIQUE ]  <att>  )
                    | SOM ( [ UNIQUE ]  <att>  )
                    | CARD ( [ UNIQUE ] * )
                    | CARD ( [ UNIQUE ] <list-att> )

<liste-rel>  ::=   <ident-rel>
                    | <identr-rel>.<var>,<identrel>.<var>

<booleen>  ::=    <fact-bool1>
                    |  <fact-bool1> <op-bool> <booleen>

<fact-bool1>        ::=    ( <booleen> )
                    |  <att>   <op>   <att>
                    |  <att>   <op>   <select1>
                    |  <att>   <op>   <constante>
                    |  SET <att> = <select2>

<ordre-att> ::= ORDERBY <list-att> ASC
                  | ORDERBY <list-att> DESC

<op>                ::=   = | < | <= | > | >= | <>
```

- REFERENCES ET BIBLIOGRAPHIE -

REFERENCES GENERALES

<ABR74> "Data semantics" By J.R. ABRIAL.
IFIP TC2 Working Conference, Cargèse, Avril 74.

<ADE82> "Bases de données et systèmes relationnels" par
C. DELOBEL et M. ADIBA.
DUNOD, Spécialité Informatique, PARIS. 1982.

<ADI78> "Un modèle relationnel et une architecture pour les
systèmes de bases de données réparties" par M. ADIBA
thèse de Docteur es Sciences. Septembre 78.
USMG-INPG GRENOBLE.

<ADI81> "Derived relations : a unified mechanism for views,
snapshots and distributed data" By M. ADIBA, Lab.
IMAG at GRENOBLE, in IEEE.

<ANS75> "Study group on data base monoprocessor systems :
interim report" ANSI (X3) SPARC
Bulletin of ACM SIGMOD 7, no 7, 1975.

<BAS81> "Update semantics of relational views" by F. BANCILHON
and N. SPYRATOS. INRIA, France.

<CHA76> "SEQUEL2:A unified approach to Data Definition,
Manipulation and Control" D.D. CHAMBERLIN and Al.
IBM Journal Research and Develop. November 1976.

<COD70> "A relational model of data for large shared data
banks", E.F. CODD, CACM June 70.

<COD72> "Relational completness of data sublanguages" E.F. CODD
in data base systems, Prentice Hall pp 65-98.

<COD79> "Extending Relational Model" by E.F. CODD, IBM Res.
Lab. San José California 95193.

<COD81> "The capabilities of relational database managment
sytems " by E.F. CODD IBM Res. Lab. San José
California 95193.

<DEL79> "Theoretical aspects of modeling in relational
data base" By C. DELOBEL, RR. Number 177, July 79,
IMAG, GRENOBLE.

<DEL80> "Les bases de données" par C. DELOBEL , Septembre

1980, IMAG GRENOBLE.

‹GAL81› "Impacts of logic on data bases" H. GALLAIRE, Lab. de Marcoussis, C.G.E. 91460 Marcoussis France.

‹ING76› "The design and implementation of INGRES" By M. STONEBRAKER, E. WONG and P. KREPS, Univ of California, ACMTODS Vol 1.Number 3, Sep 1976.

‹JOU79› "Qu'est-ce qu'une base de données cohérente ?" par M. JOUVE et C. PARENT, AFCET Journées de formation BASES DE DONNEES COHERENTES à Paris, Mai 1979.

‹KIM81› "On optimizing an SQL-like nested query" By W. KIM, IBM Res. Lab. San Jose, California 95193.

‹MIR80› "Architecture formelle d'un système d'intégrité pour une base de données réparties". Thèse d'état par S. MIRANDA, UPS Toulouse, Sep. 1980.

‹MIR82› "SGBD relationnels : Un survol et une proposition " S. MIRANDA, 82009-BDR100, Jan. 82, CERISS, UPS Toulouse.

‹MRD80› "MRDS : Multics Relational Data Store" Reference Manual, CII Honeywell Bull, 78430 Louveciennes, October 1980.

‹NAF79› "Protocole de transport pour réseaux locaux" N. NAFFAH, V. QUINT Document KAYAK REL 2.504

‹ROS80› "Processing conjunctive predicates and queries" D.J. ROSENKRANTZ and H.B. HUNT, Computer Science departement State university of New York at Albany, New York 12222.

‹SYS76› "System-R : Relational Approach to Database Managment" M.M. ASTRAHAN, M. W. BLASGEN, D.D. CHAMBERLIN, K.P. ESWARAN et Al. IBM Res. ACMTODS, vol 1, n 2, June 76.

‹WON76› "Decomposition- a strategy for query processing" By E. WONG and K. YOUSSEFI, ACMTODS, Vol 1, Number 3, September 1976.

REFERENCES D'INTEGRITE SEMANTIQUE

‹AND80› "L'intégrité et la mise à jour dans un système de gestion de base de données réparties." Projet POLYPHEME, J.M. ANDRADE, thèse de D.I. INPG Grenoble, Oct. 1980.

‹BAD78› "Data base system integrity" D.Z. BADAL, UCLA 1978 IEEE.

‹BAD79› "Semantic integrity, consistency and concurrency control in distributed database" by D.Z. BADAL Ph.D. Computer Science 1979 Univ. of California, Los Angeles.

‹BBC80› "Fast maintenance of semantic integrity assertions using redundant agregate data" By P.A. BERNSTEIN, B.T. BLAUSTEIN, E.M. CLARKE, in V.L.D.B. 1980.

‹BER81› "A simplification algorithm for integrity assertions and concrete views" by P.A. BERNSTEIN and B.T. BLAUSTEIN, Aiken Computation Lab. Harvard University Cambridge, MA 02138.

‹BER82› "Fast methods for testing quantified relational calculus assertions" P.H. BERNSTEIN, Aiken Computation Laboratory Harvard University Cambridge, MA 02138 , and B.T. BLAUSTEIN Computer Corporation of America 575 Technology Square Cambridge, MA 02139.

‹BLA81› "Enforcing database assertions : techniques and applications" By B.T. BLAUSTEIN, Ph. Doctoral at Harvard University, Center for Research in Computing technology, Cambridge, PA 02138.

‹BOU79› "Integrité sémantique d'une base de données" par D. BOULANGER. Univ Lyon 1. Afcet "Bases de données cohérentes" à l'IPP. Paris mai 79

‹BRO78› "Specification and verification of data base semantic integrity". By M.L. BRODIE, Technical report, April 78, Computer systems research group, University of Toronto, Canada.

‹BUN77› "Implementing alerting techniques in database systems" By O.P. BUNEMAN, H.L. MORGAN, Dept of Decision Science, The Wharton School, University of Pennsylvania, PA 19174.

‹BPO79› "Coste and performance analysis of semantic integrity validation methods" by D.Z. BADAL and G.J. POPEK. Computer Science Dept.,UCLA, Los Angeles,Ca 90024.

‹DAT81› "Referential integrity" C.J. DATE. IBM General products Division. 1981 IEEE.

‹ESW75› "Functional specification of subsystem for data base integrity" By K.P. ESWARAN, D.D. CHAMBERLIN. IBM Res. Rep. June 1975.

‹ESW76› "Specifications, implementation and interaction of a trigger subsystem in an integrated database system". K. ESWARAN. IBM Res. Lab. San José California.

‹FLO74› "Consistency auditing of databases" by J.J. FLORENTIN. Computer Journal 17, 2 (1974), pp 52-58.

‹FRT82› "ISIS : une méthode pour contrôler l'intégrité sémantique dans le SGBD relationnel MICROBE" par L. FERRAT, Séminaire bases de données, Nov. 82, Toulouse.

‹GAR79› "Proving consistency of data base transactions" G. GARDARIN and M. MELKANOFF, Proc. of the 5th Int. Conf. on VLDB, Tokyo, Sept. 1979.

‹HAM78› "Efficient Monitoring of database assertions". M. HAMMER, S. SARIN, ACM SIGMOD Int. Conference on Management of Data, June 1978.

‹HOM81› "Extending the use of agregate data in integrity constraint evaluation in a relational data base" by P.V. HOMEIER, March 1981.

‹JAR82› "Range nesting : a fast method to evaluate quantified queries" By M. JARKE NEW YORK University, NEW YORK 10006 and J. KOCH Universitat HAMBURG Schluterstr 70 D-2000 HAMBURG 13 R.F.A.

‹LAF79› "An approach to automatic maintenance of semantic integrity in large design data bases" By F. LAFUE, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

‹LAF82› "Semantic integrity dependencies and delayed integrity checking" By Gilles E.M. LAFUE, Computer Science Dept. Rutgers University, New Brunswick, N.J. 08903. In VLDB 82 at Mexico City.

<LIP79> "On semantic issues connected with incomplete information databases" By W. LIPSKI ACMTODS Vol 4, number 3, Sep. 1979.

<McL76> "High level expression of semantic integrity specification in a relational database system" D.J. McLEOD

<NIC79> "Contributions a l'étude théorique des bases de données - apports de la logique mathematique - par J.M. NICOLAS, Thèse d'état à l'université Paul Sabatier de Toulouse, Decembre 1979.

<NIC82a> "Logic for improving integrity checking in relational data bases" By J.M. NICOLAS, ONERA-CERT, Toulouse.

<NIC82b> "Les règles de cohérence ou contraintes d'intégrité" par J.M. NICOLAS, ONERA-CERT, Toulouse. Extrait du rapport BD3 par M. ADIBA et al, "Bases de données : nouvelles perspectives", 1982.

<RIF80> "Maintaining the Integrity of a Relational Database with assertion tests as specified by boolean expressions" By E.M. RIFKIN, Bell Laboratories Holmdel, New Jersey 07733.

<SAR77> "Automatic synthesis of efficient procedures for database integrity checking" By Sunil Kumar SARIN, Master of Science at Massachussetts Institue of Technology.September 1977.

<STO75> "Implementation of integrity constraints and views by query modification" By M. STONEBRAKER, Electronics Res. Report. Univ. of California Berkeley March 75.

<WAL81> "Automatic modification of transactions to preserve data base integrity without undoing updates" By A. WALKER and S.C. SALVETER, Dept of Computer Science University of NEW YORK at Stony Brook, NEW YORK 11794. (technical report 81/026, June 81).

<WEB76> "A semantic model of integrity constraints on relational database" By H. WEBER, Technische Universitat Berlin. 1976.

<WEB77> "The D-graph model of large shared data bases : a representation of integrity constraints and views

as abstracts data types" H. WEBER. IBM Research Lab. San José. California 95193.

‹WIL80› "A conceptual model for semantic integrity checking" By G.A. WILSON. Computer Corporation of America. 575 Technology Square Cambridge. MA 02139.

REFERENCES MICROBE

‹ACO82› "Gestion des contraintes dans un sous-système d'intégrité sémantique" Par F. ACOSTA. Rapport de DEA. INPG Lab. IMAG. Septembre 82.

‹AZR81› "MIDEC : un algorithme de décomposition de requêtes dynamique décentralisé" par F. AZROU. rapport DEA. INPG. Projet MICROBE. Lab. IMAG. Juin 81.

‹BEN81› "Conception d'un commutateur de processus réseau pour la mise en oeuvre d'applications réparties" par A. BENSAID. rapport de 3ème année CERI. lab. IMAG. Septembre 81.

‹CHA82› "INGRID : une interface relationnelle graphique" par T. CHAIX et J.C. GUILLAUME. Rapport de 3ème année Ensimag. INPG. Juin 82.

‹FER80› "MICROBE: Un système de bases de données relationnelle répartie sur un réseau local de micro-ordinateurs". F. FERNANDEZ, L. FERRAT,G.T. NGUYEN, G. SERGEANT. Congrès de l' AFCET Nancy (FRANCE). Novembre 1980.

‹FER81a› "Micro-mémoire relationnelle (MIMER)" F. FERNANDEZ Rapport de Recherche n0 233. lab IMAG Jan. 1981

‹FER81b› "La micro-mémoire relationnelle (MIMER) : Un outil pour la construction de SGBD relationnels" par F. FERNANDEZ. Projet MICROBE. Thèse de Docteur-ingénieur. Nov-1981. USMG GRENOBLE.

‹FRT80› "Traduction d'un langage relationnel de haut niveau en arborescences algébriques". L. FERRAT Projet Microbe Rap. de D.E.A Lab. IMAG Sep. 80

‹FRT81a› "Projet MICROBE: Guide utilisateur du langage MIQUEL". L. FERRAT. Note technique. Lab IMAG. Janvier 1981.

<FRT82a> "Projet MICROBE : du langage MIQUEL aux arborescences algébriques" par L. FERRAT, Note interne, Mai 81, Lab. IMAG, GRENOBLE.

<FRT82b> "L'intégrité sémantique dans les bases de données relationnelles" par L. FERRAT, Fevrier 1982, Projet MICROBE, Laboratoire IMAG.

<FRT82c> "INGRID, MIQUEL et PASREL : les 3 interfaces relationnelles de MICROBE" par L. FERRAT, J.C. GUILLAUME et M. MOUTAMANI, Séminaire de bases de données, Nov. 1982, Toulouse.

<GAL83> "Application de l'intelligence artificielle à l'optimisation des requêtes relationnelles" par H. GALY, Thèse de 3ème cycle, INPG, Lab. IMAG, à paraitre.

<LEE81> "MIOPE : Définition et réalisation d'un interpréteur de requêtes dans MICROBE" par Y.J. LEE, Rap. DEA. Lab. IMAG, Septembre 1981

<LEE83> "Le choix des chemins d'accès dans MICROBE" par Y.J. LEE Thèse de 3ème cycle, INPG, Laboratoire IMAG, à paraitre.

<NGU81a> "Relational query processing on a local network database system" By F. FERNANDEZ, L. FERRAT, H. GALY, G.T. NGUYEN. Lab. IMAG at GRENOBLE.

<NGU81b> "A high level user interface for a local network database system" By G.T. NGUYEN, L. FERRAT, H. GALY, RR 257, Juillet 81, Laboratoire IMAG, Université de GRENOBLE.

<NGU82> "MICROBE : Manuel de référence" par F. FERNANDEZ, L. FERRAT, Y.J. LEE et G.T. NGUYEN. Laboraroire IMAG, Janvier 82.

<NGU83> "MICROBE : un système de gestion de bases de données relationnelles" par G.T. NGUYEN, Y.J. LEE et P. WINNINGER, Journées Conception, implantation et utilisation de SGBD relationnels sur micro-ordinateurs Toulouse, Février 83.

<PRA82> "Sécurité de fonctionnement et reprise après panne d'une base de données relationnelles" par S. PRAWIROSUSANTO, rapport de DEA INPG, Projet MICROBE, Septembre 82.

<SAR81> "Les B*-arbres dans MIMER" par M. SARAJLIC et

RESUME : Cette thèse présente le sous-système ISIS, une méthode d'expression et de contrôle de l'intégrité sémantique dans les bases de données relationnelles. Celle-ci est régie par un ensemble de lois appelées contraintes d'intégrité sémantiques (CIS). D'abord, nous avons abordé les problèmes de classification et d'expression des CIS. Pour gérer ces CIS, nous les avons traduites en arborescences algébriques via un ensemble de règles prédéfinies. Une preuve sémantique de ces règles a été donnée. Ensuite, nous les avons stockées dans le catalogue de la base de données. Quand la base de données est mise à jour, des données peuvent être incohérentes, pour éviter cela nous avons élaboré une méthode de validation optimisée des opérations de mise à jour. Enfin, une réalisation est effectuée dans le cadre du projet MICROBE.

MOTS CLES : bases de données relationnelles, intégrit sémantique, algèbre relationnelle, arborescence algébrique, validatio des opérations de mise à jour.

F. FERNANDEZ, Mars 81, projet MICROBE.
Laboratoire IMAG.

<WIN82> "MIRES : un optimiseur de requêtes pour le SGBD relationnel MICROBE" par P. WINNINGER, rapport DEA, Lab. IMAG, Septembre 82.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

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

VU le rapport de présentation de
Monsieur DELOBEL, Professeur

Monsieur FERRAT LOUNAS

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR DE TROISIEME CYCLE, spécialité "Informatique".

Fait à Grenoble, le 06 mai 1983

Le Président de l'I.N.P.-G.

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

P.O. le Vice-Président,