Aspects dynamiques et gestion du temps dans les systèmes de bases de données généralisées

Ngoc Bui Quang

THESE

présentée par

BUI QUANG Ngoc

pour obtenir le titre de **DOCTEUR**

de **L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(arrêté ministériel du 5 juillet 1984)

Spécialité : *INFORMATIQUE*

ASPECTS DYNAMIQUES ET GESTION DU TEMPS DANS LES SYSTEMES DE BASES DE DONNEES GENERALISEES

Date de soutenance : *20 Novembre 1986*

Composition du jury :

Président	*J.Mossière*
Rapporteurs	*F.Bodart*
	C.Rolland
Examinateurs	*M.Adiba*
	M.Lopez

Thèse préparée au sein du Laboratoire de Genie Informatique
à l'Université Scientifique, Technologique et Médicale de Grenoble

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH
Vice-Présidents : B. BAUDELET
H. CHERADAME
R. CARRE
J.M. PIERRARD

Année universitaire 1984-1985

Professeurs des Universités

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

BESSON	Jean	LOUCHET	François
BONNETAIN	Lucien	PARIAUD	Jean-Charles
BONNIER	Etienne	RAMEAU	Jean-Jacques
DURAND	Françis	SOHM	Jean-Claude
GUYOT	Pierre	SOUQUET	Jean-Louis

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

BARIBAUD	Michel	GENTY	Pierre
BLIMAN	Samuel	GUERIN	Bernard
BUYLE BODIN	Maurice	POUPOT	Christian
CHENEVIER	Pierre	SERMET	Pierre
COHEN	Joseph	ZADWORNY	François
COUMES	André		

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

BARRAUD	Alain	JOUBERT	Jean-Claude
BAUDELET	Bernard	JOURDAIN	Geneviève
BLOCH	Daniel	LACOUME	Jean-Louis
BRISSONNEAU	Pierre	LONGEQUEUE	Jean-Pierre
CAVAIGNAC	Jean-François	MASSELOT	Christian
CHARTIER	Germain	MORET	Roger
CHERUY	Arlette	PAUTHENET	René
DURAND	Jean-Louis	PERRET	René
FELICI	Noël	PERRET	Robert
FOULARD	Claude	POLOUJADOFF	Michel
GAUBERT	Claude	SABONNADIERE	Jean-Claude
IVANES	Marcel	SCHLENKER	Claire
JALINIER	Jean-Michel	SCHLENKER	Michel
JAUSSAUD	Pierre		

E.N.S.H.G.

BOIS	Philippe	LESPINARD	Georges
BOUVARD	Maurice	MOREAU	René
LESIEUR	Marcel	PIAU	Jean-Michel

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

ANCEAU	François	MOSSIERE	Jacques
FONLUPT	Jean	ROBERT	François
LATOMBE	Jean-Claude	SAUCIER	Gabrielle
MAZARE	Guy	VEILLON	Gérard

U.E.R.M.C.P.P.

CHERADAME	Hervé	RENAUD	Maurice
CHIAVERINA	Jean	ROBERT	André
GANDINI	Alessandro	SILVY	Jacques

Professeurs Associés

BLACKWELDER	Ronald	ENSHG
HAYASHI	Hirashi	ENSIEG
PURDY	Gary	ENSEEG

Professeurs à l'Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S.

Directeurs de recherche :

CARRE	René
FRUCHARD	Robert
JORRAND	Philippe
VACHAUD	Georges

Maître de recherche :

ALLIBERT	Michel	JOUD	Jean-Charles
ANSARA	Ibrahim	KAMARINOS	Georges
ARMAND	Michel	KLEITZ	Michel
BINDER	Gilbert	LANDAU	Ioan-Dore
BORNARD	Guy	LASJAUNIAS	Jean-Claude
DAVID	René	MERMET	Jean
DESPORTES	Jacques	MUNIER	Jacques
DRIOLE	Jean	PIAU	Monique
GIGNOUX	Damien	PORTESEIL	Jean-Louis
GIVORD	Dominique	THOLENCE	Jean-Louis
GUELIN	Pierre	VERDILLON	André
HOPFINGER	Emile	SUERY	Michel

Personnalités habilitées à diriger des travaux de recherche
(Décision du conseil scientifique)

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

ALLIBERT	Colette	HAMMOU	Abdelkader
BERNARD	Claude	MALMEJAC	Yves (CENG)
BONNET	Roland	MARTIN GARIN	Régina
CAILLET	Marcel	NGUYEN TRUONG	Bernadette
CHATILLON	Catherine	RAVAINE	Denis
CHATILLON	Christian	SAINFORT	(CENG)
COULON	Michel	SARRAZIN	Pierre
DIARD	Jean-Paul	SIMON	Jean-Paul
EUSTATHOPOULOS	Nicolas	TOUZAIN	Philippe
FOSTER	Panayotis	URBAIN	Georges(ODEILLO)
GALERIE	Alain		

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

BARIBAUD	Michel	DOLMAZON	Jean-Marc
BOREL	Joseph	HERAULT	Jeanny
CHOVET	Alain	MONLLOR	Christian
CHEHIKIAN	Alain		

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

BORNARD	Guy	LEJEUNE	Gérard
DESCHIZEAUX	Pierre	MAZUER	Jean
GLANGEAUD	François	PERARD	Jacques
KOFMAN	Walter	REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine	OBLED	Charles
BOIS	Daniel	ROWE	Alain
DARVE	Félix	VAUCLIN	Michel
MICHEL	Jean-Marie	WACK	Bernard

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

BERT	Didier	DELLA DORA	Jean
CALMET	Jacques	FONLUPT	Jean
COURTIN	Jacques	SIFAKIS	Joseph
COURTOIS	Bernard		

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean	NIFENECKER	Hervé
COEURE	Philippe (LETI)	PERROUD	Paul
DELHAYE	Jean-Marc (STT)	PEUZIN	Jean-Claude(LETI)
DUPUY	Michel (LETI)	TAIEB	Maurice
JOUVE	Hubert (LETI)	VINCENDON	Marc
NICOLAU	Yvan (LETI)		

Laboratoires extérieurs

C.N.E.T.

DEMOULIN	Eric	MERCKEL	Gérard
DEVINE	R.A.B.	PAULEAU	Yves
GERBER	Roland		

I.N.S.A. Lyon

GAUBERT	C.

****

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: M. M. MERMET
Directeur des Etudes et de la formation	: M. J. LEVASSEUR
Directeur des Recherches	: M. J. LEVY
Secrétaire Général	: Mle M. CLERGUE

Professeurs de la 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Metallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique-Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

Je tiens à remercier

Monsieur Jacques Mossière, Professeur à l'Institut National Polytechnique de Grenoble, Directeur du Laboratoire de Genie Informatique à l'IMAG pour l'intérêt qu'il a témoigné à mes études doctorales et de me faire l'honneur de présider le jury de cette thèse.

Monsieur Michel Adiba, Professeur à l'Université Scientifique, Technologique et Médicale de Grenoble, qui est à l'origine et Directeur de cette thèse, pour m'avoir accueilli au sein de son équipe de recherche et m'avoir proposé le sujet de cette thèse. Sa totale disponibilité à mon égard, sa grande expérience, ses nombreux conseils et encouragements représentent sans nul doute un facteur décisif dans l'aboutissement de ce travail.

Monsieur François Bodart, Professeur aux Facultés Universitaires Notre-Dame de la Paix, Namur, Belgique et Madame Colette Rolland, Professeur à l'Université de Paris I, d'avoir bien voulu juger mon travail et participer à ce jury.

Monsieur Mauricio Lopez, Ingénieur du Centre de Recherche BULL, pour l'intérêt qu'il a porté à ce travail et pour sa participation au jury de ma thèse.

Monsieur José Palazzo Moreira de Oliveira, actuellement enseignant à l'Université de Porto Allegre (Brésil), pour son aide efficace à la réalisation de mon DEA, et aussi pour sa première idée de l'extension du modèle TIGRE pour le temps.

Mes collègues du projet TIGRE, Christine Collet, Fernando Velez, pour l'intérêt qu'ils ont porté à ce travail et pour les nombreuses discussions qui en ont découlé.

Mes collègues de l'équipe Bases de Données, Monique Chabre-Peccoud, Jean-Pierre Giraudin, Gia Toan Nguyen et Marie-Christine Fauvet, ainsi qu'Olivier Oudot et Jean-Michel Hufflen pour la patience de corriger le manuscrit de cette thèse.

Tous mes amis vietnamiens et français grenoblois pour leur amitié aussi bien à l'intérieur qu'à l'extérieur de l'environnement de travail.

Les membres du service reprographie pour le soin et l'efficacité qu'ils ont apporté au tirage de cette thèse.

Il me faut également remercier l'Institut National Polytechnique de Grenoble et l'Institut Polytechnique de Hanoï de m'avoir accordé une bourse d'études de 3e cycle dans le cadre de la Coopération entre l'INPG et les Instituts Polytechniques du Vietnam.

Cuối cùng tôi không thể nhắc tới ở đây lòng biết ơn vô hạn đối với vợ tôi, Minh Nguyệt, và con tôi, Nguyệt Minh, những người đã gánh chịu sự xa cách trong thời gian qua và luôn luôn là niềm động viên tinh thần bất tận từ đất Việt xa xôi của tôi trong thời gian tôi chuẩn bị luận án này.

A ma femme,

à ma fille

RESUME

Nous introduisons ici la gestion du temps et des historiques dans les systèmes de de bases de données généralisées où les données à manipuler sont des documents multimédia. Différentes notions liées au temps sont incorporées dans le modèle de données généralisées TIGRE pour décrire les aspects dynamiques des objets de la base. Les types de temps, les fonctions et les opérations associées, ainsi que le concept de versions temporelles et la structure d'historiques sont proposés. Le langage LAMBDA associé au modèle de données TIGRE est étendu pour manipuler les données temporelles et les données historiques.

Nous décrivons ensuite des applications du concept de versions temporelles sur différentes classes d'objets : les photographies et le schéma conceptuel d'une base de données TIGRE. On propose alors les notions de photographie dynamique, d'album et de film.

Nous étudions également l'architecture du SGBD TIGRE pour la prise en compte de la gestion du temps et des historiques.

Mots-clés :

Donnée généralisée, temps, donnée historique, version temporelle, langage de manipulation de données, photographie dynamique, schéma conceptuel, zone d'historiques.

TABLE DES MATIERES

INTRODUCTION

Face à l'évolution des besoins en matière de gestion de données, les recherches d'aujourd'hui envigagent de développer pour les SGBD des nouvelles fonctionnalités satisfaisant de nouvelles applications informatiques, par exemple la bureautique, où l'on manipule des documents multimédia, la CAO qui traite des grands volumes d'information à nature et à structure très diverses, ou des bases de données déductives, voire des bases de données intélligentes. Ces nouveaux SGBD sont développés autour des SGBD relationnels qui ont marqué une stabilisation vers le début des années 80.

A notre connaissance à l'heure actuelle la plupart des SGBD ne sont capables que de fournir des versions *les plus récentes* et *les plus cohérentes des données*. Pour de nombreuses applications cette limitation n'est pas acceptable. De plus en plus la gestion des historiques ou de l'ensemble des anciennes valeurs de certaines informations est exigée. Ces dernières sont en grande partie associées à la notion du temps dont le rôle dans certaines domaines d'application, comme la bureautique, est de plus en plus justifié. Ce concept de temps est nécessaire pour spécifier le temps de création ou le temps de vie d'un document, pour contrôler et coordonner des activités d'une opération bureautique. Le besoin de la gestion du temps dans les bases de données a été mis en évidence depuis longtemps [BUBE 77], [WIED 75]. Bon nombre de travaux de recherche en la matière sont apparus dans la littérature et une première synthèse [BOLO 82] présente 68 travaux sur le temps entre 1960 et 1982, dont la moitié ont été publiés après 1977.

Ces efforts sont consacrés principalement à l'incorporation de la notion de temps dans les bases de données, à l'aide d'un modèle de donnée, dit *modèle temporel*. Selon le degré d'intégration du temps dans le système on peut constater :

- La manipulation du temps est faite de façon explicite par des programmes d'application [WIED 75].

- Des extensions pour la manipulation du temps par des programmes dits "Experts" sont proposées : "Time expert" du système INGRES [OVER 82].

- Le concept de temps est incorporé directement au modèle de données comme dans le modèle TERM [KLOP 81].

Outre la sémantique du temps et son incorporation dans les bases de données, le *langage de manipulation* des données temporelles et *l'interface avec l'usager* doivent être pris en compte dans la conception de systèmes de gestion de bases de données [AVRI 83].

C'est dans cette ligne que ce travail a été effectué dans le cadre du projet TIGRE (Traitement d'Informations Généralisées REparties) mené conjointement par le Centre de Recherche BULL et le Laboratoire de Génie Informatque de l'IMAG. Plus précisement nous avons étendu le modèle de données généralisées TIGRE et le langage associé, LAMBDA, pour inclure la notions de temps et la notion d'historiques. D'une part, les différents types de temps (*temps simple, restreints, intervalle, durée et période*) sont définis comme types de base du modèle TIGRE. D'autre part, les fonctions et les opérations sur le temps peuvent être directement manipulées depuis LAMBDA. Nous avons ensuite introduit la notion de versions temporelles avec différents types : *version manuelle, version périodique et version successive* et celles-ci sont appliquées à des documents généralisés dans un modèle sémantique tel que TIGRE. Ainsi un type TIGRE peut être caractérisé par deux structures : *structure de données et structure d'historiques*. De cette façon le schéma de la base contient la représentation explicite du temps et de différents concepts associés comme *les historiques, les versions et les photographies* (vue sur une base de données figée à un instant). Dans cette approche, la sémantique du temps fait partie intégrante du modèle.

Nous avons appliqué la notion d'historique aux *photographies* et au *schéma conceptuel*. Ceci nous permet de définir la notion *d'album* et de *film*, comme la suite des vues et des photographies sur une base de données.

L'organisation de cette thèse est la suivante :

Dans le chapitre 1 nous présentons les travaux les plus connus concernant la notion de temps dans les bases de données. Le chapitre 2 présente une description générale du projet TIGRE, dans lequel ce travail est effectué.

Au chapitre 3 nous présentons l'extension du modèle de données TIGRE pour la notion de temps. Les types de temps, les fonctions et les opérations sur le temps sont introduits. Les clauses temporelles sont également introduites dans le langage LAMBDA ce qui permet de manipuler les données temporelles. La manipulation est illustrée par la gestion d'une base de données universitaire.

Le chapitre 4 présente notre approche pour les versions temporelles d'objets. La structure d'un type TIGRE est caractérisée par deux aspects : structure de données et structure d'historiques. Trois types de versions sont considérés : manuelle, périodique et successive. Le concept de versions temporelles est appliqué au modèle sémantique et généralisé TIGRE. LAMBDA est étendu pour manipuler les différentes versions d'objets de la base.

Dans le chapitre 5 nous appliquons la structure d'historiques aux photographies. D'une part les photographies sont définies comme un type (dérivé) de données du modèle TIGRE. D'autre part les photographies dynamiques sont des photographies pour lesquelles la structure d'historiques n'est pas vide. Ainsi les notions d'album et de film sont définies. Nous étudions aussi dans le chapitre 5 le problème lié au rafraîchissement des films.

Le chapitre 6 présente notre application de versions temporelles sur le schéma d'une base de données TIGRE afin de maintenir l'historicité du schéma conceptuel.

Enfin, dans le chapitre 7 nous étudions l'architecture du SGBD TIGRE pour la prise en compte de la gestion du temps et des historiques.

CHAPITRE 1

GESTION DU TEMPS DANS LES BASES DE DONNEES : ETAT DE L'ART

1- Modèle temporel

Le besoin de la gestion du temps dans les bases de données a été mis en évidence depuis longtemps [BUBE 77], [WIED 75]. Bon nombre de travaux de recherche en la matière sont apparus dans la littérature et la première synthèse [BOLO 82] présente 68 travaux sur le temps entre 1960 et 1982, dont la moitié a été publiées après 1977. Ces efforts sont consacrés à l'incorporation de la notion de temps dans les bases de données, basée sur un modèle de données, dit **modèle temporel**.

Un modèle est dit temporel si la *sémantique du temps* y est incluse. La sémantique du temps peut intervenir sous différentes formes, à différents niveaux, mais elle est explicitement représentée dans le modèle.

Quel que soit le domaine d'applications, il faut bien distinguer chaque aspect du modèle temporel :

1) Modèle du temps (modéliser le temps lui-même)

2) Modèle d'objets temporels

3) Modèle de systèmes temporels

Sur l'aspect du modèle du temps on s'intéresse au rôle du temps dans des systèmes d'information, à la représentation du temps dans ces systèmes et à sa sémantique, ainsi qu'à la manipulation du temps à travers sa structure et ses opérations.

Le mot "objet" peut désigner des choses comme: état, événement, entité, association, processus...etc. Un objet temporel est celui qui comprend le temps dans ses descriptions ou bien le temps est un composant qui constitue l'objet. Par exemple une personne est caractérisée par sa date de naissance, par son travail, qui est à son tour caractérisé par le contrat avec la date de signature, avec la durée de validité du contrat,..etc.

Chaque objet est spécifié par ses états et ses événements. Un événement désigne un changement entre états. Une entité, une association et un processus peuvent comprendre des composants qui sont eux-même des objets.

Sur l'aspect du modèle d'objets temporels on s'intéresse aux états des entités du monde réel, aux associations qui lient des entités. On s'intéresse aussi aux événements et aux processus comprenant ces entités et associations, au temps de ces états et ces événements, aux changements des états via des événements et des processus.

Quant à l'aspect du modèle de systèmes temporels, on s'intéresse aux états d'un système d'information (qui stocke en général des informations temporelles), aux changements d'états par des processus internes et par des événements externes (usagers par exemple), c'est-à-dire par des transactions.

Récemment, plusieurs recherches se sont orientées vers la sémantique du temps dans les bases de données. Elles ont montré comment sa gestion peut être incorporée dans un SGBD, quel langage de manipulation de données temporelles et quel interface peuvent être fournis à l'usager [AVRI 83].

Dans ce chapitre on envisage de faire une synthèse des travaux publiés sur ces aspects pour mieux situer les travaux présentés dans cette thèse.

2- Modélisation du temps

2.1- La vue continue du temps

Le temps peut être vu comme isomorphe aux nombres réels [PALA 84]. Cette approche est prise pour la plupart des problèmes physiques. Pour donner la représentation d'un certain point du temps, la solution naturelle est de définir un instant de référence et une unité de temps - souvent une seconde, une minute ou bien une heure. Chaque événement peut être associé à un point du temps, l'interprétation de cette association est : l'événement se produit à X unités de temps à partir de l'instant de référence.

Si cette solution est très bien adaptée aux problèmes physiques, elle ne l'est pas aux problèmes de gestion. D'une part la précision d'un événement de gestion n'est pas si fine que celle des problèmes physiques. D'autre part un événement de gestion se déroule à un point d'un intervalle quelconque, par exemple dans la journée du 15/06/1986, dans le mois de Mars/1986, etc. Pour cette classe d'application on emploie une représentation discrète : un *calendrier*, qui est présenté dans le paragraphe suivant.

2.2- Calendrier : représentation vectorielle du temps

Dand ce modèle, on s'intéresse d'abord à un ensemble fini de type intervalle de temps Tl = {Tl1,..,Tln}, par exemple Tl = **{seconde, minute,.., année}**. Le type mois signifie par exemple l'ensemble des toutes les périodes mensuelles telle que "Mars 1986". Les Tl1, .., Tln sont souvent des intervalles augmentés de temps, c'est aussi une séquence de domaines [BREU 79], [BUBE 80].

S'il y a n domaines T1, T2,.., Tn alors un vecteur $(t_k,..,t_n)$ où $1 <= k <= n$ et t_j - un entier pour le j-ème domaine, représente un point généralisé de temps, par exemple (05,1985). Le début et la fin d'un tel point généralisé de temps appartiennent au domaine plus fin suivant.

On peut citer ci-dessous les ensembles d'intervalles de temps TI employés dans la littérature :

- *(jour, mois, année)* dans TOD [WIED 75] et dans TERM [KLOP 81].

- *(semaine, seconde, minute, heure, jour, mois, année)* dans

"Time expert" de l'INGRES [OVER 82].

- *(semaine, minute, heure, jour, mois, année)* dans TSOS [BARB 85].

Avec l'introduction du concept de calendrier des problèmes se posent sur la manipulation des intervalles car les opérations de manipulation seront parfois définies sur des intervalles de granularité différente.

2.3- Abstraction du temps absolu et du temps relatif [BOLO 83]

On va rappeler une autre approche de temps présentée dans [BOLO 83] : **abstraction absolue et abstraction relative** du temps. La vision du temps comme séquence linéaire de points est appelée le temps absolu. Cette vision est adoptée dans la plupart des systèmes d'information où les informations temporelles sont représentées à l'aide du calendrier ou des états de l'horloge du système. Pour la comparaison, les dates et le temps d'horloge sont souvent convertis en entier représentant le nombre d'unités de temps par rapport à un temps original, par exemple 1/Janvier/1900. Dans cette vision du temps les dates et le temps d'horloge sont appelés les points du temps absolu. L'usage du temps absolu est répandu dans les applications scientifiques et administratives où l'information temporelle est précise et disponible pour chaque enregistrement.

Mais, dans la plupart des cas, l'information temporelle est représentée par l'association temporelle dans l'ordre **"avant" et "après"**. Exemple : la fin d'une activité sera 5 jours après son début. La vision du temps comme ensemble de telles associations temporelles entre les événements est appelée le temps relatif.

Les auteurs de [BOLO 83] font deux remarques pour cette dichotomie: temps absolu et temps relatif. Premièrement, cette dichotomie mérite une attention spéciale pour la sémantique du niveau conceptuel de données. Deuxièment, il est préférable que la théorie

de temps soit basée sur la vision relative de temps. Le temps lui-même n'est pas absolument observable, il est souvent observé par les horloges ou par les événements célestes particuliers.

Pour la théorie de base du temps, il faut faire attention aux deux relations atomiques "avant" et "pendant". La théorie du temps doit inclure des axiomes temporels comme propriétés invariantes générales puis considérer des spécifications temporelles pour le cas particulier. Les axiomes suivants sont formulés:

-pendant(x,x)
-(pendant(x,y) & pendant(y,z)) => pendant(x,z)
-(avant(x,y) & avant(y,z)) => avant(x,z)
-avant(x,y) => non avant(y,x)
-(avant(x,y) & pendant(z,y)) => avant(x,z)
-(pendant(x,y) & avant(y,z)) => avant(x,z)

Les autres relations temporelles peuvent être considérées comme dérivées de ces deux relations. Par exemple : *même-temps(x,y) = pendant(x,y) & pendant(y,x).*

On peut constater que l'ensemble des intervalles de dans les réelle donne un domaine abstrait de l'interprétation pour cette théorie avec les relations " <" et "<=" qui signifient respectivement "avant" et "pendant". Les relations "avant" et "pendant" liant deux objets temporels si et seulement si les relations "<" et "<=" ont lieu entre les intervalles assignés à ces objets dans cette interpétation.

2.4- "Time generic" de L.Anderson [ANDE 81, ANDE 82]

Au niveau conceptuel, le concept "Time generic" proposé par Anderson est l'un des plus clairs.

Dans cette approche les propositions suivantes sont formulées :

- Le temps, dans un contexte particulier, consiste en un ensemble d'éléments temporels indivisibles et discrets. On suppose que Tc est la catégorie dont l'extension est cet ensemble pour un contexte c quelconque. Le catégorie des points de temps dans le temps générique représente la génération de toute la catégorie Tc.

- La relation < donne une relation d'ordre totale aux objets de cette catégorie Tc. La génération de cette relation est la relation "avant".

-Chaque élément temporel de Tc possède une durée. Ainsi il existe une fonction *d* qui applique chaque élément de Tc à un nombre réel positif. La génération de toute durée est alors la durée pour tous les points de temps

-Il existe une *métrique m* qui applique des paires d'éléments de Tc aux nombres réels positifs. La quantité de temps entre deux éléments de Tc est la somme de toutes les durées des éléments entre eux. La génération dans le temps générique de cette *métrique m* est la relation "*métrique*".

-Il existe une relation temporelle "now" qui correspond à "regarder l'horloge" dans un contexte particulier ou encore "temps courant" de ce contexte.

-Un intervalle est un ensemble connexe des éléments de Tc y compris les deux bornes de l'intervalle. Il ne manque aucun élément ordonné entre deux bornes. On suppose que une borne peut être "now" ou "unknown".

-Une période du temps générique est décrite par trois caractéristiques : la durée de répétition, la durée entre deux répetitions consécutives et l'intervalle qui couvre toutes les répétitions.

La catégorie de temps est alors représentée comme union de tous les points de temps, tous les intervalles, et toutes les periodes de temps. Dans cette catégorie on peut définir les relations "*avant*" et "*égal*". Cependant L.Anderson permet de comparer un intervalle et une période par la relation "avant", ce qui nous semble insensé dans la sémantique du temps car une période est en effet un ensemble d'intervalles.

Il faut souligner que pour des applications particulières, on décrit d'abord une solution temporelle, par exemple le calendrier grégorien, et les autres notions du temps comme intervalle de temps, période et métrique du temps sont construites d'après cette solution.

3- Le temps et les bases de données

Dans le modèle relationnel de CODD [CODD 70], une relation d'une base de données est définie comme une représentation (état) d'un phénomène du monde réel qui évolue au cours du temps. Cependant la plupart des SGBD actuels ne sont capables de fournir pour un objet du monde réel, que l'état le plus récent. Cette limitation n'est plus acceptable pour de nombreuses applications où l'on s'intéresse également aux états antérieurs, par exemple :

"Le salaire de l'employé X a-t-il augmenté durant ces 5 dernières années?"

Pour résoudre les problèmes liés à différents états du monde réel, le concept de temps a été introduit dans les bases de données. Le temps constitue la troisième dimension du modèle relationnel. A chaque événement on associe une valeur de temps, par exemple : le salaire de X est 15K à partir de Mars/1985. Pour intégrer le temps dans le modèle relationnel, les relations représentant les données temporelles doivent contenir un (ou plusieurs) attribut - dit **attribut de temps**. Cependant le temps peut s'intégrer à un modèle de données de différentes manières selon sa sémantique. De plus, le temps peut être associé à un événement de deux façons : le moment où l'événement se déroule dans le monde réel et le moment où cet événement (ou plutôt ses données) est enregistré dans la base. Dans certains cas ces deux temps peuvent être le même. Exemple : l'enregistrement des signaux dans un système en temps réel. Dans d'autres cas ils peuvent être différents. Exemple : un employé a changé son adresse le 20/12/1985 mais ce changement n'est enregistré dans la base que le 25/12/1985, c-à-d cinq jours après.

Dans la littérature, le premier temps est par convention appelé **temps logique** et le deuxième - **temps physique**. Ces termes ne sont pas respectés par tous, certains les appelent temps valide, temps effectif ou temps d'événement (pour temps logique) et temps de transaction (pour temps physique). Dans cette thèse nous adoptons les termes temps physique et temps logique tels qu'ils sont définis ci-dessus.

La sémantique du temps logique et du temps physique associés aux données temporelles est caractérisée dans [SNOD 85] :

(1) Le temps logique correspond au monde réel et le temps physique à la représentation de ce monde dans la base. Ils correspondent donc à la **réalité** et à la **représentation**.

(2) Ils sont distingués par la **flexibilité de mise à jour**. Le temps logique peut être modifié lors de la perception de divergence entre les données. Au contraire, le temps physique, une fois enregistré, n'est modifié en aucun cas, il est donc seulement inséré.

(3) En ce qui concerne la **dépendance d'application**. Le temps logique est caractérisé comme dépendant de l'application. C'est l'usager qui le définit et le manipule. Il assure aussi le contrôle d'intégrité (à l'intérieur de son application) du temps logique. Quant au temps physique, il est indépendant de l'application, généré et contrôlé automatiquement par le SGBD. Donc il a une sémantique simple.

3.1- Le temps et les programmes d'applications [WIED 75]

L'un des premiers systèmes d'information temporelle est "Time Oriented Database - TOD" [WIED 75]. Ce système stocke et manipule des informations médicales associées au temps. Le temps dans TOD est donc un temps logique.

Dans TOD les entités sont décrites par des attributs statiques et dynamiques. Un attribut statique est invariable par rapport au temps, par exemple la date de naissance d'une personne. Les attributs dynamiques sont variables comme une adresse qui peut changer plusieurs fois dans la vie d'une personne. Ils changent au cours du temps et les valeurs peuvent concerner les applications et quelques unes doivent être stockés dans la base de données. Les informations temporelles sont alors vues comme une matrice à trois dimensions: entité, attribut et le temps. Le modèle du temps de TOD est un modèle temporel discret, chaque n-uplet a un attribut temporel désignant le temps où un ensemble de propriétés est observé pour une entité.

Le type de données DATE est défini comme une chaine numérique DDMONYY pour présenter le temps dans chaque enregistrement où DD-deux chiffres pour le jour, MON pour le mois (les trois caractères initiales du mot mois en anglais) et YY deux chiffres pour l'année.

Il faut remarquer que la manipulation du temps est faite d'une façon explicite par les programmes d'application (écrits en PL/1). Le SGBD ou plus généralement le système de gestion de fichiers ne peut que stocker des données de type traditionnel comme des entiers, des réels ou des chaînes de caractères. Toute sémantique associée au temps appartient à la structure logique des programmes. A ce niveau l'usager doit connaître la sémantique associée au temps et assurer la validité des opérations définies sur ce type. L'intégrité de la base est aussi la responsabilité de l'usager.

3.2- "Time expert" du système INGRES [OVER 82]

Dans les programmes d'applications, en plus des types traditionnels comme entier, réel, caractère ou chaîne de caractères, plusieurs types de données sont utilisés et partagés par les usagers. Afin d'aider les usagers à manipuler de données avec une sémantique étendue aux types des données de base, des programmes dits "expert" sont réalisés (voir [STON 80]). Le "time-expert" du système INGRES et sa mise en œuvre sont décrits dans [OVER 82]. Pour l'usager, la sémantique d'un attribut défini sur un type étendu, tel que le temps, paraît appartenir à la définition de la base. Un expert peut être vu comme une

procédure de contrôle et un ensemble de procédures spécialisées dans le traitement d'un type particulier de données. Cette méthode a pour caractéristiques principales l'indépendance entre le SGBD et les experts. Le SGBD n'a pas besoin de connaître la façon de traiter le type de données, ou la sémantique contenue d'un expert. Pour réaliser "Time expert" le SGBD INGRES ne doit être modifié ou additionné que dans 62 lignes de code.

Un expert est associé à un attribut pendant la définition d'une relation. Pour permettre cette définition, la syntaxe de la commande CREATE a été modifiée de la façon suivante :

CREATE `rel-name((field-name = format; expert-name)...)`

Cette syntaxe donne la possibilité de définir un attribut sur un nom d'expert en plus des types de base. L'effet de cette commande est la création d'une relation de nom rel-name et l'indication que l'attribut field-name est associé à l'expert de nom expert-name. Un expert est considéré comme une procédure reconnue par le SGBD et qui peut traiter quatre sortes d'appels:

a) "*Expert-field operator value*"

Exemple: RANGE OF E IS EVENT
REPLACE(E.time = "present-time")
WHERE E.name = "e1"

b) "*Value-1* `comparaison-operator` *value-2*"

La réponse sera une valeur qui appartient à l'ensemble (true, maybe, false)

c) "*expert-field* `arithmetic-operator` *constant*"

Exemple REPLACE (E.time = E.time + "1:30")
WHERE E.name = "lunch"

d) "*external-representation* <--------- *value*"

Ce type de requête produit une chaîne de caractères avec la représentation externe (pour l'usager) de la valeur du temps. Par exemple :

RETRIVE (E.time)
WHERE E.name = "lunch"

Les formats externes acceptés par le time-expert sont :

1) *Dates simples* : Dimanche Avril 8 16:30:00 1981

2) *Période de date* : jeudi 18 novembre 17 - vendredi 18 novembre 18:30

3) *Intervalle de temps relatif* : Il est possible de définir, également des périodes relatives à l'instant présent. Par exemple "aujourd'hui", "demain" "l'année dernière" ou "dans 2 semaines à partir d'aujourd'hui"

4) *Intervalle de temps* : "7 jours" , "3 années"

5) *Intervalle répétitif* : "lundi" , "vendredi", "12-13:30"

3.3- Intégration directe du concept de temps au modèle de données : modèle TERM

Dans cette solution, la sémantique du temps devient structurelle, elle appartient au modèle de données. La définition d'un schéma contient la représentation explicite du temps et des concepts associés tels que des séries historiques, versions et photographies. L'inconvénient de cette solution est le besoin de modifier le logiciel d'un SGBD pour qu'il accepte le nouveau type de données et la sémantique associée. Dans le cadre de développement d'un nouveau système de gestion de bases de données, cette solution est la plus envisageable.

Dans TERM (Time Extented ERM) [KLOP 81] le type de temps DATE est un type de base du modèle de données. Ce dernier est une extension du modèle Entité-Relation de [CHEN 76]. Une entité ou association sont décrites par une assertion d'existence et par des attributs et des rôles. Ils peuvent être constants ou variables (statiques ou dynamiques) par rapport au temps. Trois structures sont considérées : structure de temps, structure de valeur et structure historique. Un historique est vue comme un ensemble du type (valeur, temps), qui sont appelés les états de l'historique.

Deux mécanismes : dérivation et approximation sont proposés pour retrouver l'état d'un historique selon la possibilité de retrouver un résultat exact ou non. Si une approximation est appliquée elle doit être explicitement selectionnée, autrement la dérivation est appliquée par défaut.

Dans [KLOP 83] la gestion de l'incertitude est basée sur la logique ternaire (vrai, faux, inconnu). Ceci est important car dans la gestion des données temporelles la valeur inconnue est souvent rencontrée. L'ensemble des valeurs d'attribut est étendu par l'addition des valeurs nondéfinies : *nil, incertain , inconnu*. Pour les détails voir [KLOP 83].

3.4- Intégration des données temporelles à un modèle statique : attribut de temps

Dans le travail [CLIF 83], la sémantique du modèle Entité-Relation a été étendue au moyen de la sémantique du temps sous forme de la logique intentionnelle Π_S. Cette logique sert à la sémantique du temps de même que la logique du premier ordre sert au modèle relationnel (statique). Les relations complétées sont introduites dans ce modèle (Voir la figure 1.1). Deux nouveaux attributs sont ajoutés à chaque relation : attribut ETAT et attribut EXISTER. Alors la relation EMPLOYE peut être vue comme le montre la figure 1.1.

ETAT	EMPLOYE	EXISTER	DEPARTM	SALAIRE
1984	Marcheteau	1	B	10 K
1984	Dupond	1	A	12 K
1984	Spyrator	0	Nil	Nil
1984	Berman	0	Nil	Nil
1984	Savio	1	A	11 K
1985	Dupont	1	A	12 K
1985	Spyrator	1	B	15 K
1985	Berman	1	A	10 K
1985	Savio	0	Nil	Nil
1985	Marcheteau	0	Nil	Nil

Figure 1.1 : Relation complétée EMPLOYE avec deux attributs de plus : ETAT et EXISTER.

Une relation de la base est alors une collection de tous les n-uplets à tous les états. Pour un n-uplet qui n'existe pas à l'état E, toutes les valeurs non-clé sont nulle.

Dans une autre approche (prise par la plupart de travaux sur le temps) le modèle relationnel statique de CODD est employé comme modèle de base pour les données temporelles. Chaque relation temporelle est incorporée dans une relation statique contenant de plus un attribut(s) temporel(s), nommé **attribut de temps**.

Dans ce cas, la sémantique de modèle de données ne contient pas explicitement la sémantique du temps. Le langage de manipulation doit traduire les requêtes contenant le temps en requêtes équivalentes sur les relations statiques du modèle de base.

A notre connaissance, la plupart des travaux concernent la gestion du temps logique. L'attribut de temps peut contenir le temps logique à partir duquel le n-uplet est valide [CLIF 83]. Une autre manière consiste à définir un attribut contenant le debut et la fin de l'intervalle durant lequel le n-uplet est valide, cet attribut étant de type intervalle. Dans [SNOD 84, SNOD 85] deux attributs de temps sont utilisés, l'un pour représenter des intervalles du temps logique (temps valide) et l'autre pour des intervalles du temps physique (temps de transaction), voir la figure 1.2. Cependant l'usage du temps physique n'est pas indiqué dans [SNOD 84, SNOD 85]. D'autre part Snodgrass suppose que le temps valide (le cas particulier du temps logique) est automatiquement géré par le SGBD. Ceci n'est pas tout à fait naturel car le temps valide est dépendant des applications. De plus le temps valide n'est pas forcément représenté dans le schéma de données.

Nom	Salaire	Temps valide	Temps de transaction
Dupond	15 K	10/83 - 02/85	15/11/83 - 7/2/85
Berman	12 K	08/82 - 04/84	17/7/82 - 3/5/84
Dupond	17 K	03/85 - 04/86	7/2/85 - 14/4/86
Berman	14 K	05/84 - now	3/5/84 -
Dupond	16 K	05/86 - now	14/4/86 -

Figure 1.2 : Relation EMPSAL avec deux attributs de temps pour le temps valide et le temps de transaction.

Il faut noter que les relations temporelles ci-dessus sont des relations en première forme normale. Dans cette classe de relations il y a nettement une redondance de données. Les valeurs des attributs statiques sont présentées dans tous les n-uplets, par exemple les noms d'employés de la relation EMPLOYE ci-dessus. Pour éviter cet inconvénient le temps peut être introduit à l'intérieur des attributs dynamiques (attributs qui peuvent prendre des différentes valeurs au cours du temps) [GADI 85], [CLIF 85]. On peut illustrer ceci par la figure 1.3.

Nom	Département	Salaire
Dupond	A [9/71-12/76]	10 K [9/71-2/75]
[9/71-now]	B [12/76-now]	12 K [3/75-8/80]
		14 K [9/80-now]
Berman	B [6/78-10/80]	8 K [6/78-10/80]
[6/78-now]	C [11/80-now]	10 K [11/80-now]

Figure 1.3 : Relation EMPLOYE avec le temps (logique) à l'intérieur des attributs dynamiques.

Une algèbre étendue pour la gestion du temps dans les relations non en première forme normale est proposée dans [CLIF 85]. Les opérations *compactage, décompactage, triplet-décomposition, triplet-formation, tranche de temps et suppression de temps* sont introduites. Cependant la sémantique de cette algèbre (comparable à celle de CODD) n'est pas évidente.

3.5- Approche de versions temporelles [LUM 84], [DADA 84]

Dans le paragraphe précédent, on s'est concentré sur la sémantique des attributs de temps et leur incorporation dans un modèle statique. Dans cette approche, tous les n-uplets d'une relation temporelle se situent dans le même plan, ils sont traités de la même façon par le SGBD. Il n'y a pas de distinction entre les *données courantes et les données historiques* (non courantes). Le même mécanisme est utilisé pour répondre à des interrogations sur les données courantes et les données historiques. Ceci peut entraîner des performances non souhaitables du système où les interrogations sur les données courantes sont souvent demandées.

Dans [LUM 84], [DADA 84] pour éviter l'inconvénient mentionné ci-dessus est proposée une approche. Toute relation temporelle est partitionnée en deux relations, l'une correspond à la version courante, l'autre aux données historiques. Le SGBD répond aux requêtes sur les données courantes comme si l'on n'avait pas la notion de versions temporelles. Dans ce cas, le temps de réponse est le même que pour le modèle de données statiques non temporelles. Les index sont aussi divisés en deux, chacun correspond à une relation de données courantes ou historiques. La notion de "delta" de versions est introduite dans [DADA 84] et une stratégie pour cette technique a été choisie. La gestion de versions temporelles n'est pas une couche supérieure à un SGBD mais elle est intégrée directement dans ce dernier.

L'usage des versions temporelles pour le processus de mise à jour, pour le contrôle de concurrence et la reprise de données est également discuté dans ces travaux. Cependant on ne trouve aucun élément sur le langage de manipulation des versions temporelles.

3.6- Notion de photographie [ADIB 80], [ADIB 83]

Les bases de données actuelles s'emploient à fournir aux usagers la version la plus récente et la plus cohérente possible d'une information. Cependant de nombreuses applications tolèrent ou même requièrent une vision des informations telles qu'elles étaient à un certain moment du passé, ce qui correspond à une photographie sur la base de

données. Une photo peut donc être considérée comme une *vue matérialisée* à un instant donné. La notion de photo fut explorée dans [ADIB 80] , [ADIB 83] avec sa sémantique, sa définition et l'opération associée : rafraîchissement.

Supposons que la base contienne les relations DEPT(Dno, Dnom, Dir, Nempl) et EMP(Dno, Enom, Age, Sal). Alors la commande de SQL :

```
DEFINE SNAPSHOT MOYENSAL(Dept, Sal) AS
    SELECT Dno, AVG(Sal)
    FROM EMP
    GROUP BY Dno
```

définit une photo contenant le salaire moyen par départements à l'instant de la définition de la photo. La définition de photos est donc basée sur des interrogations du langage de manipulation.

Une opération associée aux photos est définie : le rafraîchissement. L'usager peut rafraîchir une photo à la demande avec la commande :

```
REFRESH SNAPSHOT <nom-photo>
```

ce qui implique que le système remplace le nouveau contenu de la photo au moment où le rafraîchissement est demandé. Le rafraîchissement peut être fait automatiquement d'une façon périodique. Il est alors défini comme suit :

```
DEFINE SNAPSHOT <nom-photo> <liste-attributs>
  AS   <Requête>
  REFRESHED EVERY <période>
```

et à la fin de chaque période le système rafraîchit la photo à partir de l'état courant de la base.

L'usage de la notion de photo est aussi mentionné dans [ADIB 80], [ADIB 83], notamment dans le contrôle d'accès des usagers à certaines portions d'information de la base, dans la gestion des transactions longues et dans le mécanisme de mise à jour des copies multiples concernant les BD réparties.

3.7- Le temps et la bureautique - modèle TSOS

La gestion du temps est une caractéristique essentielle de la bureautique. Dans ce domaine, en plus de la gestion des attributs de temps de données, la définition et la gestion des associations temporelles entre les activités sont évidentes. Le modèle TSOS (Temporal Semantic Office System)- une extension du modèle SOS [BRAC 83, BARB 85]- comprend un modèle du temps convenable aux exigences ci-dessus. D'une part un calendrier (semaine, minute,.., année) à différentes granularités est introduit et d'autre part, les déclenchements et les contrôles des activités sont caractérisés par des associations temporelles. Exemples :

1/ Règle : conditional part
{ time-point : 1985:11:30-16h}
body print document where id = D10 on printer where id = P02

2/ Considérons les activités A1, A2 et les associations temporelles :

A1.starting-time = from A2.ending-time
A2.ending-time = time-point (1985:11:02)

Dans [MAIO 86] un "time expert" est défini pour la gestion des associations temporelles. Les événements (début et fin d'une activité par exemple) sont les sommets d'un graphe dont les arcs sont les associations temporelles. A partir de cela le graphe peut être partitionné en classes (des sommets). Les connaissances sur ce graphe peuvent être plus précises grâce à la propagation du temps quand il y a une nouvelle information temporelle concernant les événements ou les associations. Le raisonnement sur l'ensemble du graphe permet de décider les déclenchements ou les contrôles sur telles ou telles activités. Il permet aussi de fournir des réponses à différentes questions sur les aspects temporels du système bureautique.

4- Conclusion

Après avoir vu le rôle du temps dans les systèmes d'information et plusieurs concepts de temps dans les différents travaux, on peut maintenant faire une synthèse nécessaire au développement de la notion de temps pour le projet TIGRE.

a) L'intégration du temps dans une base de données a été mise en évidence. Cette intégration est faite sous une forme convenable aux contextes d'application ou de recherche. Le temps est traité sous deux formes : **absolue** et **relative**. Les deux formes de cette dichotomie vont de pair dans les concepts temporels, dans les implantations et applications

b) Il est préférable de choisir une représentation discrète pour le temps, convenant à tous niveaux : conception, implantation et application. Dans la pratique le calendrier grégorien, étendu par l'heure, est choisi.

c) A partir de cette représentation, définir les types de temps comme : **les types de base (les points de temps), intervalle, durée et période** avec des relations : *"avant"*, *"même temps" et "après"*. Construire des opérations possibles sur les types de temps pour la manipulation des données temporelles. Ces opérations doivent avoir une sémantique claire au niveau conceptuel aussi bien que dans les applications.

d) Il existe des attributs **dynamiques** et **statiques** pour les objets temporels. A chaque attribut est associé un type de temps dépendant des contextes d'application.

e) Le temps est souvent introduit dans un modèle statique comme un attribut spécial appelé **attribut de temps**. Deux modèles de temps sont considérés dans les modèles temporels : **temps logique** et **temps physique**.

g) Les **données historiques** sont importantes dans plusieurs applications. Il faut tenir compte de ces données dans la phase de conception et dans la phase d'implantation des systèmes d'information. Il est souhaitable que la gestion des données historiques soit prise en compte par les SGBDs.

Le chapitres suivants développent les points ci-dessus dans le cadre du modèle TIGRE.

CHAPITRE 2

PROJET TIGRE

Les systèmes de bases de données ont été conçus pour gérer des grandes quantités de données alphanumériques formatées, principalement dans le domaine de la gestion. Cependant, de nouvelles applications informatiques ont des besoins plus spécifiques en matière de gestion de données, comme par exemple la bureautique, où l'on manipule des documents, la CAO qui traite des grands volumes d'information à nature et à structure très diverses, le genie logiciel, où l'on gère des programmes, la synthèse de parole, où l'on manipule la voix digitalisée, le traitement d'images, etc.

Pendant longtemps, ces types d'applications ont été developpés par des logiciels "ad-hoc" mais ils nécessitent aujourd'hui le développement de SGBD offrant des fonctionnalités nouvelles. Il s'agit de gérer de manière intégrée des données dites "généralisées", volumineuses et/ou à structure complexe, généralement multi-média (des textes, des images, des graphiques, des voix digitalisées) apparaissant dans les applications ci-dessus.

C'est dans cette ligne que le projet TIGRE (Traitement d'Informations Généraligées REparties) a été lancé conjointement par le Centre de Recherche BULL et le Laboratoire de Genie Informatique de l'IMAG. L'objectif général du projet TIGRE est la manupilation des données généralisées dans un contexte bureautique. A terme, il doit conduire à la réalisation d'un système expérimental s'appuyant sur un serveur de base de données généralisées et plusieurs postes de travail communiquant grâce à un réseau local (voir figure 2.1)

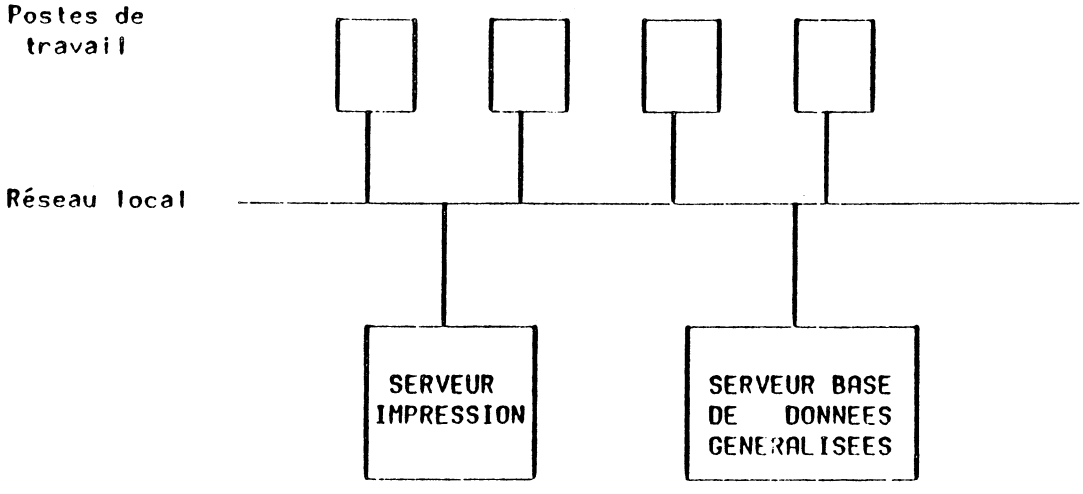

Figure 2.1 Architecture générale du projet TIGRE

Les travaux du projet TIGRE portent sur les sujets suivants :

(1) Conception d'un poste de travail sophistiqué

(2) Modèle et langage de données généralisées

(3) Architecture du SGBD généralisé

(4) Applications et recherches pilotes

Les résultats de ces recherches se trouvent dans l'annexe des rappports de recherche TIGRE. Dans ce chapitre nous allons présenter plus en détail les sujets (2) et (3) pour mieux situer le travail de cette thèse dans l'environnement du projet TIGRE.

1- Modèle de données généralisées TIGRE

Le modèle de données TIGRE a été défini dans le but de fournir le formalisme nécessaire à la modélisation d'applications qui manipulent des données généralisées. Le schéma conceptuel est considéré comme une collection de types de données, définis au moyen de types de base et de constructeurs particuliers.

Les applications bureautiques manipulent des documents structurés logiquement et comportant des informations de natures diverses : des photographies, des formules mathématiques, des textes, des voix digitalisées, etc. Un modèle de document a été défini dans le cadre du projet TIGRE pour en tenir compte et intègre tous ces éléments dans ce qu'on appelle les "documents généraligés".

L'approche adoptée pour la modélisation a consisté à intégrer le modèle de document généralisé à un modèle de données sémantique, dans le but de mieux pouvoir décrire les documents dans l'environnement où ils évoluent. Le modèle de données "hôte" a été le modèle Entité-Association. Plus précisement, on considère les catégories de types de données suivants dans le modèle TIGRE :

1.1- Types de base

Les types de base sont soit simples : entier, réel, booléen, chaines de caractères, soit restreints : scalaire et intervalle (en sens de PASCAL). On les appelle restreints parce qu'ils spécifient implicitement une contrainte d'intégrité par rapport aux types de base simples sur lesquels ils sont définis. Par exemple, une valeur d'un type intervalle est un entier ou un réel qui doit être compris entre une limite inférieure et une limite supérieure données.

Exemple : *type* salaire : (3000..50000) ;

I.2- Types construits

Les types construits sont obtenus par application des constructeurs suivants : "Renommage", "Enregistrement", "Tableau" et "Document". Le constructeur de renommage offre la notion forte de type en permettant de renommer un type de base. Le constructeur enregistrement est similaire au "record" de PASCAL, mais il n'est défini que sur des types de base ou renommés. Le constructeur tableau (Array) produit des types dont les éléments sont des listes d'un même type de base ou renommé.

Le constructeur Document est utilisé pour la définition de ce que l'on entend par "document généralisé". Les types document définis ainsi spécifient une structure hiérachique qui doit être satisfaite par tous les documents de ce type, finis ou partiellement en cours de création. La structure hiérachique des types document est construite à l'aide d'opérateurs de composition qui peuvent être vus comme des sous-constructeurs du constructeur document. Ces sous-constructeurs définissent des éléments de structure (nœuds non-feuilles dans la hiérachie) à partir des objet déjà définis. Ces derniers peuvent être soit des composants élémentaires (appellés unités), soit des éléments de structure, soit des structures de types documents déjà définis. Une unité peut être texte, graphique, image, symbole mathématique ou référence à un élément associé ou à un élément de structure.

Les constructeurs de structure sont :

- Mise en séquence d'objets (agrégat) : "begin .... end"
- Liste d'objets d'une même classe : "list (min, max) of ... ". Min et max fixent la cardinalité de la liste.
- Choix entre plusieurs objets : "case s of v_1:e_1,..., v_n:e_1;" s est appelé le sélecteur, v_1,...,v_n sont les valeurs du sélecteur et e_1,...,e_n les éléments correspondant à chaque cas.

Les éléments associés sont plutôt liés à la présentation de documents. Dans ces éléments associés les paramètres et les fonctions ont une importance particulière car elles permettent l'échange de données entre les documents et d'autres objets de la base. Un paramètre désigne une partie variable d'un document, à laquelle on peut accéder directement par son nom, sans avoir besoin de passer par la structure. Une fonction désigne une procédure algorithmique précise, par exemple des calculs arithmétiques, des références à des objets externes au document.

On peut illustrer le constructeur document par le type Acte (Figure 2.2) du schéma de la gestion des Conférences emprunté à [VELE 85]. Les actes d'une conférence sont vus comme un document de type "Acte" qui est composé de sessions, elle-mêmes composées d'articles qui peuvent être des articles complets ou des résumés. Les types documents rapport et paragraphe, utilisés pour définir la structure des articles, ont été prédéfinis par ailleurs.

```
type CONFERENCE : entity
     nom : string(10) ;
     année : (1960..2000) ;
     ville : string(20) ;
     actes : Acte ;
end ;

type CHERCHEUR : entity
     nom : string(20) ;
     spécialité : string(10) ;
     organisme : string(10) ;
end ;

type Acte : document
  structure
    begin
    | page-pré entation : list (1,*) of unité ;
    | corps : list (1,*) of
    |         session : begin
    |                  | titre : string ;
    |                  | contenu : list(1,20) of
    |                  |    article : begin
    |                  |              | auteur : list(1,*) of string;
    |                  |              | titre : string ;
    |                  |              | contenu : case type-article of
    |                  |              |   art-complet : rapport ;
    |                  |              |   résumé : list(1,5) of pargraphe ;
    |                  |             end ;
    |                 end ;
    end ;

type ORGANISATION : relationship
     between CONFERENCE : conferenceOrganisée (1,*)
         and CHERCHEUR  : comitéd'Oganisation (1,*)
     fonction : string (10);
end ;
```

Figure 2.2 Un schéma TIGRE concernant la gestion des Conférences

I.3- Types classe

Les premiers types classe sont des entités et des associations. Les associations matérialisent un lien sémantique entre des entités. Les entités jouent un rôle vis-à-vis de l'association, qui est spécifié dans la définition de cette dernière. Par exemple les chercheurs jouent un rôle de comité d'organisation vis-à-vis de l'association ORGANISATION. On peut imposer des restrictions sur les nombres minimaux et maximaux de participations d'une occurrence d'une entité à une association, par exemple chaque occurrence de CONFERENCE doit être liée avec au moins une occurrence de CHERCHEUR.

Les types classe peuvent être aussi obtenus à l'aide d'opérateurs de type et sont appelés type classe dérivés. Deux formes d'opérateurs sont considérées dans le modèle TIGRE : la généralisation et l'agrégation. Trois opérateurs de généralisation ont été définis : la spécialisation, l'union et l'intersection. Ils représentent un outil permettant de modéliser le fait qu'une classe d'objet est plus spécifique ou plus générale qu'une autre. Par exemple la classe des ingénieurs est une spécialisation de la classe des employés d'une compagnie. Les opérateurs d'agrégation forment un outil permettant de modéliser le fait que dans le monde réel, certaines entités peuvent faire partie d'autres. Par exemple les lettres et les contrats peuvent faire partie des dossiers. Deux opérateurs d'agrégation sont distingués : agrégation d'entités (plusieurs entités produisent un type classe) et agrégation associative (une association et les entités liées produisent un type classe).

Considérons la définition du type EMPLOYE :

type EMPLOYE : *entity*
nom : string(20) ;
sexe : (masc, fem) ;
date-naiss : jour ;
adresse : t-adresse ;
catégorie : (ingénieur, secrétaire, technicien)
end ;

où les types *jour* et *t-adresse* sont prédéfinis. Alors le type INGENIEUR peut être défini comme une spécialisation de EMPLOYE :

type INGENIEUR : *specialization of* EMPLOYE
where catégorie = 'ingénieur' ;
spécialité : (informatique, mécanique, électrique)
end ;

La définition suivante illustre la notion d'agrégation :

type DOSSIER : entity-aggregate of LETTRE (0,*)
and CONTRAT (1,1)
N#-dossier : integer ;
end;

Dans cet exemple on suppose que les types entité LETTRE et CONTRAT sont prédéfinis. Un dossier a un numéro (N#), zero ou plusieurs lettres, un et seulement un dossier.

Pour une lecture plus complète sur le modèle de données TIGRE voir [LOPE 83], [VELE 84].

2- Langage de manipulation de données généralisées TIGRE : LAMBDA

LAMBDA (LAngage de définition et de Manipulation des Bases de Données génér Alisées) est le langage associé au modèle de données TIGRE. Il est un langage ensembliste de haut niveau qui n'est pas destiné à l'utilisateur final mais plutôt à une intégration dans un langage de programation typé. Les caractéristiques les plus importantes de LAMBDA du point de vue de l'interrogation [VELE 85a, VELE 85b] sont :

1. La capacité de retrouver des documents ou des sous-ensembles des documents à partir de leurs liaisons sémantiques avec les autres informations de la base ou par leur contenu.

2. L'utilisation des trajets et des variables permettant de supprimer les prédicats de jointure présents dans des langages comme SQL ou QUEL. Les trajets reflètent les liaisons sémantiques entre les données (associations, agrégations, généralisations).

3. La possibilité de manipuler de façon homogène des ensembles de données factuelles et des ensembles de données textuelles.

Dans ce paragraphe nous ne nous intéressons qu'à des énoncés d'interrogation de LAMBDA empruntés de [VELE 85a]. Une description complète de ce langage, ainsi que sa complétude et la comparaison avec d'autres langages se trouvent dans [VELE 84]. La forme générale d'un énoncé d'interrogation est la suivante :

SELECT Valeurs
FROM Trajets
[WHERE Condition]

Comme dans le langage SQL, les Valeurs précisent l'information que l'on veut trouver, les Trajets servent à désigner où se trouve l'information demandée et la Condition spécifie des prédicats qui doivent être vérifiés par les données à retrouver.

Exemple 2.1 :

Retrouver le nom et l'organisme où travaille chaque membre du comité d'organisation du VLDB 80.

SELECT t.nom, t.organisme
FROM comitéd'Organisation t *of* CONFERENCE c
WHERE c.nom = 'VLDB' *and* c.année = 1980 ;

Dans la requête de l'exemple 2.1 le trajet "comitéd'Organisation t *of* CONFERENCE c" sert à désigner où se trouve l'information démandée. Il se compose de ce que nous appelons un pas de l'entité CONFERENCE (la racine du trajet) à l'entité CHERCHEUR car le rôle de ce dernier vis-à-vis de l'association ORGANISATION est noté dans le trajet. Ceci signifie que c ∈ CONFERENCE, t ∈ CHERCHEUR et (c,t) ∈ ORGANISATION et ceci correspond à une jointure implicite.

Les trajets en LAMBDA peuvent être linéaires ou arborescents. Un trajet linéaire est une suite de pas. Le départ d'un pas est la cible du pas précédent. Un trajet arborescent se divise en plusieurs branches à partir d'une entité ou association. Pour spécifier cette division, on utilise des parenthèses. Un sous-trajet arborescent qui se divise en plusieurs branches à partir une entité (association) E a la forme :

(<sous-trajet 1>, <sous-trajet 2>) *of* <trajet jusqu'à E>

où <trajet jusqu'à E> est soit E, si E est la racine, soit un trajet linéaire jusqu'à E. Exemple :

Exemple 2.2 :

Retrouver les noms et les fonctions des membres du comité d'organisation des conférences ayant eu lieu à Grenoble.

```
SELECT   s.nom, o.fonction
FROM     (comitéd'Organisation s, conferenceOrganisé c)
         of ORGANISATION o
WHERE    c.ville = 'Grenoble'
```

En LAMBDA on peut formuler des prédicats de jointure explicite. Dans ce cas on spécifie plusieurs trajets dans la clause FROM. Des entités désignées par deux trajets différents peuvent être liées par un prédicat dans la clause WHERE (voir l'exemple 2.6).

Dans la clause SELECT de LAMBDA on peut noter des expressions qui permettent de retrouver une partie d'un document. Ces expressions ont la forme d'un trajet dans la structure hiérarchique d'un document et sont appelées "trajets document". Exemple :

Exemple 2.3 :

Retrouver le contenu de toutes les sessions du VLDB 80.

```
SELECT   contenu of session 1..* of c.actes
FROM     CONFERENCE c
WHERE    c.nom = 'VLDB' and c.année = 1980
```

La notation 1..* indique que l'on veut toutes les sessions. Les trajets document peuvent apparaître dans la clause WHERE et peuvent être arborescents.

Exemple 2.4 :

Retrouver dans les actes du VLDB 80, le titre de chaque session ainsi que le titre de chaque article.

```
SELECT  (titre, titre of article 1..* of contenu)
            of session 1..* of corps of c.actes
FROM      CONFERENCE c
WHERE     c.nom = 'VLDB' and c.année = 1980
```

LAMBDA permet de retrouver des documents par leur contenu, plus précisement par le contenu de leurs unités de nature textuelle. On peut spécifier complètement ou partiellement des chaînes de caractères à l'aide du caractère spécial '*'. Exemple :

Exemple 2.5 :

Retrouver les articles ayant été présentés dans des sessions touchant à la modélisation dans les conférences VLDB.

SELECT contenu *of* session s *of corps of* c.actes
FROM CONFERENCE c
WHERE c.nom = 'VLDB' *and* *model* = titre *of* s

LAMBDA permet de construire des ensembles dans une requête (l'équivalent d'une sous-requête imbriquée dans SQL). Ces ensembles peuvent apparaître comme arguments d'une fonction d'agrégation ou dans des conditions de la forme "<valeur> in <ensemble>" ou "<ensemble> <op-ensembliste> <ensemble>".

Exemple 2.6 :

Retrouver les chercheurs ayant comme spécialité les langages bases de données ou ayant participé aux comités d'organisation d'une conférence ACM SIGMOD.

SELECT s.nom
FROM CHERCHEUR s, comitéd'Organisation t *of* CONFERENCE c
WHERE s.spécialité = 'langages BD' *or*
(s *in* {t} *by* c *and* c.nom = 'ACM SIGMOD')

L'ensemble {t} *by* c est dit "indexé" par la variable c et correspond au comité d'organisation d'une conférence (une valeur de c).

3- Le SGBD TIGRE

Le SGBD TIGRE est construit sur un SGBD relationnel. Pour la première réalisation du prototype, le système relationnel MICROBE [MICR 82] développé au laboratoire IMAG a été choisi. L'interface relationnel entre SGBD TIGRE et relationnel est un ensemble de primitives de l'algèbre relationnelle. La réalisation actuelle s'appuie sur le système ORACLE.

La correspondance de schémas entre les modèles TIGRE et relationnel a été établie [PALA 83, PALA 84], [VELE 86]. La présentation de la structure SGBD TIGRE de ce paragraphe est empruntée de [VELE 85a].

Pour maintenir la structure du schéma TIGRE un ensemble des relations appelé le catalogue TIGRE (la méta-base) est créé lors de la définition du schéma. L'interprétation d'une définition TIGRE insère des n-uplets dans le catalogue et des relations vides correspondantes sont générées.

La définition d'un type document est interpretée de la même façon. Deux relations du catalogue permettent de stocker les règles de définition correspondantes. Les occurrences de documents (de n'importe quel type) d'une base TIGRE sont stockées dans plusieurs relations. Une relation stocke la structure logique des documents, à raison d'un n-uplet par élément de structure. Les unités volumineuses des documents (texte, image), appelés "gros objets", sont découpées en fragments de taille égale à la taille maximale de n-uplet du SGBD sousjacent et stockés comme des n-uplets. Ceci est représenté en relationnel par deux relations. La premième correspond à un répertoire de gros objets et la deuxième stocke les fragments de gros objets. Les unités qui ne sont pas volumineuses (éléments graphiques, symboles mathématiques) peuvent être stockées dans des n-uplets de la relation qui représente la structure logique.

Le traducteur des énoncés de manipulation de LAMBDA est divisé en deux modules principaux, l'analyseur qui accomplit l'analyse lexicale, syntaxique et sémantique et le générateur d'arborescences qui effectue la traduction proprement dite à partir d'une représentation intermédiaire de l'énoncé construit par l'analyseur. Pendant la phase d'analyse sémantique on appelle un module spécial, le gestionnaire du catalogue, qui se charge d'interroger le catalogue TIGRE pour obtenir la description d'un sous-ensemble du schéma TIGRE.

Pour implanter les fonctionnalités offertes par LAMBDA sur les documents un ensemble d'opérateurs de manipulation de document a été introduit. Ces opérateurs peuvent être considérés comme une extension des opérateurs algèbriques du modèle relationnel. Ce sont :

1- Projection à l'intérieur d'un attribut document :

project-doc (R, a, <trajet-doc> [, n])

Cet opérateur permet de retrouver un sous-ensemble de documents. Ici, R est une relation, a est un attribut document, <trajet-doc> est un trajet dans la structure de document qui atteint la (les) racine(s) du (des) sous-arbres à trouver et n est un nœud facultatif.

2-Sélection à l'intéreur d'un attribut document :

select-doc (R, a, <condition> [, n])

Cet opérateur permet d'effectuer la recherche par contenu dans les unités textuelles d'un document.

3- Extraction d'éléments d'un documents :

extract (R, a, <trajet-doc> [,n])

Cet opérateur est utilisé pour extraire des unités textuelles d'un document.

4- Regroupement d'identificateurs de noœds dans un attribut document :

group (R, a, n_1, ..., n_k)

Le résultat de la traduction d'une requête se compose donc d'un ou plusieurs arbres constitués d'opérateurs relationnels et d'opérateurs document. Ces arbres sont interpretés à l'exécution par un module, appelé l'interpreteur TIGRE. Les sous-arbore-cences relationnelles sont envoyées au SGBD relationnel et celles concernant les documents sont interpretées à l'aide du gestionnaire de documents. Les arborescences TIGRE sont mises sous une forme linéaire parenthesée (figure 2.3). L'interface avec le SGBD se charge de traduire ces arborescences en termes de la syntaxe d'entrée du SGBD. L'architecture actuelle du SGBD TIGRE est illustrée par la figure 2.4.

4- Applications et recheches pilote

Plusieurs recherches concernant différents aspects (fonctionnalités du SGBD TIGRE, bureautique) du projet TIGRE ont été étudiés ou sont en cours de spécification :

- Un mécanisme d'autorisation généralisé dans lequel la structure logique des documents et l'organisation hiérarchique d'une entreprise sont prises en compte, a été proposé [ADIB 84].

```
project(
        projec-doc(
                select-doc(
                        project-doc(
                                select(CONFERENCE ,
                                        nom = VLDB )   ,
                                actes,
                                corps--->session 1..*),
                        actes,
                        *model* = titre, session) ,
                actes, contenu, session) ,
        actes)
```

Figure 2.3 L'arborescence (forme perenthesée) de la requête de l'exemple 2.4

- Un mécanisme transactionnel pour le SGBD TIGRE a été proposé [TIGRE 30]. L'accent est mis sur la définition de niveaux de cohérence induits par divers modes de partage des documents, et sur les mécanismes de reprise prenant en compte des interactions longues avec les documents.

- Une coopération entre le serveur base de données TIGRE et un système de programmation logique tel que PROLOG a été étudié [TIGRE 7, TIGRE 12, TIGRE 15]. L'objectif est de définir un outil puissant (SYCSLOG) de vérification de contraintes sur la structure logique de données (alphanumériques et généralisées), sur les propriétés statiques et sur leur comportement dynamique.

- La définition et la manipulation des formulaires multi-média ont été proposées dans [COLL 85]. Un formulaire est vu comme un document spécial auquel on associe une sémantique particulière tant au niveau conceptuel, avec définition d'opérateurs spécifiques sur formulaires, qu'au niveau externe avec interface usager. La coopération du gestionnaire de formulaires et du SGBD TIGRE a également été étudiée.

- Application du prototype TIGRE dans un contexte médical a été abordée [TIGRE 32]. L'accent est mis sur l'utilisation de base de données généralisées pour stocker et sélectionner les images radiologiques à partir des comptes rendus associés. Une cohabitation des données textuelles et factuelles est possible grâce à l'utilisation d'une méthode d'accès aux données textuelles. Les images sont représentées logiquement dans la base de données, tandis que les images physiques sont stockés dans des fichiers externes à la BD.

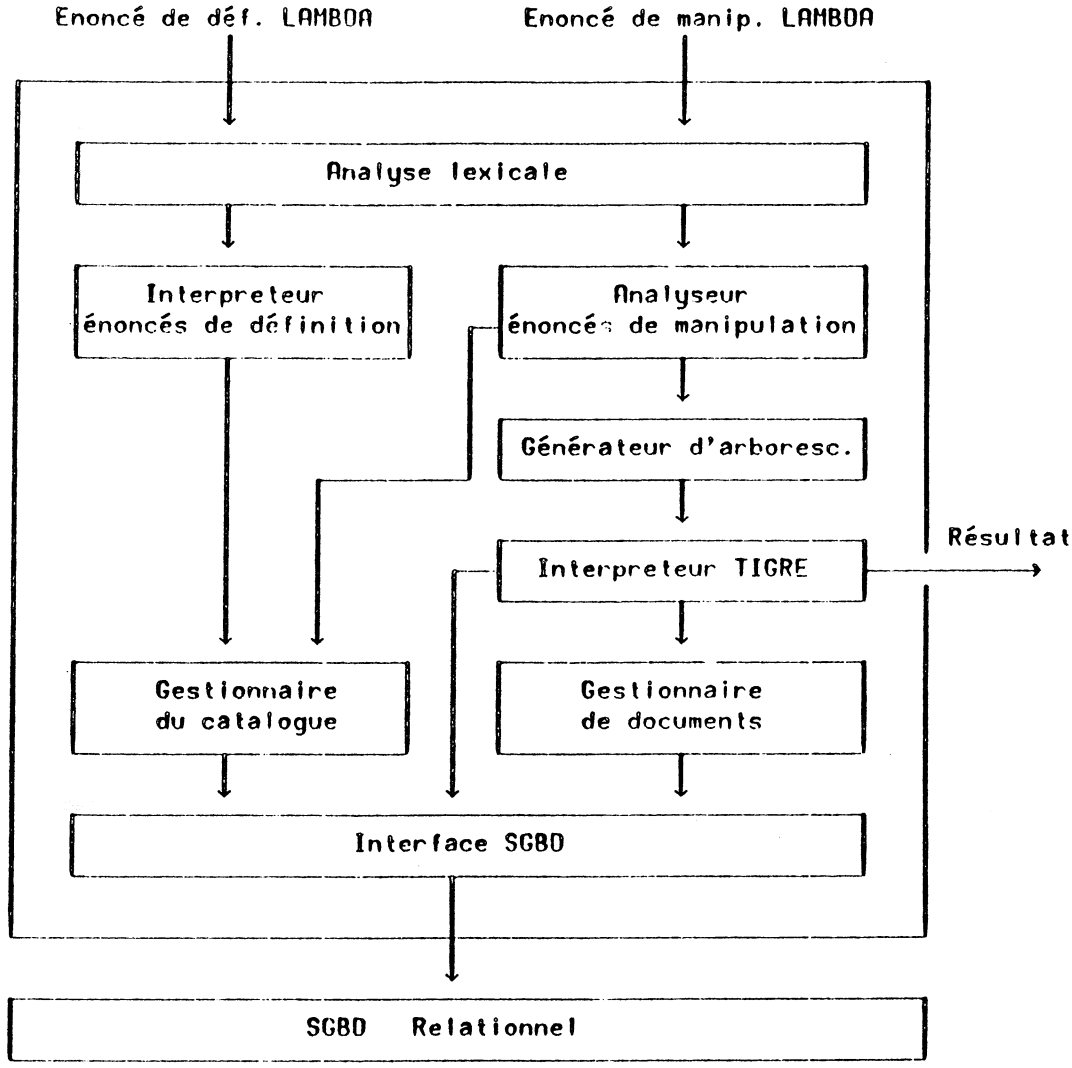

Figure 2.4 L'architecture du SGBD TIGRE

CHAPITRE 3

EXTENSION DU MODELE TIGRE ET DU LANGAGE LAMBDA POUR LA PRISE EN COMPTE DE LA NOTION DE TEMPS

1- Notre approche

Si à l'heure actuelle la bureautique est largement employée, le rôle du temps dans l'environnement bureautique est de plus en plus justifié. Il est nécessaire pour spécifier le temps de création, le temps de vie d'un document ou d'une opération bureautique ou la durée d'une activité. Le temps est utilisé pour déclencher et coordonner des activités, pour contrôler des contraintes temporelles dans des applications telles que l'édition de documents, la manipulation des formulaires ou la messagerie. Les modules suivants sont définis comme outils de base pour la bureautique :

- *Gestion de texte*
- *Gestion de temps*
- *Gestion de communication*

L'objectif majeur du projet TIGRE est de développer un SGBD généralisé, capable de répondre aux applications bureautiques. Le modèle de données TIGRE [LOPE 83] offre déjà la structure nécessaire à la gestion de texte : les documents généralisés. Ces documents sont manipulés à travers le langage LAMBDA [VELE 84, VELE 85b]. La gestion du temps dans ce contexte est développée au cours de ce chapitre.

a/ Extension du modèle TIGRE pour la notion de temps

Nous étendons le modèle de données TIGRE pour inclure la notion de temps (logique) car la sémantique du temps est liée directement à la représentation du monde réel. D'une part, dès lors que le modèle TIGRE est un modèle sémantique, nous offrons toutes les catégories de temps rencontrées dans la bureautique : types de base (points de temps), intervalles, durées et périodes. Ces types de temps peuvent être définis sur différentes granularités de temps, basés sur le calendrier grégorien auquel sont ajoutées les définitions des heures, minutes et secondes. D'autre part le modèle TIGRE est basé sur l'approche typée et nous considérons les types de temps comme types de base ce qui augmente la sémantique du modèle.

b/ Définition des fonctions et des opérations sur les types de temps

Les langages de manipulation de données relationnelles comme SQL ou QUEL permettent de modifier les types de données traditionnelles : entier, réel ou chaînes de caractères. Les données temporelles, comme d'autres données, sont dynamiques , une fois créées elles peuvent être modifiées. Pour répondre à ce besoin et dans le but

de faciliter la manipulation des données temporelles par les usagers, des opérations et des fonctions sur les types de temps ont été définies. Les liens sémantiques entre les types de temps sont exprimés à travers ces fonctions et ces opérations.

c/ Extension du langage LAMBDA pour manipuler les données temporelles

Le langage LAMBDA est ainsi étendu par ces opérations, ces fonctions et par les conditions temporelles dans la clause WHERE. LAMBDA, langage pour un modèle sémantique Entité-Association, est donc capable de manipuler non seulement les documents généralisés [VELE 84, VELE 85b], mais aussi les données temporelles, comme nous le montrerons au cours de ce chapitre.

2-Extension du modèle TIGRE

2.1- Type de base temps simple

Le premier type de temps est le type de base **"time"**. Il est défini par les périodes du calendrier grégorien auxquelles on ajoute l'heure, sous la forme d'heures, minutes, secondes. Une valeur du type "time" est:

1984/05/10 10:15:30

Ainsi on a choisi la présentation externe du type "time". C'est une chaîne de caractères de longueur 20 comme l'exemple ci-dessus.

La définition syntaxique du type "time" est donc donnée par :

<type-temps-simple>::= 'time'
<time>::= <year>'/'<month>'/'<day>' '<hour>'h'<minute>':'<second>|'now'
<year>,<month>,<day>,<hour>,<minute>,<second>::= <entier-sans-signe>

Le mot 'now' fait l'appel à une procédure qui retourne la chaîne de caractères correspondante à l'instant présent.

Dans cette classe de modélisation un système de calendrier est construit par une série ordonnée d'intervalles: année ,mois ,jour ,heure ,minute et seconde.

Les valeurs des intervalles successifs satisfont les contraintes d'intégrité propres au calendrier grégorien, étendu jusqu'à la seconde. Ce sont:

(second >= 0) AND (second <= 60) AND (minute >= 0) AND
(minute < 60) AND (hour >= 0) AND (hour < 24) AND
(day <= 31) AND ((day<> 31) OR (month in (1,3,5,7,8,10,12)))
AND ((day <> 30) OR (month <> 2)) OR ((year MOD 4 <> 0)
OR ((year MOD 100 = 0) AND (year MOD 400 <> 0)))) AND
(month >= 1) AND (month <= 12) AND (year >= 1582)

On peut définir les opérateurs de comparaison "=" et "<" signifiant égal et plus petit sur les valeurs du type "time". Deux valeurs de "time" sont égales si et seulement si les deux chaînes de caractères, qui les représentent, le sont.

L'application de l'opérateur "<" se fait en tenant compte de l'ordre établi dans la définition de type "time" : année, mois, jour, heure, minute,seconde. Selon cet ordre on détermine si valeur1 < valeur2 (deux valeurs du type "time"). D'abord ce sont l'année de valeur1 et l'année de valeur2 qui déterminent laquelle des deux valeurs est "inférieur à l'autre". Dans le cas où les deux valeurs ont le même champ "année", la comparaison se réalise au niveau du mois et ainsi de suite. Pour faire la comparaison de l'année, le mois etc... ,la comparaison "inférieur" est appliquée sur des entiers.

Exemple, le résultat de la comparaison:

1984/05/02 16:20:10 < 1986/06/06 10:15:30

est vrai, tel qu'il l'est dans la compréhension quotidienne.

Par cette définition, deux valeurs du type "time" satisfont l'opérateur "<" si et seulement si les deux chaînes de caractères, qui les représentent, le sont. C'est la raison principale pour laquelle nous avons choisi cette représentation pour le type "time".

Le type "time" est encore appelé **le type temps simple**.

2.2- Types de base temps restreints

Un premier problème pour la représentation du temps est de choisir un niveau de granularité suffisant pour plusieurs applications. Nous avons choisi la seconde comme granularité minimale. Plusieurs applications, cependant, peuvent avoir besoin de granularités différentes. Pour celles-ci on permet la définition de types restreints particuliers avec une granularité plus grande. Ces types sont des restrictions sur l'existence d'intervalles moins significatifs. Ainsi il est possible de redéfinir des calendriers avec une granularité

plus grande.

Par exemple type date : *time* > *hour*

Cette declaration spécifie un calendrier du type :
(année, mois, jour)

qui peut prendre comme valeur 1986/06/12

Les contraintes d'intégrité prédéfinies pour le calendrier de base sont applicables aux nouveaux calendriers. La syntaxe de cette restriction est:

<type-temps-restreint>::='time > '('month'|'day'|'hour'| 'minute'|'second')

Les opérateurs de comparaison ne sont définis que sur des opérandes de même précision. Cette limitation empêche l'apparition d'une valeur indéterminée comme résultat d'une comparaison.

Dans la suite, les valeurs des type temps simple et restreints sont appelées *les points de temps*. Du fait que dans la réalité le temps est isomorphe aux nombres réels, un point de temps ici est en fait un intervalle à durée déterminée.

2.3- Type intervalle de temps

Un autre type qui peut être défini (sur le type temps simple et les types restreints), est l'intervalle. Sa syntaxe est la suivante:

<type-temps-interval>::= 'time-interval'
<time-interval>::='<time>-<time>'|
'<value-time-restr-type>-<value-time-restr-type>'

L'intervalle de temps est défini par deux dates de même précision dont la première est inférieur ou égale à la deuxième. Par définition les deux extrêmes appartiennent à l'intervalle. Un type intervalle peut avoir une des extrémités définie comme "now".

Il est également possible de définir des types renommés sur un type de base temps simple, restreint ou intervalle. Exemple:

type année-scolaire : *time interval*

qui peut prendre comme valeur '1985/10/01-1986/09/30'

2.4- Type durée

Dans certaines applications particulières, on a besoin assez souvent de la notion de durée : la durée d'une bourse de 3-e cycle est de 36 mois, les vacances totales d'un fonctionnaire sont de 5 semaines. Pour la gestion de ce type d'information, on introduit le type durée, dont la syntaxe est la suivante:

<type-durée>::='duration'
<duration>::= <n-year>'y'<n-month>'m'<n-day>'d'
<n-hour>'h'<n-minute>':'<n-second>
<n-year>,....,<n-second>::= <entier-sans-signe>

Nous avons choisi comme présentation externe du type durée des chaînes de caractères de longueur 20, de même que pour le type temps simple.

Exemple: 1036y26m120d15h20:26

Pour des raisons pratiques dans la représentation externe de ce type on peut omettre certaines parties de la syntaxe ou on peut écrire les mots year, month,... complètement. Par exemple les formats:

30d , 12y30m , 26month , 30 day ..... sont également acceptés.

La différence entre cette représentation et celle du type temps simple est que l'on emploie trois caractères pour le nombre de jours. Ceci est justifié par le fait que le calendrier n'a pas de régularité dans son système *(année, mois, jour)* . Le sous-système (*jour, heure, minute, seconde*) est régulier car les égalités :

1 jour = 24 h; 1 h = 60 minutes; 1 minute = 60 secondes

sont toujours vraies, tandis qu'il y a des mois de 30 jours, de 31 jours, même de 28 jours (février). Pour cette raison il est préférable de travailler le plus possible avec les jours, et on a donc choisi de représenter le nombre de jours par 3 caractères.

On peut faire des comparaisons entre les durées, en tenant compte qu'une année fait 365 jours, qu'un mois fait 30 jours. Seulement l'égalité n'est pas transitive , il ne faut pas faire des chaînes d'égalités du type :

1 année = 12 mois = 12×30 jours = 360 jours!!!

Ainsi la comparaison 1 month 20 day < 2 month est vraie.

2.5- Type période

Dans tous les systèmes de gestion il existe des activités périodiques. Ainsi on peut considérer le déjeuner, de 12h à 13h, l'émission d'un rélevé de compte tous les 27 du mois etc. Pour permettre la représentation de ces activités, on va offrir la possibilité de déclarer le type périodique. Pour ce type, le constructeur est périodic et pour valeurs des variables associées on pourra avoir, par exemple "*month:= 12*" pour définir les mois de décembre, "*day:=10-day:=20*" pour définir une période comprise entre le 10 et le 20 de chaque mois. Nous distinguons des périodes instantanées (premier exemple ci-dessus) et des périodes intervalle (deuxième exemple). Les autres possibilités sont données par la grammaire en annexe.

Sémantiquement, une valeur du type période est une suite (finie ou infinie) des points de temps (ou intervalles de temps) qui se repète périodiquement. A chaque valeur de ce type, il faut associer un intervalle de référence où cette période se déroule. Par exemple pour l'énoncé : tous les premiers jours de chaque mois de l'année 1986, l'intervalle '1986/01/01-1986/12/31' joue le rôle de l'intervalle de référence. L'une de deux bornes de l'intervalle de référence peut être "now" ou ∝. Par défaut cet intervalle est (-∝, + ∞). Nous laissons à l'usager le soin de préciser les intervalles de référence dans ses applications.

Le type période peut bien servir à décrire des activités périodiques, en particulier les déclenchements automatiques des transactions. Une transaction est déclenchée quand différentes conditions sont vérifiées, par exemple déclencher la transaction A à minuit le premier jour de chaque mois.

2.5- Constantes du type durée

Pour faciliter la définition du schéma et l'emploi du langage de manipulation il existe la possibilité de définir des constantes du type durée. La syntaxe des constantes peut être donnée comme suit:

time-constant semestre = 6 month
week = 7 day
end;

3- Fonctions et opérations sur les types de temps

Les types associés au temps sont prévus comme des types de base du modèle TIGRE. Afin de faciliter la manipulation de données temporelles on va définir les fonctions et les opérations qui s'appliquent sur les types de temps. Ainsi ces opérations et fonctions lient, d'une façon arithmétique, les types de temps définis dans le paragraphe précédent. Elles rendent plus riche le language de définition et de manipulation des données généralisées -LAMBDA.

3.1- Fonctions de sélection sur les types de temps

Les fonctions suivantes sont définies: **year(t)**, **month(t)**, **day(t)**, **hour(t)**, **minute(t)**, second(t) et week(t) où l'argument t est un type temps simple ou restreint.

Soit VS(T) - ensemble de toutes les valeurs de ces types de temps. Alors la fonction year(t), par exemple peut être vue comme une application de VS(T) dans l'ensemple des entiers positifs N^+ :

$$\text{year} : VS(T) \longrightarrow N^+$$

Exemple year('1983/10/15') = 1983

Tandis que second(t) est vue comme une application partielle parce qu'elle ne s'applique que sur le type temps simple car la représentation des types restreints ne comprend pas de secondes. Le résultat de la fonction week(t) est un jour de la semaine sous la forme: 'Su','Mo','Tu','We','Th','Fr','Sa' .

Exemple t = '1984/05/11 15h30:25'

Alors year(t) = 1984 ; month(t) = 5 ; day(t) = 11 ;
hour(t) = 15 ; minute(t) = 30 ; second(t) = 25 et week(t) = 'Fr'

Le tableau ci-dessous indique les arguments qui peuvent s'appliquer à chaque fonction et le type de son résulttat.

argum fonct	time	time > second	time > minute	time> hour	time> day	time> month	Type du résultat
year(t)	oui	oui	oui	oui	oui	oui	entier
month(t)	oui	oui	oui	oui	oui	non	entier
day(t)	oui	oui	oui	oui	non	non	entier
hour(t)	oui	oui	oui	non	non	non	entier
minute(t)	oui	oui	non	non	non	non	entier
second(t)	oui	non	non	non	non	non	entier
week(t)	oui	oui	oui	oui	non	non	jour de la semaine

Figure 3.1 : Relation entre les fonctions de sélection et les types de temps

Si les types de temps emploient la représentation vectorielle (paragraphe 1.1.2), les fonctions de sélection sont alors des projections sur les composants formant les vecteurs. En effet notre ensemble d'intervalles est I={ *année, mois, jour, heure, minute, seconde* } dont les composants correspondent aux fonctions year(t), month(t), day(t), hour(t), minute(t) et second(t). A remarquer que la fonction week(t) fait la projection de t sur un composant, non présenté dans I, les jours de la semaine.

3.2- Opérations d'addition

Deux opérations d'addition sont définies sur des types de temps, dont la syntaxe est **addtime(t1,t2)**.

3.2.1- Addition sur deux durées

Deux opérandes sont du type durée et le résultat est aussi du type durée.

Exemple: **addtime**('10 month 10day' , '20month 15day 20h') = '30month 25day 20h'

Soit VS(D) - ensemble de toutes les valeurs du type durée. L'addition du premier cas est donc une application:

$$\text{addition 1} : VS(D) \times VS(D) \longrightarrow VS(D)$$

L'opération d'addition sur deux opérandes du type durée est effectuée selon la règle suivante :

Règle 1 : *Les entiers sur chaque composant s'additionnent respectivement. Pour le résultat si les entiers dépassent leurs limites de représentation, la conversion sera faite pour ces entiers, par exemple 100 mois = 8 ans 4 mois. Les égalités :*

1 minute = 60 secondes ; 1 heure = 60 minutes ; 1 jour = 24 heures ; 1 an = 12 mois ;

sont appliquées pour la conversion. Si le nombre de jours est trop grand (> 999) on fait la conversion 365 jours = 1 an. La convention 30 jours = 1 mois n'est pas faite pour éviter la chaînes des égalités :

$$1\ an = 12\ mois = 12 \times 30\ jours = 360\ jours$$

3.2.2- Addition entre un point de temps et une durée

Le premier opérande est du type temps simple ou un des types restreints, le deuxième est du type durée. Le type du résultat est celui du premier opérande.

Exemple: **addtime('1984/02/17' , '3month 10day')** = **'1984/05/27'**

La différence entre les granularités des opérandes est acceptée. Par exemple

addtime('1984/02/05' , '1month 2day 10h 15:00')

En effet cette addition est licite :

addtime('1984/02/05' , '1month 2day 10h 15:00') = **'1984/03/07 10:15:00'**

Selon la règle de l'addition, le type du résultat est le même que celui du premier opérande , le résultat définitif est donc:

addtime('1984/02/05' , '1month 2day 10h 15:00') = **'1984/03/07'**

L'addition, dans le deuxième cas, est donc une application:

addtime 2 : $VS(T) \times VS(D) \longrightarrow VS(T)$

Il faut prévoir des cas d'addition dits anormaux, par exemple:

addtime('1984/03/31' , '1month') = **'1984/04/31'**

Le calendrier grégorien n'accepte pas le format du résultat. Ceci à cause de l'irrégularité du calendrier grégorien: le nombre de jours n'est pas le même selon les mois, il existe des années bissextiles,..etc. Il faut donc modifier le résultat pour qu'il soit

acceptable, par exemple:

addtime('1984/03/31' , '1month') = '1984/04/30'

L'addition **addtime(t1,t2)** est effectuée selon la règle suivante :

Règle 2 : *Sur la partie (heure, minute, seconde) on exécute l'opération normalement car ce sous-système est régulier. Sur la partie (année, mois, jour) il faut garder les conventions du calendrier grégorien. D'abord on exécute l'opération sur le jour puis sur le mois et finalement sur l'année. Par exemple l'addition* **addtime(**'*1984/05/20*' , '*1y2m10d*'**)** *est faite en trois pas :*

a) **addtime('1984/05/20' , '40d') = '1984/06/29'**

b) addtime('1984/06/29' , '2m') = '1984/08/29'

c) addtime('1984/08/29' , '1y') = '1985/08/29'

Ainsi on a finalement :

addtime('1984/05/20' , '1y2m40d') = '1985/08/29'

3.3- Opérations de soustraction

Pour les opérations de soustraction trois cas ont été prévus avec la syntaxe : **subtime(t1,t2)** et la règle :

Règle 3 : t3 = subtime(t1,t2) ⇔ addtime(t2,t3) = t1

3.3.1- Soustraction sur deux durées

Le premier cas s'applique sur deux valeurs du type durée t1 et t2 telles que t1 >= t2. Le résultat est de type durée.

Exemple : **subtime('10month 20day' , '5month 4day 10h') = '5month 5day 14h'**

3.3.2- Soustraction sur deux points de temps

Le deuxième cas s'applique sur deux valeurs du type temps simple ou un des types restreints t1 et t2 telles que t1 >= t2. Il faut que t1 et t2 soient de même type. Le résultat est du type durée.

Exemple : **subtime**('1984/05/03' , '1983/04/01') = '1year 1month 2day'

Pour le deuxième cas l'opération **subtime**(t1,t2) est effectuée selon la règle suivante :

Règle 4 : *La règle est expliquée à l'aide d'un exemple. Supposons que nous voulons effectuer la soustraction* **subtime**(t1,t2) *où :*

t1 = '1984/05/08 10:00:30' et t2 = '1982/06/17 05:30:00'

L'opération est faite d'abord sur la partie (heure, minute, seconde). Ici il n'y a pas de problème.

Pour la partie (année, mois, jour) on cherche la date la plus proche à 1982/06/17 en avançant vers 1984/05/08 qui a le même jour que 08. La date trouvée est 1982/07/08. Le nombre de jours entre 1982/07/08 et 1982/06/17 est 21. La durée entre 1982/07 et 1984/05 est 1year 10 month. Finalement le résultat est 1y 10m 21d 4h30:30.

3.3.3- Soustraction entre un point de temps et une durée

Le troisième cas s'applique sur une valeur du type temps simple ou d'un type restreint et une valeur du type durée. Le type du résultat est celui du premier opérande.

Exemple : **subtime**('1984/05/03 13:00:00','1month 1h') = '1984/04/03 12:00:00'

Pour ce troisième cas la différence entre les granularités des opérandes est acceptée, comme pour le deuxième cas d'addition, présenté au paragraphe précédent. La règle 2 est respectée ici.

La figure 3.2 présente tous les cas prévus pour les opéations d'addition et de soustraction sur les types de temps. Les lignes présentent le premier opérande et les colonnes - le deuxième. Les signes "+" ou "-" indiquent que les opérations d'addition ou de soustraction respectivement peuvent s'appliquer sur les types de temps de ligne et de colonne correspondants.

3.4- Opération metrique time-distance(t1,t2)

Les arguments t1 et t2 sont des points de temps et de même précision. Le résultat de l'opération **time-distance**(t1,t2) est alors du type durée, dont la sémantique est expliquée par le nom de l'opération : durée entre deux point de temps. Cette opération metrique peut être vue comme dérivée de l'opération **subtime**(t1,t2). En effet on a :

	time	time > second	timm > minute	time> hour	time> day	time> month	duration
time	-						+ -
time > second		-					+ -
time > minute			-				+ -
time > hour				-			+ -
time > day					-		+ -
time > month						-	+ -
duration							+ -

Figure 3.2 : Relation entre les types de temps et les opérations arithmétiques

$$\text{time-distance}(t1,t2) = \begin{cases} \text{subtime}(t1,t2) & \text{si } t1 >= t2 \\ \text{subtime}(t2,t1) & \text{si } t1 < t2 \end{cases}$$

3.5- Opérations sur les intervalles

3.5.1- Opération durée pour les intervalles et pour les périodes intervalle : duration(I)

Soit I - un intervalle (ou une période intervalle). Rappelons que I = [t1-t2] et t1,t2 sont de même type (temps simple ou restreint). L'opération durée est alors définie comme la suivante:

duration (I) = subtime(t2,t1)

Le résultat est donc, par définition, du type durée.

L'opération durée peut être vue comme une application:

duration : VS(J) ——→ VS(D)

Où VS(J) est l'ensemble des valeurs du type intervalle et des périodes intervalle.

Exemple : I1 = '1983/02/13 - 1984/05/20' ; I2 = 'day:=10-day:=20'

duration(I1) = '1year 3month 7day' ; **duration**(I2) = '10 day'

3.5.2- Opération d'accès aux bornes des intervalles

On permet d'accéder aux bornes d'un intervalle par les opérations **begin** et **end**.

Soit I = [t1-t2] - un intervalle. Alors par définition on a:

begin(I) = t1 ; **end**(I) = t2

On peut voir **begin**(I) et **end**(I) comme deux applications:

begin , **end** : VS(J) ——→ VS(T)

3.5.3- Opération de multiplication scalaire pour le type durée

L'opération est définie sous la forme:

multime(k,t) où k est un entier et t -une valeur du type durée.

Le résultat est du type durée.

Exemple : **multime**(3,'10month 20 day') = '30month 60day'

3.6- Opération de modification de granularité

L'opération est définie sous la forme: **trunctime**(t,e)

où t est du type temps simple ou restreint et e appartient à l'ensemble {second,minute,hour,day,month}. Si t est du type restreint time > e1 , l'opération sera valide si et seulement si la granularité e est plus grande que e1.

Exemple: **trunctime**('1984/05/16' , day) = '1984/05'

3.7- Opérations pseudo-ensemblistes

3.7.1- Union de deux intervalles : uniontime(I1,I2)

Les opérandes sont deux valeurs du type intervalle dont les bornes sont de même type. Le résultat , s'il existe, est un intervalle.

Soit I1 = [t1-t2] et I2 = [t3-t4] - deux intervalles.

Supposons que t3 >= t1 (sinon leur rôle est inversé).

Si t3 <= t2 alors **uniontime(I1,I2)** = [t1-max(t2,t4)]

Sinon le résultat n'est pas défini.

Exemple: I1 = '1983/10/12-1984/01/15' ; I2 = '1983/10/12-1984/10/26'
uniontime(I1,I2) = '1983/10/12-1984/10/26'

3.7.2- Intersection de deux intervalles : insertime(I1,I2)

La définition de l'intersection est donnée à partir des mêmes conventions que pour l'union.

Soit I1 = [t1-t2] et I2 = [t3-t4] - deux intervalles dont les bornes sont de même type.

Supposons que t3 >= t1

Si t3 <= t2 alors **insertime(I1,I2)** = [t3-min(t2,t4)].

Sinon le résultat n'est pas défini.

On prend l'exemple ci-dessus alors on a :
insertime(I1,I2) = '1983/12/20-1984/01/15'

3.7.3- Inclusion de deux intervalles : incltime(I1,I2)

Soit I1 = [t1-t2] et I2 = [t3-t4].
On veut évaluer l'inclusion I1 $\subseteq$ I2. Si t1 >= t3 et t2 <= t4 le résultat est vrai, sinon faux.

L'inclusion **incltime(I1,I2)** est donc une fonction booléenne. La fonction est valide si les bornes de I1 et I2 sont de même type.

3.7.4- Appartenance intime(t,I)

Soit t1 une valeur de même type que les bornes d'un intervalle I = [t2-t3].

Si t2 <= t1 <= t3 la fonction booléenne **intime(t1,I)** prend la valeur vrai, sinon faux.

3.7.5- Fonction overlap(t1,t2)

La gestion du temps revient souvent à répondre à la question : deux intervalles ont-ils des points communs ? Pour faciliter les manipulations de ce genre la fonction **overlap(t1,t2)** est introduite. Ici les opérandes t1 et t2 sont non seulement du type intervalle mais aussi des types temps simple, restreints ou période.

D'abord considérons deux intervalles I1 et I2. S'il existe des points communs de ces intervalles la fonction **overlap(I1,I2)** prend la valeur vraie, si non elle est faux. Exemple :

I1 = '1982/04-1984/06' ; I2 = '1983/03-1984/08'

Alors **overlap(I1,I2)** est vraie.

Ensuite la fonction **overlap** peut s'appliquer pour un intervalle et un point de temps. Le résultat est vrai si le point appartient à l'intervalle.

Exemple **overlap**('1984/02', I1) est vraie.

Dans la pratique le besoin de la fonction **overlap** concerne aussi les activitées périodiques. Exemple : deux activitées périodiques A1 et A2 sont en exclusion mutuelle pour le partage d'une ressource d'information. Il est nécessaire que les deux intervalles de temps associées à chaque activité A1 et A2, n'aient pas de points communs. De plus l'un de deux opérandes de **overlap** peut être un point de temps et l'autre du type période.

D'une façon générale, notons VS(T,I,P) l'ensemble des valeurs des type temps simple, restreints, intervalle et période. Alors la fonction **overlap** peut être vue comme une application :

overlap : $VS(T,I,P) \times VS(T,I,P) \longrightarrow \{vrai,faux\}$

Le mot pseudo-ensembliste est employé pour distinguer nos opérations des opérations ensemblistes en LAMBDA. En effet les opérandes de nos opérations ne sont pas des ensembles, mais des valeurs (instances) des types de temps.

Pour mieux comprendre le rôle des opérations pseudo-ensemblistes considérons, par exemple deux documents D1 et D2 où D1 est une partie de D2. On peut associer à tout document D un intervalle de temps, noté I(D), qui correspond à la vie de ce document dans le système, c-à-d aux instants où ce document est accessible. Une contrainte d'intégrité liant deux documents D1 et D2 est l'inclusion $I(D2) \subseteq I(D1)$. Ou bien si l'on veut consulter le document D au moment t0, il faut que **intime(t0,I(D))** soit vrai...etc.

4- Le temps et le langage de manipulation

Toute extension à un modèle de données doit en garder les caractéristiques principales. Notre extension du modèle de données TIGRE est conforme à ce critère. En effet elle consiste à ajouter la grammaire des types de temps à la grammaire des énoncés de définition (LDD) et de modification (LMD) du langage LAMBDA. Au travers les exemples nous montrerons les nouvelles possibilités de LAMBDA apportées par l'extension : la définition et la manipulation des données généralisées liées au temps.

Ainsi on va considérer les types de temps comme types de base. Les types de base sont donc entier, réel, booléen, chaîne de caractères et les types de temps. Ces derniers comprenent *types temps simple, types temps restreints, type intervalle, type durée et type période*. La grammaire complète de ces types est donnée en annexe A.

Passons maintenant aux clauses de LAMBDA exprimant la sémantique temporelle.

4.1- Clauses liées à la sémantique temporelle

Soit x,y,z des variables temporelles. Elles peuvent être attributs d'une classe, définis sur un type de temps, une constante de temps ou un paramètre de temps de document. Considérons les clauses suivantes :

a) **x precede y ; x follow y**

b) **x longer y ; x shorter y**

L'introduction de la clause a) a pour but d'exprimer l'ordre "**avant**" et "**après**" entre les points de temps, et pour b) de comparer les durées de temps. Les types de x et y sont donc les mêmes. Pour a) il s'agit du type temps simple ou d'un type restreint et pour b) -du type durée.

c) x precede y *by* z ; x follow y *by* z

x,y sont des points de temps et de même type, z est une durée. Les clauses de c) ont la même sémantique que l'égalité z := **subtime**(x,y) ou z := **subtime**(y,x).

d) x longer y *by* z ; x shorter y *by* z

x,y,z sont de type durée.

e) move-back x *by* y ; move-foward x *by* y

Ici x est un point de temps et y est une durée. Ces instructions sont la même que x := **addtime**(x,y) ou x := **subtime**(x,y).

f) extend x *by* y ; reduce x *by* y

x et y sont du type durée. Ces instructions correspondent à x := **addtime**(x,y) et x := **subtime**(x,y).

g) not precede, not longer,...

Toutes les clauses a)-g) seront utilisées dans les requêtes LAMBDA pour manipuler les données généralisées temporelles. Ceci est illustré dans le paragraphe suivant.

4.2- Schéma illustré : une base de données universitaire

Afin d'illustrer l'usage des types de temps et leurs fonctions et opérations nous prenons une base de données généralisées modélisant la gestion universitaire. Le schéma de cette base est donné par la figure 3.3.

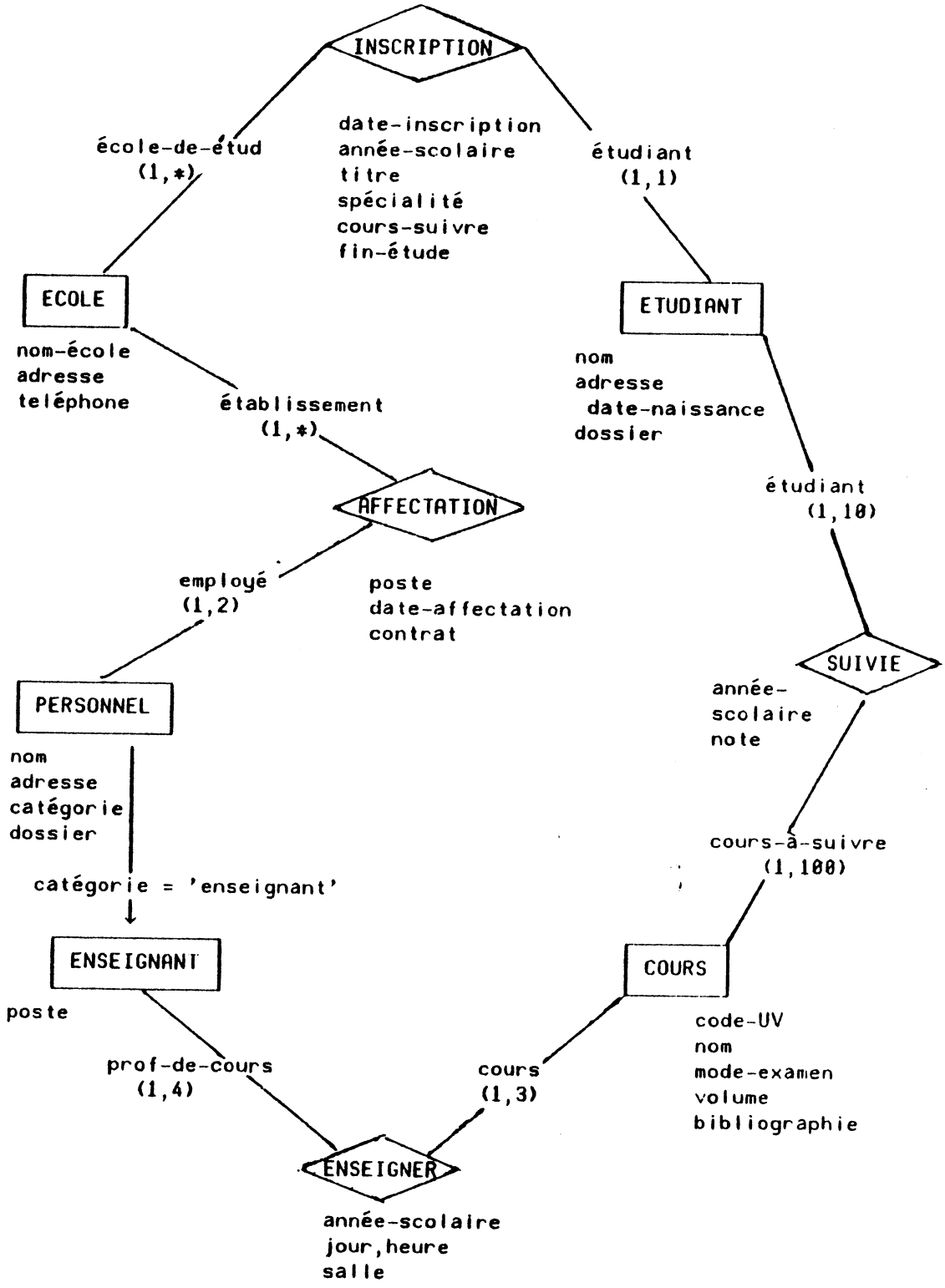

Figure 3.3 : Schéma de la base de données universitaire

On y trouve les entités ECOLE, ETUDIANT, PERSONNEL et COURS, les associations INSCRIPTION, AFFECTATION, SUIVIE et ENSEIGNER et enfin une spécialisation : ENSEIGNANT. Les énoncés de définition de ces types sont données en annexe B. On y trouve des attributs définis sur les types de temps : année-scolaire sur le type intervalle, paramètre durée-bourse sur le type durée, date-naissance et fin-étude sur le type restreint *time* > *hour*, jour et heure (de l'association ENSEIGNER) sur le type période.

Une fois la base définie il doit être possible de l'interroger et manipuler à l'aide des requêtes en LAMBDA. Afin de manipuler les données temporelles, la clause WHERE est étendue pour prendre en compte la sémantique du temps liée aux données TIGRE. Cette extension est illustrée à travers des exemples ci-dessous.

Exemple 3.1 :

Retrouver les noms et adresses des étudiants dont les durées de bourse sont supérieures à 12 mois.

```
SELECT  e.nom, e.adresse
FROM    ETUDIANT e
WHERE   parameter durée-bourse of e.dossier longer '12 month'
```

Ici l'opérateur de comparaision est employé sur deux valeurs du type durée pour sélectionner les données. De même, tous les opérateurs de comparaison sur des valeurs temporelles peuvent être employés dans la clause WHERE.

Exemple 3.2 :

Retrouver les noms des étudiants qui sont nés avant Dupond.

```
SELECT  e1.nom
FROM    ETUDIANT e, ETUDIANT e1
WHERE   e.nom = 'Dupond' and e1.date-naissance precede e.date-naissance
```

Exemple 3.3 :

Retrouver les noms des étudiants qui sont nés 15 jours avant Dupond.

```
SELECT  e1.nom
FROM    ETUDIANT e, ETUDIANT e1
WHERE   e.nom = 'Dupond' and e1.date-naissance precede e.date-naissanc
                                        by '15 day'
```

Exemple 3.4 :

Retrouver les noms des étudiants en DEA de l'INPG qui peuvent bénéficer d'un tarif réduit des billets de train (Rappelons que seuls les étudiants de moins 26 ans peuvent bénéficer de cette offre).

```
SELECT  e.nom
FROM    INSCRIPTION i of étudiant e of ECOLE ec
WHERE   ec.nom-école = 'INPG' and i.titre = 'DEA' and
                        year(e.date-naissance) >= 1961
```

Ici la fonction year(t) est employée pour la requête. Le type du résultat de year(t) est entier, donc il peut être comparé à un autre entier (dans notre exemple 1961). De même les fonctions month(t), day(t),...,second(t) et week(t) peuvent être employées dans les requêtes LAMBDA.

Exemple 3.5 :

Retrouver les étudiants ayant le même anniversaire que Dupond.

```
SELECT  e1.nom
FROM    ETUDIANT e, ETUDIANT e1
WHERE   e.nom = 'Dupond' and
            day(e1.date-naissance) = day(e.date-naissance)
        and month(e.date-naissance) = month(e1.date-naissance)
```

Exemple 3.6 :

Quel jour de la semaine que Dupond est né ?

```
SELECT  week(e.date-naissance)
FROM    ETUDIANT e
WHERE   e.nom = 'Dupond'
```

Exemple 3.7 :

Retrouver le taux moyen de bourse des étudiants faisant une thèse durant l'année scolaire 1985-1986.

```
SELECT  AVG[parameter taux-bourse of e.dossier]
FROM    INSCRIPTION i of étudiant e of ECOLE ec
WHERE   i.titre = 'thèse' and i.année-scolaire = '1985-1986'
```

Exemple 3.8 :

Le professeur Dupont désire connaitre la liste des étudiants de l'année scolaire 1983-1984 ayant suivi son cours de Bases de Données et ayant eu une note supérieure ou égale à 15.

```
SELECT  e.nom
FROM    (prof-de-cours en, SUIVIE s of étudiant et of cours c)
                                    of ENSEIGNER e
WHERE   en.nom = 'Dupont' and s.année-scolaire = '1983-1984'
        and c.nom = 'Bases de Données' and s.note >= 15
```

Exemple 3.9 :

Retrouver les noms et les salaires des employés de la même promotion que Mr Martin (Deux employés sont considérés comme étant de la même promotion si la durée entre leurs dates d'affectation est moins de deux ans).

```
SELECT  p.nom, p.adresse
FROM    PERSONEL p, PERSONEL p1
WHERE   p1.nom = 'Martin'  and
        time-distance(p1.date-affectation,p.date-affectation) shorter '2 year'
```

Exemple 3.10 :

Donner les nom des enseignants de l'INPG ayant commencé à travailler pendant l'année scolaire 1/10/1980-30/9/1981.

```
SELECT  e.nom
FROM    AFFECTATION a of établissement ec of ENSEIGNANT e
WHERE   ec.nom = 'INPG' and a.date-affectation
                overlap '1980/10/01-1981/09/30'
```

Exemple 3.11 :

Retrouver les cours que l'étudiant Dupond suit le lundi dans l'année 1985-1986.

```
SELECT  c.nom
FROM    (suivre-cours e, ENSEIGNER en of prof-de-cours) of COURS c
WHERE   e.nom = 'Dupond' and en.jour = 'Lu' and
          en.année-scolaire = '1985-1986'
```

Dans les exemples ci-dessus nous ne nous intéressons qu'à des interrogations sur les données temporelles. Les exemples suivants nous montreront que LAMBDA permet également de modifier les données temporelles.

Exemple 3.12 :

A partir du deuxième semestre de cette année le cours Bases de Données est enseigné par Mr Dupond avec l'emploi du temps : Lundi, 10h-12h, sale D111.

D'abord il nous faut supprimer la liaison entre l'enseignant du cours Base de Données et ce cours existant avant le deuxième semestre.

```
DELETE  e
FROM    cours c of ENSEIGNER e
WHERE   c.nom = 'Bases de Données'
```

Et ensuite insérer le nouveau enseignant, le nouvel emploi du temps pour le cours Bases de Données.

```
INSERT (e,c) (jour := 'Lu', heure := '10h-12h', sale := 'D111',
              année-scolaire = '1985-1986') TO ENSEIGNER
FROM   ENSEIGNANT e, COURS c
WHERE  e.nom = 'Bases de Données' and e.nom = 'Dupond'
```

Exemple 3.13 :

Pour terminer son projet de DEA l'étudiant Dupont avait demandé au CROUS de prolonger sa bourse de 5 mois. Sa demande a été acceptée. Il faut reporter dans la base de données les modifications dûes à la satisfaction de la demande.

D'abord il faut modifier la durée de bourse de Dupont.

```
Begin-transaction
    X:= SELECT parameter durée-bourse of e.dossier
        FROM   INSCRIPTION i of étudiant e of ECOLE ec
        WHERE  e.nom ='Dupont' AND i.titre = 'DEA'

    extend X by '5 month'

        REPLACE parameter durée-bourse of e.dossier := X
        FROM   INSCRIPTION i of étudiant e of ECOLE ec
        WHERE  e.nom ='Dupont' AND i.titre = 'DEA'

    /* Ensuite il faut faire reculer sa date de fin étude. */

    X := SELECT  i.fin-étude
         FROM    INSCRIPTION i of étudiant e of ECOLE ec
         WHERE   e.nom ='Dupont' and i.titre = 'DEA'

    move-foward X by '5 month'

        REPLACE  i.fin-étude := X
        FROM     INSCRIPTION i of étudiant e of ECOLE ec
        WHERE    e.nom ='Dupont' and i.titre = 'DEA'

End-transaction
```

Dans l'exemple ci-dessus nous avons employé la notion de transaction du langage LAMBDA. L'énoncé exten I X *by* '5 month' est un énoncé LAMBDA. La variable X est une variable temporaire . C'est la façon qui nous semble la meilleure pour exécuter les opérations sur des types de temps. En effet, si nous incluons l'addition dans l'énoncé de modification (on n'aurait donc pas besoin de l'énoncé de selection ni de la variable temporaire X). Ceci briserait l'homogénéité du langage, car LAMBDA ne calcule pas des expressions arithmétiques. Ce serait un manque d'homogénéité que l'on puisse additionner dans une commande REPLACE 5 mois à la valeur d'un attribut défini sur un type de temps, alors que l'on ne peut pas additonner 1000F à la valeur de l'attribut "salaire" d'un type EMPLOYE dans une commande REPLACE.

Dans un programme (transaction) LAMBDA on n'a pas besoin de déclarer X. Par contre dans un couplage de LAMBDA et d'un langage de programmation il faut déclarer que X est une variable LAMBDA.

Dans l'exemple 3.13 les opérations d'addition sont employées sur deux valeurs du type durée (au premier cas) et sur une valeur du type restreint (date : time > hour) et une valeur du type durée (au deuxième cas).

En général les opérations d'addition, de soustraction et d'autres opérations peuvent être employées pour modifier les données liées au temps.

CHAPITRE 4

NOTION D'HISTORIQUE DANS LES BASES DE DONNEES GENERALISEES

1- Motivation et objectifs

A notre connaissance à l'heure actuelle la plupart des SGBD ne sont capables que de fournir des versions les plus récentes et les plus cohérentes des données. Pour de nombreuses applications cette limitation n'est pas acceptable. De plus en plus la gestion des historiques ou l'ensemble des anciennes valeurs de certaines informations est exigée. Par exemple les applications économiques avec l'utilisation de séries chronologiques, la gestion bancaire (anciens relevés de comptes). Même dans les applications traditionnelles de gestion d'une entreprise on peut vouloir poser des questions du genre:

"Le salaire de l'employé X a-t-il augmenté durant ces 5 dernières années?"

La notion de versions est en grande partie associée à celle du temps et nous avons montré dans le chapitre précédent comment le temps pouvait être pris en compte dans un SGBD . D'autre part, plusieurs études ont été proposées dans la littérature pour résoudre les besoins ci-dessus. La plupart d'entre elles sont consacrées à l'introduction de la notion de temps dans la structure de données, voir [BOLO 82], [KLOP 81], [OVER 82]. Dans TERM [KLOP 81] le temps est introduit explicitement au modèle Entité-Association. Il est en fait un type de données particulier défini et manipulé par l'usager et satisfait un certain nombre de prédicats logiques. La solution de cette approche fournit un nouveau type de données : le temps, associé à d'autres données comme un attribut spécial. La gestion des historiques (produit cartésien de l'ensemble de données et l'ensemble de valeurs dans le temps) est faite à l'aide de cet attribut.

Dans cette approche tous les n-uplets d'une relation temporelle se situent dans le même plan, ils sont traités de la même façon par le SGBD. Il n'y a pas de distinction entre les **données courantes** et les **donnés historiques** (non courantes). Le même mécanisme est utilisé pour répondre à des interrogations sur les données courantes et les données historiques. Ceci peut entrainer des performances non souhaitables du système où les données courantes sont souvant accédées. D'autre part les attributs de temps contiennent toujours le temps absolu. Cependant la sémantique du temps est également liée au temps relatif [BOLO 83], par exemple, les interrogations : "Quel est l'avant dernier salaire de Dupond ?", "A-t-il le salaire de Dupond toujours augmenté ?". Pour répondre à ces questions l'attribut de temps n'est pas forcément représenté dans le schéma conceptuel, dans le cas où la base peut conserver les salaires successifs de Dupond.

Dans les SGBD où le temps est géré, le problème peut être vu autrement. Les versions successives d'un objet pourraient être identifiées par le temps du système, associé aux versions dans chaque transaction. Ainsi la dernière version d'un objet est la version

courante et les autres sont des **versions historiques**. La gestion des historiques est donc **necéssaire**, et **peut être** prise en compte pas les SGBD sans ajouter l'attribut de temps.

Dans les nouvelles applictions des SGBD que sont la bureautique ou la CAO les objets à gérer sont plus complexes mais il est également nécessaire de se poser les problèmes de gestion de versions successives pour ces objets complexes. Ainsi dans un processus de conception on transforme un objet en le faisant passer par des versions successives qui reflètent les étapes du travail, à savoir étapes fonctionnelle, logique, électrique et implantée [RIEU 86]. Ici la notion de version est liée à différentes représentations de l'objet à concevoir, à différentes implantations dépendant des choix technologiques. La gestion des versions et alternatives d'un processus CAO a été décrite dans [KATZ 82, DITT 85].

De la même façon on est amené en bureautique à gérer plusieurs versions d'un document [BEN 86]. Dans cet environnement la notion de version n'est pas forcément liée à chaque modification de document. L'édition de document est en général une transaction longue qui laisse le document dans des états éventuellement incohérents. Une version du document peut être obtenue après plusieurs modifications et elle est une **décision humaine**. Dans ce cas on distingue la dernière version (cohérente) de la version en cours d'élaboration qui peut être incohérente.

D'ores et déjà différentes approches ont été proposées pour étendre un SGBD de manière à ce qu'il supporte la notion d'historiques. A notre connaissance les travaux concernant cette solution sont très peu nombreux. Dans [LUM 84], [DADA 84] une extension de la structure d'un SGBD relationnel a été proposée pour stocker les anciennes valeurs de variables de la base sans aborder la définition et la manipulation de ces historiques.

Dans ce chapitre nous proposons une solution pour la gestion des historiques dans un système de gestion de base de données généralisées du point de vue **chronologique**, c-à-d nous considérons l'évolution d'un objet (complexe) par rapport au temps sans préciser d'applications spécifiques. Nos objectifs sont les suivants :

a/ Introduire la structure d'historiques dans le modèle TIGRE

Cette structure est basée sur les notions de *périodicité, de modification, de persistance* et aussi sur les types de versions : *version manuelle, version périodique et version successive*. Ainsi la structure d'un type TIGRE est caractérisée par deux aspects : *structure de données et structure d'historiques*.
La notion d'historique sera appliquée à un modèle sémantique, tel que le modèle TIGRE, ainsi qu'aux objets complexes (les documents).

b/ Assurer l'accès rapide aux versions courantes

En général, bien que le système gère des données historiques on peut s'attendre à ce que la plupart des requêtes concernent des versions courantes de la base. Le temps d'accès aux versions courantes constitue donc une part importante des performances de notre système.

c/ Assurer la sélection des historiques

Dans le langage relationel cela veut dire que l'on doit pouvoir choisir les attributs d'une relation qui devraient être "historisés". Il est clair que le besoin de l'historisation n'est pas le même pour tous les attributs d'une relation, par exemple le nom et la date de naissance d'un employé ne sont jamais changés tandis que son adresse et son salaire changent de temps en temps. Avec la notion de type du modèle TIGRE [LOPE 83] cette sélection des historiques est satisfaite.

d/ Fournir un interface à l'usager pour manipuler les données historiques

Dans le cadre du projet TIGRE le langage LAMBDA [VELE 84] est utilisé pour définir et manipuler les données généralisées. Nous illustrons dans la section 5 des propositions d'extension à LAMBDA permettant de manipuler les données historiques.

e/ Prendre en compte des changements du schéma conceptuel

Certains SGBD (MICROBE [MICR 82] par exemple) permettent de modifier dynamiquement le schéma conceptuel. On peut ajouter (supprimer) un nouvel attribut à une relation. Il faudrait pouvoir prendre en compte ces changements dans la gestion des historiques. Ceci sera présenté dans le chapitre 6.

2- Définition des historiques et des versions d'objets : concepts de base

Considérons une variable (un objet) V de la base de données qui prend au cours du temps des valeurs successives. Notons (v_i,t_i) les couples (valeur,temps) pour cette variable : à l'instant t_i V a pris la valeur v_i. Cette variable est appelée **historique** si un ensemble de tels couples est maintenu dans la base. Ces valeurs v_i sont appelées **versions temporelles** de la variable V. Les t_i sont tous distincts mais ce ne est pas forcément le cas pour les v_i. Prenons comme exemple le salaire de l'employé X, l'intérêt de telles historiques est de pouvoir répondre à des questions du genre :

1) Salaires de l'employé X en Janvier 1985?

2) Salaires de l'employé X au cours de douze derniers mois?

3) Historique complet du salaire de l'employé X?

La question 1) fait explicitement référence à un *instant précis* du passé. La question 2) fait référence à une *série périodique* du temps, mois par mois, du salaire de X en tenant compte des 12 derniers. Quant à la question 3) on s'interesse à tous les couples (salaire,temps) qui correspondent cette fois explicitement aux *versions successives* du salaire.

Notre approche concernant la notion d'historique est basée sur les trois notions suivantes :

1) **La périodicité** des t_i c-à-d la séquence des instants que l'on désire considérer pour caractériser les valeurs v_i. Comme nous le verrons plus loin cette périodicité n'est pas forcément liée à un changement de valeur. Ainsi si l'on s'interesse au salaire de l'employé X au cours des 12 derniers mois, la périodicité est mensuelle mais il se peut que le salaire de X n'ait pas changé pendant plusieurs mois consécutifs.

2) **La modification** d'un objet dont la valeur passe de v à v'. Cette modification peut ou non donner lieu à une nouvelle version de l'objet. Il appartiendra au concepteur de la base de le définir précisement.

3) **La persistance** des valeurs dans la base. En effet on peut considérer que les v_i successives sont "empilées" et on voudrait donc que les n dernières valeurs ou tous les valeurs succesives de V soient maintenues dans la base.

Considérons la relation EMPSAL(E#, NOM, POSTE, SALAIRE) qui donne pour chaque employé identifié par son numéro E# son nom, le poste qu'il occupe et le salaire qu'il gagne. Supposons que l'on veuille conserver pour chaque employé l'évolution de son salaire au cours du temps. Une première solution consiste à ajouter le constituant DATE et de le laisser gérer par l'usager de la base. Ceci peut conduire à des incohérences si le SGBD ne gère pas explicitement le concept de temps et considère DATE comme un simple entier ou une chaine de caractères.

Si le SGBD gère le temps alors pour un employé, tous les n-uplets de EMPSAL sont mis en quelque sorte sur le même plan et il n'y aura pas vraiment de notion d'historique avec dernière version et versions précédentes. C'est précisement le but de notre approche. En effet sans ajouter explicitement le constituant DATE nous allons définir EMPSAL comme une relation historique avec une périodicité, par exemple annuelle et la conservation des 10 valeurs précédentes. Ainsi pour un employé la version la plus récente sera celle qui correspond au poste et au salaire actuels tandis qu'en accèdant aux versions précédentes on pourra avoir l'historique complet.

Si l'on transforme cette approche en termes du modèle TIGRE on définira alors une entité dynamique de la manière suivante :

Exemple 4.1 :

```
type EMPSAL : dynamic entity                    ┐
              each year                         │
              last 10                           │--structure d'historiques
              with time > month                 ┘
               e# : integer ;                   ┐
               nom  : string (15) ;             │
               poste : string(10) ;             │--structure de données
               salaire : (2000..40000) ;        │
             end;                               ┘
```

Dans cet exemple le mot **dynamic** indique qu'il s'agit d'une donnée de type historique. La périodicité (annuelle) est indiquée par la clause **each year** et la clause **last 10** indique qu'il faut garder 10 dernières valeurs pour cette donnée. Le format du temps associé est année indiqué par la clause **with time > month**. La première partie de la définition (avec les mots-clef *dynamic, each, last et with* en italique) correspond à la **structure d'historiques** du type EMPSAL. Le reste est la **structure de données**. Ainsi pour un type TIGRE sa structure est caractérisée par deux structures : structure d'historiques et structure de données. Un type statique TIGRE possède une structure d'historiques vide.

La premième approche pour construire les historiques est de laisser l'usager décider de garder une nouvelle version après une ou plusieurs modifications sur un objet donné, lorsqu'il juge que cela est nécessaire. Dans ce cas on parlera d'**Historiques à Versions Manuelles (HVM)**. Pour illustrer ce type d'historique considérons un document *rapport de recherche*, pour lequel on voudrait garder dans la base quelques versions :

```
type Rapport-RR : dynamic manual document
        begin
          introduction : text ;
          corps : list (1,*) of
              chapitre : list (1,*) of text ;
          conclusion : text ;
        end ;
```

L'édition d'un document de ce type peut être soumise à plusieurs modifications. Cependant à un instant donné, l'usager peut décider que la version courante devrait être gardée, car cette version courante est **cohérente (jolie** version). Dans ce cas l'historique du document contient toutes les versions **successives** que l'usager a décidé de garder.

Si l'on veut un traitement automatique des historiques, on peut utiliser la clause **'each'** pour se référer à une notion de périodicité, par exemple chaque jour ou chaque mois. Dans ce cas on parlera d'**Historiques à Versions Périodiques (HVP)**. Pour un HVP toute modification est effectuée sur la version courante et les nouvelles versions (historiques) sont générées dans la base seulement à la fin de chaque période par copie de la version courante.

Quand il n'y a pas de 'each' ni 'manu el' dans la définition historique il s'agit toujours d'un **Historique à Versions Successives (HVS)** (c-à-d pour chaque modification on retient la valeur). Dans ce cas chaque valeur de V est une version et l'ensemble de telles valeurs constitue l'historique de V. Exemple:

```
type échange : dynamic
                real ;
```

Pour ce type HVS représentant le taux d'échange entre le dollar et le franc toutes les valeurs sont stockées dans la base.

La clause 'last' indique la persistance de V qui se rapporte aux nombres de *versions* historiques que l'on désire stocker dans la base. Une persistance nulle correspond à un type statique où l'on ne conserve que la valeur la plus récente. Pour une variable historique s'il n'y a pas de 'last' la persistance de V est théoriquement illimitée.

La clause 'with' spécifie le type de temps associé aux données historiques ou le format de t_j. Si cette clause est absente le type temps simple 'time' est associé par défaut.

Le choix du type de temps devant être associé à une donnée est une décision laissée au concepteur.

Il faut remarquer ici que le temps associé aux données historiques ci-dessus est le temps interne du système, ou le **temps physique**. Nous avons fait l'hypothèse dans ce chapitre que le décalage entre le temps d'un événement du monde réel et le temps de son enregistrement dans la base n'est pas significatif. Pour une gestion complète des données historiques on a besoin aussi bien du temps physique que du **temps logique** , voir [SNOD 85] et aussi [DADA 84]. Ceci peut être illustré par l'exemple suivant :

Considérons le salaire de l'employé Dupond. Avant le 30 Juin 1984 son salaire était 8000 F. A partir du 1 Juillet 1984 il touche la somme 9000 F. Le nouveau salaire est (ce qui se passe très souvant) retroactif, il a été enregistré dans la base seulement le 15 Juillet 1984. Supposons que l'on s'interesse à la question :

"Quel est le salaire de Dupond le 10 Juillet 1984 ?"

Si l'on prend comme référence le temps physique, la réponse serait 8000 F, ce qui est faux. D'autre part avec un seul temps logique on ne peut pas savoir la retroactivité du salaire de Dupond.

3- Historiques et modèle de données. Propagation d'historicité

Dans le modèle relationnel à structure plate la notion d'historique n'a de sens qu'en considèrant qu'elle s'applique au niveau d'une relation. Cela signifie qu'il faut être capable de (re)construire toutes les instances qu'une relation R a eu au cours du temps. Le passage d'une instance (r_j,t_j) à (r_{j+1},t_{j+1}) se faisant par insertion, suppression, modification de n-uplets dans r_j. La version courante est justement (r_{j+1},t_{j+1}).

Dans le cadre du projet TIGRE un modèle de données a été défini, voir le chapitre 2. C'est une extension du modèle Entité-Association visant à prendre en compte des documents généralisés. Une autre extension du modèle a également été proposée dans le

chapitre 3 et dans le paragraphe précédent pour inclure des concepts liés au temps : les types de temps et les historiques.

Dans un modèle tel que TIGRE où on introduit, grâce aux entités et aux associations, une certaine sémantique, la notion d'historique peut être vue autrement. Chaque occurrence d'une entité une fois crée dans la base possède un indificateur interne (un surrogate) et éventuellement plusieurs constituants qui sont typés. En rendant certains types historiques on peut aussi conserver pour une occurence d'entité, l'historique des valeurs prises par tel ou tel constituant.

Dans TIGRE on peut appliquer la notion d'historique aux types construits (renommé, liste, enregistrement, document) et également aux classes (c-à-d aux Entités ou Associations) mais alors les types composants *ne devront pas être historiques*. On interdit ainsi les historiques d'historiques qui n'ont pas une sémantique claire. Cependant lorsque l'on rend une entité historique, ceci se propage sur ses types composants. Cette propagation est discutée plus en détails plus loin.

Considérons l'exemple suivant :

Exemple 4.2 :

```
type T-employé : dynamic
                 entity
                    nom : string(20);
                    adresse : adresse ;
                    salaire : (2000..30000) ;
                    département : string(10) ;
                  end ;
```

où type adresse a la stucture suivante :

```
type adresse : record
                 numéro : integer;
                 rue : string(15);
                 ville : string(15);
                 code-postal : integer;
               end;
```

Avec cette définition on pourra conserver l'historique des adresses d'un employé donné et aussi pouvoir poser des questions du type :

"Donnez moi toutes les villes où monsieur Dupond a habité "

Mais on aura également l'historique de tous les employés qui auront été crées dans la base en tenant compte des *suppressions éventuelles* de certains employés au cours du temps.

Pour le concepteur de la base, le choix de placer la notion d'historique sur une entité (ou une association) ou sur un type composant est un problème de modélisation qui dépend de l'application envisagée. Notre but ici est de définir et de spécifier les mécanismes nécessaires à la gestion des historiques dans un SGBD généralisé.

La propagation de l'historicité impose les **mêmes caractéristiques** d'un type historique à ses types composants, c-à-d la même périodicité de t_i, la même persistance , les mêmes versions (HVM, HVS ou HVP) ...etc. C'est aussi pour cette raison qu'ici on ne traite pas le problème "historique des historiques".

On dit qu'un type est **défini explicitement** comme un historique s'il est défini dans le schéma conceptuel. Un type est **défini implicitement** comme un historique s'il est devenu historique par propagation de l'historicité.

Nous allons formuler les principes de la propagaption de l'historicité s'appliquant au modèle TIGRE.

a/ Historique et types de base

- Un type de base n'est pas un historique, c-à-d un historique est défini comme un type en LAMBDA.

b/ Historique et types construits

- Les types construits (renommé, liste, enregistrement, document) peuvent être historiques.
- Les types composants d'un enregistrement ou d'une liste historique sont statiques.
- Les types composants d'un document non historique peuvent être historiques.

En effet on n'a pas besoin parfois de stocker des anciennes valeurs pour tout document mais seulement pour certaines de ses parties. De plus il faut faire attention à la notion d'historique appliquée aux documents car ce sont de gros objets.

- Les nœuds d'un document historique défini comme type en LAMBDA sont

historiques (définition implicite).

c/ Historique et types classe

- Les types composants d'une classe historique le sont aussi (définition implicite).
- Une association historique ne lie que des entités statiques. Si une association est historique il faut propager son historicité aux entités liées à cette association. Considérons l'exemple suivant :

type livraison : *relationship dynamic*
each month
between fournisseur *and* article
quantité : integer ;
end ;

Pour cet historique on s'interesse aux quantités d'articles fournis mois par mois par des fournisseurs. A cause des problèmes de cohérence des données il faut que les fournisseurs et les articles auxquels les livraisons font référence se trouvent également mois par mois dans la base. Ceci entraine la propagation de l'historicité de l'association *livraison* aux entités liées : *fournisseur* et *article*.
- Une spécialisation d'un historique l'est aussi (définition implicite).
- Une intersection (union) des historiques l'est aussi (définition implicite) si leurs historicités sont les mêmes. Par contre s'il existe un composant statique de l'intersection (union), le type résultat est statique.
- Les agrégés d'un agrégat (agrégat d'entités ou agrégat associatif) historique sont aussi historiques (définition implicite). A l'inverse il est possible qu'un agrégat statique soit défini sur des agrégés dynamiques.

Dans certain cas la propagation des historiques est une chaine de propagations. Par exemple si un agrégat associatif est historique, il faut propager son historicité à l'association et aux entités qui le produisent. Pour chacune d'elles la propagation continue pour leurs types construits composants.

Considérons deux entités ECOLE et ETUDIANT, empruntées du chapitre 3 :

Exemple 4.3

type ECOLE : *entity*
nom-école : (INPG,U1,U2,U3);
adresse : t-adresse ;
telephone : integer;
enseignement : description;
end;

type ETUDIANT : *entity*
nom : string(20) ;
date-naissance : date ;
adresse : t-adresse ;
bourse : dossier-bourse ;
end;

et l'association historique INSCRIPTION :

type INSCRIPTION : *dynamic*
last 5
relationship between ETUDIANT : étudiant (1,1)
and ECOLE : école (1,*)
date-inscription : date;
année-scolaire : time-interval;
spécialité : string (20);
catégorie : (Thèse, DEA, MIAG, Licence, DEUG) ;
cours-à-suivre : cours-liste;
end;

où : date est un type renommé ('time> hour')
t-adresse est un enregistrement
description et dossier-bourse sont de type document
cours-liste est une liste de string(20)
Ils sont tous statiques et prédéfinis .

L'association INSCRIPTION est un HVS avec persistance 5. Il nous faut propager son historicité à ECOLE et ETUDIANT, ensuite à t-adresse, description, dossier-bourse et cours-liste comme types construits composants. Ainsi les types ECOLE, ETUDIANT, t-adresse, description, dossier-bourse et cours-liste sont devenus historiques (définition implicite).

La propagation d'historicité n'est pas un problème trivial. Considérons trois entités statiques A, B, C et deux associations R1 et R2, R1 liant A et B, R2 liant B et C. Supposons que l'on définisse R1 comme historique, ceci rendra implicitement A et B historiques. Alors si l'on désire que R2 soit historique, un problème se pose, car B est déjà historique. Cependant si R2 et B possèdent la même historicité, la définition d'historique de R2 est possible. Plus généralement, en considérant les historicités compatibles, nous permettons la propagation d'historicité sur les historiques définis implicitement.

4- Historiques et mise à jour

Les mises à jour sur un type historique sont différentes de celles faites sur un type statique.

4.1- Opérations de mise à jour

L'insertion d'une occurrence sur un type historique est toujours faite sur la version courante avec le temps associé pour la création de cet occurence.

La modification sur un HVS est effectuée en deux phases: les n-uplets à modifier sont copiés avec le temps associé dans **une zone d'historiques** (qui est un ensemble de relations gérant les données historiques) et ensuite les nouveaux n-uplets sont insérés dans la version courante. Ainsi la relation de la version courante d'un historique est séparée de celle des anciennes versions. Cela assure l'accès rapide aux versions courantes des historiques.

Si l'usager veut faire une modification sur un HVM, il devrait dire explicitement que cette modification générera une nouvelle version ou non. Dans le langage de manipulation de données cela veut dire que la commande REPLACE modifie la version courante sans garder l'ancienne ou la nouvelle valeur (de modification) dans la zone d'historiques. Si l'usager veut y garder une nouvelle version, avant ou après une modification, il peut utiliser une autre commande : GENERATE-VERSION.

La modification sur un HVP se passe normalement comme si l'on n'avait pas de notion d'historiques. Seulement la version courante de cet historique est copiée d'une façon périodique dans la zone d'historiques. La périodicité de cette action est définie par la clause 'each'.

Pour la suppression il faut considérer les cas suivants:

- Si un n-uplet d'une classe historique est supprimé il est enregistré dans la zone d'historiques avec l'indication de suppression.

- La suppression d'un n-uplet d'une classe historique doit être suivie par la suppression sur ses types composants qui sont définis comme historique d'une façon implicite.

- Si un n-uplet d'une classe non historique contenant des attributs historiques est supprimé on supprime pour chacun de ses attributs sa chaine d'historiques. Exemple : chaque employé possède son adresse définie comme historique. Si un employé est supprimé la chaine de ses adresses l'est aussi.

4.2-Persistance

Chaque fois qu'une unité est insérée dans la zone d'historiques on doit vérifier la persistance de cette unité. Cette notion est importante parce qu'elle permet de contrôler la taille de l'espace de stockage réservé pour l'historique. Par exemple supposons que l'on a une persistance p alors à partir du (p + 1)-ème insertion de telle ou telle unité la version la plus ancienne de cette unité doit être supprimée de la zone d'historiques.

Pour assurer les performances de notre système au niveau de l'implantation on pourrait songer à une exécution différée de la récupération de l'espace de stockage. Par exemple cette récupération pourrait être faite dans la nuit. Si dans la journée il y a des problèmes de stockage elle pourrait être faite immédiatement.

4.3-Correction et modification

Par définition on ne peut pas modifier les données de la zone d'historiques. Cependant on peut s'apercevoir d'erreurs pour ces données, par exemple le nombre de visiteurs d'avant hier introduit dans la base s'avère aujourd'hui incorrect. De ce fait il est indispensable de disposer d'une commande de correction sur les données historiques. Celle-ci est sémantiquement différente de la modification. L'insertion, la modification et la

suppression sont toujours exécutées sur les versions courantes. On verra plus loins des exemples concernant la correction des données.

5- Historiques et langage de manipulation

Pour l'usager qui désire manipuler les données historiques, il faut qu'il puisse y avoir accès par le langage de manipulation. Rappelons d'abord que sur la version courante d'une variable historique de la base TIGRE on peut corriger les données , les interroger et les mettre jour, tandis que sur les versions historiques on ne peut qu'interroger ou corriger (en cas d'erreur).

On va formuler des interrogations sur les données historiques. Le temps associé peut intervenir dans une interrogation de deux façons :

a/ Sous forme de temps explicite (ou absolu)

Exemple :

Q1: " Donnez-moi la valeur de V dans la base à l'instant t_0 "
Q2: " Donnez-moi les valeurs de V dans la base à l'intervalle $[t_1,t_2]$ "
Q3: " Donnez-moi les valeurs de V dans la base après (avant) l'instant t_0 "

b/ Sous forme de temps implicite (ou relatif)

C'est le cas des versions de la variable V. Exemple :

Q4: " Donnez-moi la dernière version de V "
Q5: " Donnez-moi les 3 dernières versions de V "
Q6: " Donnez-moi toutes les versions de V "

De ce fait on va employer le mot-clé **'version'** pour construire des requêtes sur des données historiques. Les requêtes se portent sur les versions $\{v_i\}$ de V maintenues dans la base qui sont manuelles, périodiques ou successives.

Les données TIGRE sont définies et manipulées à l'aide de LAMBDA. Dans le chapitre 3 nous avons proposé des extensions à LAMBDA permettant de manipuler les données temporelles TIGRE. On va montrer dans cette section comment l'usager peut manipuler les données historiques en LAMBDA. Nous utilisons seulement une association AFFECTATION liant deux entités EMPLOYE et DEPARTEMENT pour la gestion des

employés d'une compagnie. Le schéma est défini comme suit:

```
type EMPLOYE : entity
        nom : string(20);
        adresse : adresse;
        catégorie : (ingénieur,secrétaire,technicien);
        contrat : contrat-travail;
end;
```

```
type contrat-travail : document
            structure
               begin contractant :
                     begin employeur : texte = TO
                           employé : référence = titulaire
                     end;
                  corps : list(1,5) of clause
                  signature : image
               end;
            constants unité TO := 'Compagnie ABC'
                  parameter
                   titulaire : texte ;
                   durée-contrat : duration;
                   date-signature : time>hour;
                   établissement : string(20);
                   niveau : string(20);
                  function salaire : sal(niveau,établissement);
      end;
type DEPARTEMENT : entity
         nom : string(10) ;
         chef : string(20) ;
         budget : integer ;
         effectif : integer ;
      end ;
type AFFECTATION : dynamic relationship with time > hour
                  between EMPLOYE : employé (1,1)
                  and   DEPARTEMENT : département (1,*)
            date-affect : time > hour ;
            poste   : string(15) ;
         end ;
```

Le type *adresse* est défini dans l'exemple 4.2 et *clause* est un type document prédéfini. Par définition l'association AFFECTATION est un HVS. Les deux entités liées, EMPLOYE et DEPARTEMENT, deviennent alors HVS (voir les principes de propagation d'historicité du paragraphe 3). Le document *contrat* de chaque employé est donc un HVS, il est renouvelable de temps en temps et ses valeurs successives sont gardées dans la base.

Exemple 4.4 :

La dernière adresse de l'employé Dupont

```
SELECT last version of e.adresse
FROM   EMPLOYE e
WHERE  e.nom = 'Dupont'
```

Pour les données historiques on associe par convention à chaque version un entier ou une fonction last-k, 1 à la première version, 2 à la deuxième version,...., last-1 à l'avant dernière, et last à la dernière version (ou version courante). La requête de l'exemple 4.4 est la même que celle sur la version courante c-à-d elle est équivalente à la suivante :

```
SELECT  e.adresse
FROM    EMPLOYE e
WHERE   e.nom = 'Dupont'
```

Exemple 4.5 :

Les noms et les trois premiers salaires de tous les ingénieurs du département dont le chef est Martin.

```
SELECT e.nom,1..3 version of eval  sal(parameter
          niveau,établissement) of  e.contrat
FROM   AFFECTATION a of employé e of DEPARTEMENT d
WHERE e.catégorie = 'ingénieur' and d.chef = 'Martin'
```

Exemple 4.6 :

Les trois derniers salaires de Dupont

```
SELECT 3 last version of eval  sal(parmeter niveau,établissement) of  e.contrat
FROM   EMPLOYE e
WHERE e.nom = 'Dupont'
```

Exemple 4.7 :

On s'interesse à toutes les premières clauses des contrats de Dupont

```
SELECT  all version of clause l of e.contrat
FROM    EMPLOYE e
WHERE   e.nom = 'Dupont'
```

On peut faire une autre requête équivalente à cette dernière avec '**1..last version**'.

Examinons maintenant des interrogations avec le temps explicite.

Exemple 4.8 :

L'adresse de Dupont au 30/6/1982

```
SELECT  version at '1982/06/30' of e.adresse
FROM    EMPLOYE e
WHERE   e.nom ='Dupont'
```

Ainsi le mot-clé 'version at' est employé pour interroger les données historiques avec le temps explicite. La valeur de temps après ce mot doit avoir une granularité plus grande ou égale à celle du type de temps associé à l'historique interrogé.

Exemple 4.9 :

Les salaires de Dupont pendant la période 1970-1980

```
SELECT version during '1970-1980' of eval  sal(parameter
        niveau,établissement) of  e.contrat
FROM    EMPLOYE e
WHERE e.nom = 'Dupont'
```

Le mot-clé during doit être suivi par une valeur du type 'time-interval'.

Exemple 4.10 :

Les salaires de Dupont après 1980

```
SELECT  version after '1980' of eval sal(parameter
                niveau, établissement) of e.contrat
FROM    EMPSAL e
WHERE   e.nom = 'Dupont'
```

Remarquer que pour un historique à VP quelconque sur V on lui associe une série périodique de temps $T = \{t_1, t_2.., t_n\}$ auxquels les valeurs de V sont retenues dans la base. Supposons que l'on s'intéresse à l'état de V dans la base à l'instant t_0. Si t_0 appartient à T la réponse est claire. Par contre dans le cas où t_0 n'appartient pas à T, une solution simpliste consiste à retrouver t_i de T tel que t_i soit le plus proche de t_0 et le résultat (approximatif) serait la valeur de V à l'instant t_i. Mais il peut exister t_i et t_{i+1} de T "équidistants" de t_0. Dans ce cas on peut prendre comme réponse la valeur de V à l'instant t_{i+1} qui nous semble la plus récente.

Exemple 4.11 :

Liste des employés dont l'ancienneté est supérieure à 10 ans (on suppose que tous les contrats actuels se terminent cette année)

```
SELECT  e.nom
FROM    EMPOLYOE e
WHERE   SUM[all version of parameter durée-contrat of e.contrat]
                        longer '10 year'
```

Comme nous l'avons proposé au début du chapitre, la base de données historiques peut gérer des données éventuellement supprimées. L'exemple suivant le montre :

Exemple 4.12 :

La moyenne par département des salaires des employés licenciés au moment de leur licenciement.

```
SELECT AVG[last version of eval sal(parameter niveau,
        établissment) of e.contrat] by d
FROM    employé e of DEPATEMENT d
WHERE   end of e precede 'now'
```

Les mots-clef **end** *of* suivis par la variable e indiquent que la variable e désigne des faits déjà supprimés de la base. La sémantique de **precede** 'now' signifie que le moment de suppression précède le moment actuel. Si l'on veut les mêmes salaires mais pour les employés supprimés avant 01/01/1984, il faudrait remplacer 'now' par '1984/01/01'.

Exemple 4.13 :

La moyenne par département des salaires d'embauches des employés embauchés après le 01/01/1980.

```
SELECT AVG[first version of eval sal(parameter niveau,
          établissment) of e.contrat] by d
FROM     employé e of DAPARTEMENT d
WHERE    begin of e follow '1980/01/01'
```

Dans cet exemple le mot-clef **begin** *of* désigne le moment où les faits, désignés par la variable e, ont été entrés dans la base.

Exemple 4.14 :

Qui était chef du département employant Dupond au moment où celui-ci avait le salaire de 6000 F ?.

```
SELECT   d.chef
FROM     employé e of DEPARTEMENT d
WHERE    d.begin overlap [e.begin, e.end] and eval
            sal(parameter niveau, établissment) of e.contrat = 6000
```

L'objet désigné par e peut prendre différentes valeurs au cours du temps. Il prend par exemple la valeur v pendant un intervalle (ouvert à la deuxième borne) $[t_1,t_2)$ ce qui signifie que l'objet prend la valeur v à partir du moment t_1 et la garde jusqu'au moment t_2. Les expressions e.**begin** et e.**end** correspondent à t_1 et t_2 respectivement.

Jusqu'ici on a montré la possibilité de LAMBDA pour interroger des données historiques, examinons maintenant le cas de correction.

Exemple 4.15 :

Dans le contrat actuel de Dupont on a detecté une erreur du parametre durée-contrat qui était 2 ans mais sa valeur exacte est 3 ans.

```
CORRECT parameter durée-contrat of e.contrat := '3year'
FROM    EMPLOYE e
WHERE   e.nom = 'Dupont'
```

Dans cet exemple la correction est faite sur la version courante. On ne voit pas le mot-clé 'version' ou 'version at' dans la requête. Par contre le mot-clé CORRECT est employé à la place de REPLACE. Il s'agit donc d'une correction et non pas d'une modification, il n'y a pas d'insertion dans l'historique.

Exemple 4.16 :

Le code postal de l'avant dernière adresse de Martin (38056) est erroné, il doit être 38072

```
CORRECT last-1 version of e.adresse.code-postal := 38072
FROM    EMPLOYE e
WHERE   e.nom = 'Martin'
```

Cette fois la correction est faite sur une des anciennes versions.

6- Unités de gestion des historiques

Sur une base de données on peut faire différentes opérations de manipulation: interrogation, insertion, modification ou suppression de données sans parler du changement de la structure de la base (création ou suppression d'un objet, modification de la structure de tel ou tel objet). Le volume de données à manipuler est par conséquent très varié: des n-uplets, des relations ou des objets volumineux (les documents). On va définir ici ce qu'on appelle unités de gestion des historiques, à savoir les informations à manipuler pour la gestion des historiques sans prendre en compte le contenu ou la sémantique de ces unités.

Ainsi l'unité de gestion d'une journalisation peut être un n-uplet, une page ou une relation selon les implantations. Pour la gestion les versions des objets intervenant dans des applications CAO [KATZ 82] les versions en cours et ses alternatives sont choisies comme

unité de gestion.

En ce qui concerne les données relationnelles un n-uplet peut être considéré comme unité de gestion. Ceci a été proposé et réalisé dans [LUM 84] pour incorporer des mécanismes de gestion des anciennes valeurs dans un SGBD relationnel.

Dans le modèle TIGRE il y a deux groupes de données : les données avec représentation interne relationnelle et les gros objets (les documents généralisées). Pour le premier groupe de données l'unité de gestion choisie est un n-uplet (relationnel). Lors de la modification d'un n-uplet d'un historique à VS l'ancien n-uplet est copié avec le temps associé dans la zone d'historiques. Le nouveau n-uplet est inséré dans la relation de la version courante.

En ce qui concerne un historique à VP, les n-uplets sont périodiquement copiés dans la zone d'historiques (la périodicité de cette opération est définie par la clause 'each'). Les modifications sur cet historique sont faites sur la version courante comme si l'on n'avait pas de notion d'historique. Pour la copie périodique, un n-uplet est inséré dans la zone d'historiques seulement s'il est différent de sa photo la plus récente ou s'il n'a pas encore été enregistré dans la zone d'historiques. Ceci afin de ne pas gaspiller l'espace de stockage.

Ainsi pour les types historiques (non document) leurs relations-P [PALA 83, CODD 79] seront doublées: une pour la version courante et l'autre pour la zone d'historiques. Dans le chapitre 7 nous décrirerons d'une façon plus détaillée ces relations.

Pour les documents l'unité de gestion est une feuille de l'arborescence de document. Dans la première version du prototype TIGRE chaque feuille de document est stockée comme un gros objet (autre possibilité : tout le document est stocké comme un gros objet). Etant donné que la structure arborescente d'un document est stabilisée (c-à-d on ne considère pas ici le problème de changement de structure de document) l'historique d'un document se passe donc sur le contenu de ses feuilles. C'est pour cette raison que nous avons choisi une feuille de document comme unité de gestion. Le mécanisme de gestion de documents historiques par feuille sera détaillé dans le chapitre 7.

Grâce à la notion de surrogate du prototype TIGRE la gestion des historiques est allegée. En effet les historiques d'une unité (que ce soit un n-uplet ou une feuille de document) sont chainés par un même surrogate. Le temps associé à ces historiques les identifie, donc dans chaque relation de la zone d'historiques le couple (surrogate, temps associé) joue le rôle de la clé primaire. Dans le cas d'absence de surrogate des pointeurs les remplaceraient pour chainer les historiques d'une unité.

Examinons maintenant le cas de suppression d'une unité, un n-uplet par exemple. Le n-uplet à supprimer est aussi copié dans la zone d'historiques avec une indication de suppression. Cette indication nous dit que le n-uplet avec tel surrogate a été supprimé et la chaine des historiques est finie par ce dernier n-uplet. Il n'y a pas d'insertion dans la version courante.

Remarque: La gestion des historiques par n-uplet n'est pas une solution optimale car elle entraine de la redondance des données. En effet notre choix est basé sur le fait que les attributs de chaque type construit (enregistrement ou liste) ou classe TIGRE sont représentés par une relation-P dans le prototype. Si chaque objet est représenté par plusieurs relations-P on peut accepter la gestion des historiques par champs (par attributs). Ceci nous garantirait moins de redondance de données car les modifications sont faites en réalité sur les champs (attributs) plutôt que sur les n-uplets entiers. Mais cette approche entraine une augmentation du nombre d'accès aux relations de la base.

Pour les feuilles de document on ne peut pas "descendre" plus bas, notamment vers les pages ou les blocs de stockage. A la différence de données relationnelles les versions d'une même feuille de document ne sont pas les mêmes tant par le contenu que par la taille de stockage.

CHAPITRE 5

PHOTO DYNAMIQUE

Le concept d'historique d'un objet de la base, proposé dans le chapitre 4, est destiné à gérer les anciennes valeurs des types classe d'une base TIGRE. On a pu voir comment les données historiques TIGRE peuvent être stockées et manipulées. Au délà de ces objets de base, le concept d'historique peut avoir d'autres applications. Dans ce chapitre nous allons décrire l'application de ce concept sur les photographies et celle sur le schéma de la base sera abordée dans le chapitre suivant.

L'organisation de ce chapitre est la suivante :

Après avoir rappelé les utilités des vues et des photos dans le paragraphe 1 nous allons définir, dans le paragraphe 2, les photographies comme un type de données du modèle TIGRE avec l'opération associée : **rafraîchissement**. Le paragraphe 3 présente notre approche de l'application du **concept d'historique** aux photos TIGRE ce qui nous permet, par conséquent, de définir la notion **d'album** et de **film**. Dans le paragraphe 4 nous étudierons le problèmes liés à des rafraîchichements de **photos dynamiques** à VS (films). Deux techniques : *détection de transactions ignorées et U-filtre (D-filtre)* seront proposées. Nous concluons le chapitre par le paragraphe 5 abordant le problème **photo de photos**.

1- Utilités des vues et des photographies

Les bases de données maintiennent un vaste volume de données et permettent aux usagers de les manipuler. Ces données sont en général partagées par les usagers. Cependant, chacun d'entre eux peut s'intéresser à telle ou telle portion de la base, d'où la notion de vues ou fenêtres dynamiques sur les données d'une base. Les vues sont supportées par plusieurs SGBD comme INGRES [STON 76], SYSTEM R [CHAM 76], QUERY BY EXAMPLE [IBM 78]. Les vues sont définies à l'aide de requêtes en language d'interrogation et on peut faire des interrogations sur les vues.

Exemple 5.1 :

Considerons l'exemple classique de gestion de l'entreprise où l'on s'interesse aux relations EMPLOYE (No-SS, nom, deptm, catégorie, salaire) et DEPARTM (nom, chef, No-empl). En SEQUEL on peut définir la vue sur les employés du département administratif :

```
DEFINE VEW ADMIN AS
        SELECT No-SS, nom, catégorie, salaire
        FROM   EMPLOYE
        WHERE  deptm = 'Administratif'
```

Les bases de données actuelles s'emploient à fournir aux usagers la version la plus récente et la plus cohérente possible d'une information. Cependant de nombreuses applications tolèrent ou même, requièrent une vision des informations telles qu'elles étaient à un certain instant du passé ce qui correspond à une **photographie** sur la base de données. L'exemple le plus typique de ce genre d'applications est le relèvement des comptes bancaires. Aux jours de diffusion de relèvements on doit connaître les états des comptes à diffuser et puis les stocker pour les imprimer ou les consulter d'une façon indépendante des données de la base. Une photo peut donc être considérée comme une **vue matérialisée à un instant donné**. La notion de photo fut explorée dans [ADIB 80, ADIB 83] avec sa sémantique, sa définition et l'opération associée : rafraîchissement. La plupart des utilités des photos ci-dessous sont empruntées dans ces articles.

Une vue ou une photo permet à l'usager d'interroger ou de manipuler seulement un sous-ensemble d'informations de la base le concernant. Ainsi les vues et les photos sont utilisées pour respecter une **indépendance logique** des applications des usagers vis à vis de la structure du schéma conceptuel, pour cacher (isoler) aux usagers une réorganisation éventuelle du schéma conceptuel. De ce fait, au niveau externe les accès des usagers à la base sont plus naturels.

Les vues et les photos peuvent donc fournir un mécanisme du contrôle d'accès à une base de données par la limite d'accès à certaines portions. La définition de vues des vues (photos des photos) peut aussi donner un intérêt particulier à l'organisation hiérarchique de l'entreprise du système d'autorisation [ADIB 84]. Par exemple les ingénieurs d'un département ne peuvent consulter que les données sur eux-même. Ces dernières peuvent être obtenues par la vue des ingénieurs de la vue du département, voir l'exemple 5.1.

Pour les bases de données généralisées dont les données sont volumineuses et de structure complexe, la notion de photos avec rafraîchissement joue un rôle important. En effet l'existence d'une photo avec rafraîchissement pour un document ou toute donnée de grande taille permet d'accroitre considérablement la disponibilité des données. On peut avoir une transaction très longue sur ces données, l'édition ou la modification d'un document par exemple. Plutôt que de faire attendre les autres usagers qui voudraient seulement consulter ce document on peut leur en fournir une photo, certes en retard sur l'original mais dont ils seront pleinement satisfaits. Il est nécessaire que les usagers soient parfaitement au courant qu'ils manipulent une photo.

Un autre exemple de l'utilité de tels mécanismes concerne les bases de données reparties où bon nombre d'efforts ont été inverstis dans la recherche d'algorithmes (souvent fort coûteux) pour mettre à jour des copies multiples d'un même objet, copies réparties sur différents sites. Le mécanisme de photos permet à coût réduit, une duplication où certains sites possèdent une copie en retard par rapport à l'original, tant qu'ils s'en contentent. Dès que le rafraîchissement sera nécessaire on le mettra en œuvre sur les sites demandant la nouvelle version.

Le mécanisme de vues est supporté par plusieurs SGBD par exemple INGRES, QUERY BY EXAMPLE et SYSTEM R. Ces systèmes ne stockent que les définitions des vues. Ils reconstruisent chaque requête d'interrogation sur une vue par sa définition et obtiennent finalement une requête sur les relations de base de cette vue (technique *query modification* [STON 75]). Comme la vue n'est pas stockée, l'inconvénient majeur de cette approche est de réévaluer la vue à chaque accès. Cependant cette approche donne plusieurs avantages importants. Premièrement, les modifications sur la base sont facilement reflétées sur les vues et vice-versa (sous certaines conditions). Deuxièmement, il est possible d'utiliser l'optimisation d'accès avec la connaissance globale de la définition des vues et la requête d'interrogation. Troisièmement elle est conceptuellement simple.

Dans IS/1-PRTV [TODD 76] dès qu'on accède à une vue (par une requête sur elle), celle-ci est évaluée et le resultat est stocké dans la base. Tout accès à la vue est faite sur cette relation stockée sans besoin de la réévaluer. Lorsque les relations de base sont modifiées la relation stockée pour la vue est détruite et à l'accès suivant elle sera réévaluée. Cette approche ne permet pas de refléter les changements de la base pour les vues entre le moment de leur destruction de vues et celui de leur nouvelle réévaluation.

QUERY BY EXAMPLE permet aux usagers de définir une photo par une requête d'interrogation. La définition de la photo est évaluée la première fois et le resultat est stocké. Toute requête sur la photo est directement exécutée sur ce resultat stocké.

La base de données MADAM : MAp-based DAta Model [KOEN 81] supporte le mécanisme de données derivées. MADAM représente une synthèse de langage de haut niveau, d'association binaire et d'aproche Entité-Association pour la modélisation des données. Un type de données derivées est défini par une expression logique. Cette dernière possède une sémantique plus large qu'une requête d'interrogation en SEQUEL ou QUEL. Toute donnée derivée de MADAM est matérialisée. Une méthode dite méthode de différentiels a été proposée pour éviter la recalculation coûteuse de données derivées. Nous y reviendrons dans le paragraphe 4.

2- Photographie : un type de données TIGRE

2.1- Définition des photos

L'objectif du projet TIGRE est de construire un système capable de manipuler, outre les données classiques, les documents généralisées à structure complexe et qui sont volumineux. Dans cet environnement les utilités des photos décrites dans le paragraphe 1 nous font penser à introduire le mécanisme de photos dans le SGBD TIGRE. En effet les transactions de documents sont en général longues, même quelque heures dans le cas d'édition d'un document. D'autre part les documents sont partagés par plusieurs postes de travail. Le mécanisme de photos pourrait donc améliorer les performances du système pour un bon nombre d'applications, en particulier les applications reparties. Le mécanisme de photos avec rafraîchissement périodique peut apporter encore de nouveaux développements aux données historiques TIGRE. Il s'agit des photos statiques et des photos dynamiques (à VM, à VP ou à VS). Les photos dynamiques seront abordées dans le paragraphe suivant.

Dans TIGRE la sémantique entre les objets à modéliser est exprimée explicitement sous forme d'opérateurs, notamment par deux formes d'abstraction : la généralisation et l'agrégation [LOPE 83]. Il y a trois opérateurs de généralisation : la spécialisation, l'union et l'intersection. Ils sont des outils pour modéliser le fait qu'une classe d'objets est plus spécifique ou plus générale qu'une autre. Les opérateurs d'agrégation sont des outils pour modéliser le fait que dans le monde réel certains objets peuvent faire partie d'autres.

Notre approche est de considérer les **photos comme un type de données, notamment type classe**. On considère un autre opérateur sur les classes TIGRE : opérateur **photographie**. C'est l'un des opérateurs de dérivation. Il exprime le fait que les informations de notre base ne jouent pas le même rôle, certaines peuvent être dérivées à partir d'autres, les unes ont plus d'intérêt pour l'usager que les autres. A chaque dérivation de données on

associe souvent : *les classes de données à dériver (D), les règles de dérivation (R) et les conditions de dérivation (C).* Les données dérivées sont alors obtenues par l'application des règles R sur les données D lorsque les conditions C ont lieu.

Les données déductives peuvent servir d'exemple aux données dérivées. Les règles de dérivation sont des prédicats logiques du premier ordre, par exemple le problème Père-Fils-Ancêtre bien connu dans la littérature :

1/ Si x est père de y alors x est ancêtre de y

2/ Si x est ancêtre de y et y est ancêtre de z alors x est ancêtre de z

Pour cet exemple les couples (x,y) où x est père de y sont suffisants pour déduire (dériver) les ancêtres de toutes les personnes enregistrées dans la base.

Les vues et les photos se groupent dans une autre classe de dérivation. Les règles de dérivation sont exprimées sous forme de requêtes d'interrogation en langage associé aux données, voir exemple 5.1. Cette classe de dérivation a l'avantage majeur d'utiliser les primitives du SGBD pour maintenir et manipuler les données dérivées. En effet pour les données relationnelles le résultat d'une requête relationnelle est aussi une relation (propriété de couverture de requêtes relationnelles). D'autres règles de dérivation pour les photos sont des résultats de certains programmes d'application de l'usager. Pour cette classe de dérivation on ne voit pas clairement le lien sémantique de ces données dérivées avec le reste des données de la base et elles perdent évidemment l'avantage évoqué ci-dessus.

Une photo TIGRE est définie à l'aide d'une requête d'interrogation en LAMBDA. Considérons l'exemple suivant basé sur le schéma de données universitaire du chapitre 3.

Exemple 5.2 :

type INFOR-ETUD : **snapshot** (nom-étud, école, année-scolaire, spécialité,
titre, cours-suivre)
AS SELECT e.nom, ec.nom-école, i.anné-scolaire, i.spécialité,
i.titre, i.cours-suivre
FROM INSCRIPTION i *of* étudiant e *of* ECOLE ec

Cette photo contient les informations concernant les études de tous les étudiants. Elle est stockée dans la base et on peut seulement la consulter à n'importe quel moment pour obtenir les informations concernant les études des étudiants sans besoin d'accéder aux trois classes : ECOLE, ETUDIANT, INSCRIPTION. Etant donné que durant une année scolaire ces informations ne sont pas modifiées on n'a pas besoin de rafraîchir cette photo

dans une période inférieure à une année scolaire. Là il y a encore un intérêt à ce type de données, photographie : il nous permet de consulter rapidement une portion d'informations de la base.

On peut donner un autre exemple de photo sur les documents de type *rapport de recherche* (voir l'annexe B).

Exemple 5.3 :

type SOMMAIRE-BD : **snapshot** (nom-auteur, lieu-travail, titre, résumé, conclusion)
AS SELECT (nom, lieu-travail, titre) *of* page-couverture *of* r.contenu,
résumé *of* r.contenu, conclusion *of* r.contenu
FROM RAPPORT-RECHERCHE r
WHERE matière *of* page-couverture *of* r.contenu = ' Base de Données'

Ainsi cette photo contient tous les élements les plus importants pour une consultation rapide des rapports de recherche concernant les bases de données.

D'une façon générale on dénote par REQ (T1,T2,..,Tn) une requête d'interrogation en LAMBDA où Ti 1 <= i <= n sont de type classe. Dans un premier temps on ne considère que des Ti non dérivés c-à-d que parmi les Ti il n'y a pas de photo. Le problème "photo de photos" sera discuté dans le paragraphe 5. L'énoncé de définition d'une photo peut alors être formellement décrit comme suit:

type <nom-de-photo> : *snapshot* (<liste-des-attibuts>)
AS REQ (T1,T2,..,Tn)

Sémantiquement un type photo est un type classe TIGRE et les attributs d'une photo sont définis par la requête associée. Le type de chaque attribut hérite donc du type correspondant dans la clause SELECT de la requête définissant la photo. Par exemple pour la photo SOMMAIRE-BD de l'exemple 5.3 les attributs nom-auteur, lieu-travail, titre sont des chaines de caractères, résumé et conclusion sont des documents. Une fois la photo définie, le SGBD TIGRE évalue la requête (si la requête est correcte), génère l'espace de stockage pour la photo et y stocke le résultat de l'évaluation. A partir de ce moment on peut interroger la photo.

Exemple 5.4 :

Donnez moi tous les titres et résumés des rapports de recherche en BD ayant effectués à l'IMAG :

```
SELECT  s.titre, s.résumé
FROM    SOMMAIRE-BD s
WHERE   s.lieu-travail = 'Institut IMAG'
```

Ainsi on a enrichi les aspects sémantiques du modèle de données TIGRE en ajoutant un nouvel opérateur de dérivation : **la photographie**, dont le résultat est considéré comme un type classe. Nous allons voir les différences entre les trois formes d'opérateurs du modèle TIGRE : la généralisation, l'agrégation et la photographie. Tout d'abord nous pouvons considérer que la généralisation est un cas particulier de la photo. En effet, dans le cas de la photo, on ne permet pas d'ajouter d'autres attributs suplémentaires (rappelons que ceci a déjà été suggéré dans [ADIB 80]). La généralisation et l'agrégation sont liées plus particulièrement à la représentation du monde réel tandis que la photographie est plutôt liée aux programmes d'application de l'usager sur les données déjà existantes dans la base. D'autre part les opérations autorisées sur les deux premières sont la consultation et la mise à jour mais pour la photographie seulement la consultation et le rafraîchissement. En effet les photos dérivées d'autres données sont moins indépendantes que les données du type généralisation ou agrégation.

2.2- Opération associée à une photo : le rafraîchissement

On peut créer, détruire, interroger une photo mais on peut également la rafraîchir. Cette opération permet de modifier le contenu de photo. Pour cela on dispose en LAMBDA de deux commandes :

1/ REFRESH <nom-de-photo>

2/ REFRESH <nom-de-photo> AT <valeur-de-temps>

Dans le premier cas il s'agit d'un rafraîchissement immédiat, le SGBD TIGRE évalue la requête de définition de la photo et remplace le contenu de la photo par le résultat de cet évaluation. Pour la deuxième commande nous explicitons le terme <valeur-de-temps> précédée du mot-clé AT :

- Cette valeur appartient *au futur*. Il s'agit d'un rafraîchissement différé. Au moment précisé le SGBD évaluera la requête de définition de la photo et remplacera son contenu par le résultat de l'évaluation. Ceci nous donne une possibilité de faire en différé un rafraîchissement dont l'exécution est coûteuse, dans la nuit par exemple.

- Cette valeur appartient *au passé*. On veut voir l'état de la photo à un instant du passé. Le SGBD évalue immédiatement la requête de définition si tous les T1,T2,..,Tn de cette requête sont historiques et si les faits de ces types Ti correspondant à ce moment-là sont retenus dans la base. Autrement le SGBD retourne un message d'erreur à l'usager.

Ces commandes sont illustrées par les exemples suivants :

Exemple 5.5 :

Rafraîchissement immédiat de la photo SOMMAIRE-BD

REFRESH SOMMAIRE-BD

Exemple 5.6 :

Etant donné que toutes les inscriptions de l'année scolaire 85-86 sont terminées à 16 h le 29/Nov/1985 rafraîchir INFOR-ETUD à 2h dans la nuit suivante :

REFRESH INFOR-ETUD AT '1985/11/30 02h'

2.3- Support de photos dans TIGRE

Nous allons voir maintenant comment le SGBD TIGRE supporte la notion de photo. Les photos sont physiquement stockées, et elles sont réévaluées lorsque un rafraîchissement est demandé. Pour ce genre de données la technique de compilation est mieux adaptée [ADIB 80]. Dans TIGRE une requête LAMBDA est transformée par le translateur et le générateur d'arborescence TIGRE en un arbre algèbrique (étendu par les opérateurs sur les documents [VELE 84]) sous forme de liste parenthesée [VELE 85a]. Cette liste parenthesée est ensuite soumis à l'interpréteur TIGRE. La solution immédiate pour le support de photos dans TIGRE est de stocker la liste correspondant à la requête de définition d'une photo comme gros objet. Dès que l'on veut rafraîchir la photo, la liste parenthesée correspondant à la photo est appelée dans la mémoire centrale et soumise à l'interpréteur TIGRE. Après avoir terminé l'interprétation de cet arbre algèbrique le rafraîchissement se

termine par le remplacement du contenu de la photo par le résultat obtenu.

3- Photos dynamiques. Notion d'album et de film

On a exploré dans le paragraphe précédent la définition des photos TIGRE et l'opération associée : rafraîchissement. Une photo possède sa structure de données caractérisée par la requête de définition. Jusqu'ici nous avons seulement considéré le côté statique des photos en ignorant leur côté dynamique. En fait la structure d'historiques developpée dans le chapitre 3 peut s'appliquer à n'importe quel type de données du modèle TIGRE y compris le type photo. La notion de photo dynamique est abordée dans ce qui suit.

L'opération de rafraîchissement est la seule qui permet de modifier le contenu d'une photo. Ce rafaîchissement reflète l'état d'une portion de la base à un instant donné. Jusqu'au prochain rafraîchissement le contenu de la photo ne change pas en dépit des changements éventuels de la base. De nombreuses applications requièrent un rafraîchissement périodique de la photo, par exemple chaque jour ou chaque mois. L'exemple typique de ce genre d'applications est de figer l'état de chaque compte au premier jour de chaque mois pour les relevés de compte. Dans certaines applications on s'intéresse aux rafraîchissements immédiats des photos, par exemple si l'on veut savoir exactement les salaires des chefs d'une entreprise à n'importe quel moment on peut définir la photo "Salaires de tous les chefs departement" (voir l'exemple 5.1) en demandant un rafraîchissement immédiat après chaque modification de la base concernant ces salaires. Il y a même des applications qui requièrent de stocker explicitement un nombre de versions d'une photo. Exemple : stocker chaque jour le chiffre d'affaire d'un super marché. Ce dernier est en fait une photo des sommes des ventes de tous les articles du super marché. Une valeur de cette somme correspond à une version de la photo.

Les exemples ci-dessus ont mis en évidence le fait que la structure d'historiques peut parfaitement s'appliquer aux photographies. Il s'agit des photos dynamiques qui sont caractérisées , comme d'autres historiques, par la *périodicité* du temps de rafraîchissement, par le *type de version (VM,VP ou VS)*, par la *persistance* dans la base et par le *format* du temps associé.

Exemple 5.7 :

Définir la photo sur le nom, l'année scolaire et les cours à suivre de tous les étudiants en DEA informatique de l'INPG avec rafraîchissement immédiat, après chaque modification de la base (voir le schéma base de données universitaires du chapitre 3) :

type DEA-Informat : **dynamic snapshot** (nom, année-scolaire, cours-suivre)
AS SELECT e.nom, i.année-scolaire, i.cours-suivre
FROM INSCRIPTION i *of* étudiant e *of* ECOLE ec
WHERE i.spécialité = 'Informatique' AND i.titre = 'DEA'
AND ec.nom-école = 'INPG'

Ce type photo est un historique à VS avec persistance illimitée. Chaque fois que l'on ajoute (supprime) un étudiant ou que l'on modifie la liste de ses cours à suivre d'un étudiant on doit les reporter immédiatement à DEA-Informat ou bien on le rafraîchit après chaque modification de la base concernant les étudiants en DEA informatique.

Exemple 5.8 :

Définir la photo sur les 10 derniers salaires moyens de chaque année pour tous les départements (voir l'exemple 5.1) :

type SALAIRE-MOYEN : **dynamic snapshot** (nom-départm,salaire-moyen)
each year
last 10
with time> month
AS SELECT d.nom , AVG [e.salaire] *by* d
FROM DEPARTM d, EMPLOYE e
WHERE e.déptm = d.nom

SALAIRE-MOYEN est un historique à VP avec périodicité annuelle de rafraîchissement et persistance 10. Chaque année on le rafraîchit avec l'année courante associée en tenant compte les 10 derniers salaires moyens pour chaque département.

D'une façon générale la syntaxe des énoncés de définition de photos dynamiques peut être vue comme la suivante :

type <nom-photo> : **dynamic snapshot** (<liste-des attributs>)
<structure-d'historiques>
AS REQ (T1,T2,..,Tn)

La syntaxe de la structure historique a déjà été définie dans le chapitre 4. Les types T1,T2,..,Tn peuvent être statiques ou historiques. Autrement dit une photo historique peut être définie sur d'autres historiques. Ceci est tout à fait différent des données historiques de base - les types historiques non photo (paragraphe 3 du chapitre 4). De plus il n'y a pas de propagation d'historicité pour les types source d'une photo dynamique. Là le mécanisme de photos dynamiques peut encore apporter une solution en assurant la selection des historiques (paragraphe 1 du chapitre 4). En effet supposons que les Ti soient statiques et que l'on veuille *historiser* une portion de données sur ces Ti. Au lieu d'historiser chaque attribut de cette portion au sein de Ti (comme on l'a fait dans le chapitre 3) on peut regrouper cette portion dans une photo dynamique (exemple 5.7).

Pour une photo dynamique à Version Manuelle (VM), une nouvelle version est générée par la commande GENERATE-VERSION (paragaphe 4 du chapitre 4). Le résultat de cette commande est de copier le contenu courant de la photo dans la zone d'historiques. La commande REFRESH peut modifier le contenu de la photo mais, comme toute modification sur les HVMs, elle peut ne pas générer une nouvelle version (voir la définition des historiques dans le chapitre 4).

Pour une photo dynamique à VP la périodicité de rafraîchissement est indiquée par la clause 'each' qui peut être mensuelle, annuelle ou chaque lundi de la semaine (each 'Mo'). S'il n'y a pas de 'each' dans l'énoncé de définition il s'agit d'une photo dynamique à VS pour laquelle le rafaîchissement doit être effectué après chaque modification sur T1,T2,..,Tn.

Comme pour d'autres historiques, la clause 'last' indique la persistance de la photo dynamique. Pour les données non photo une persistance nulle correspond à un type statique et toute modification est effectuée sur la version courante sans garder des versions dans la zone d'historiques. Cependant la sémantique de la photo dynamique peut tenir compte de la persistance nulle pour les photos dynamiques à VS ou à VP. Par exemple on peut demander au système de rafraichir une photo chaque mois (le rafraichissement est effectué sur la version courant du contenu de la photo), dans ce cas la photo est dynamique mais a une persistance nulle et on emploie le mot-clé 'last only' pour définir une photo dynamique à persistance nulle. Exemple :

Exemple 5.9 :

Définir la photo sur les salaires moyens des employés par département avec un rafraichissement annuel.

type SALAIRE-DEPTM **dynamic snapshot** (nom-deptm, sal-moyen)
each year
last only
AS SELECT d.nom, AVG [c.salaire] *by* d
FROM DEPARTM d, EMPLOYE e
WHERE e.déptm = d.nom

D'une façon imagée une photo dynamique à VP est une collection des photos figées à différents instants. Elle nous donne **un album automatique** *sur la base de données. De la même façon une photo dynamique à VM est un* **album manuel**. *Quant à une photo dynamique à VS elle reflète tous les changements de la base sur les types source T1,T2,..,Tn ce qui correspond à* **un film** *sur notre base.*

La figure 5.1 nous permet de regarder l'évolution des notions : *vue, photo, album et film.*

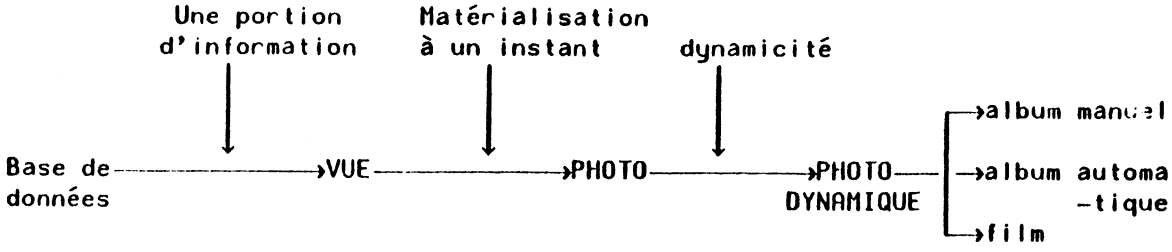

Figure 5.1 L'évolution des notions : vue, photo, album et film

Le mécanisme de gestion des historiques proposé dans le chapitre 4 reste le même pour les photos dynamiques. Plus exactement l'espace de stockage de la version courante est séparé de celui des anciennes versions, c-à-d il y a aussi une zone d'historiques pour les photos dynamiques. A chaque rafraîchissement d'une photo dynamique, au lieu de détruire l'ancienne version de la photo, on renvoie sa copie dans la zone d'historiques. Le

temps associé est joint à la version courante pour respecter la sémantique de la photo.

La consultation des photos dynamiques nous permet de regarder certaines portions de la base à des moments du passé.

Exemple 5.10 :

Le salaire moyen du département administratif en 1980 (voir l'exemple 5.8)

```
SELECT  version at '1980' of s.salaire-moyen
FROM    SALAIRE-MOYEN s
WHERE   s.nom-départm = 'Administratif'
```

Exemple 5.11 :

La requête suivante nous permet de savoir si l'étudiant Dupond a redoublé son DEA informatique de l'INPG :

```
SELECT  all version of d
FROM    DEA-Informat d
WHERE   d.nom = 'Dupond'
```

Remarque : Considérons la requête : donnez-moi l'état de la photo dynamique P à l'instant t_0. D'une part comme pour d'autres historiques la granularité de t_0 ne doit pas être plus petite que celle du temps associé à P. D'autre part si P est un historique à VP, t_0 doit alors être un instant auquel P a réellement été figée. Ceci est un peu différent de la remarque (paragraphe 5 chapitre 4), faite sur les données historiques non photo, pour lesquelles on peut se contenter d'un résultat approximatif.

4- Rafraîchissement des photos dynamiques à VS

L'implantation du mécanisme de rafraîchissements de photos dynamiques à VS (un film) pose des problèmes car un rafraîchissement est en général une opération coûteuse. Il s'agit, comme pour *les metteurs en scène*, de trouver comment un tournage peut être le mieux réalisé. La méthode brutale consiste à détruire le contenu de la photo à rafraîchir et de la récréer par réévaluation complète de la requête de définition. Prenons la photo DEA-Informat (l'exemple 5.7) et supposons qu'il y a une centaine d'étudiants en DEA informatique de l'INPG et qu'un étudiant quelconque a changé sa liste de cours à suivre, par exemple il a choisi un nouveau cours. Cette modification provoque un rafraîchissement

(réévaluation) de DEA-Informat car cette dernière est un historique à VS. Il est évident que cette réévaluation totale est prohibitive. D'autre part cette photo dynamique est définie sur les classes : ETUDIANT, ECOLE et INSCRIPTION et toute modification sur l'une de ces classes provoque en principe une réévaluation de DEA-Informat. Imaginons qu'un étudiant change son adresse, cette modification sur le type source ETUDIANT laisse DEA-Informat inchangé, la réévaluation s'avère inutile et prohibitive. Cet exemple nous montre qu'il y a des modifications sur les types source d'une photo qui ne changent pas son contenu.

La technique d'implantation du mécanisme de rafraîchissement de photos dynamiques doit tenir compte de :

1/ Les modifications sur les types sources ne changeant pas le contenu de la photo à rafraîchir. Dans ce cas le rafraîchissement n'est pas exécuté.

2/ Dans le cas contraire, c-à-d la modification peut changer le contenu de la photo à rafraîchir il faut élaborer une méthode permettant d'éviter le plus possible une réévaluation globale de la requête de définition.

Une méthode dite différentielle a été proposée [KOEN 81] pour éviter la réévaluation globale des données dérivées de la base de données MADAM. L'idée de cette méthode consiste à remplacer n'importe quelle transaction T sur les relations sources par une autre transaction T' sur les relations sources aussi que sur les relations dérivées, où T' est équivalente à la séquence { T, réévaluation des relations dérivées }. Afin d'illustrer cette méthode considérons l'exemple suivant :

Exemple 5.12 :

La photo sur les sommes des salaires de tous les employés de chaque département :

type SALAIRE-TOTAL : *snapshot* (nom-dép, salaire-total)
AS SELECT d.nom, SUM [e.salaire] *by* d
FROM DEPARTM d, EMPLOYE e

Considérons maintenant la transaction T remplaçant l'ancien salaire de l'employé Dupond du département administratif par son nouveau salaire :

```
REPLACE  e.salaire = 12000
FROM     EMPLOYE e
WHERE    e.nom-dep = 'Administratif' AND e.nom = 'Dupond'
```

et la transaction séquentielle T' :

```
REPLACE  s.salaire-total = s.salaire-total - salaire.vieux
FROM     SALAIRE-TOTAL s
WHERE    s.nom-dep = 'Admin'

T

REPLACE  s.salaire-total = s.salaire-total + salaire.nouveau
FROM     SALAIRE-TOTAL s
WHERE    s.nom-dep = 'Admin'
```

où salaire.vieux et salaire.nouveau dénotent le vieux salaire et le nouveau salaire de Dupond. Evidemment T' est équivalente à la séquence {T, rafraîchir SALAIRE-TOTAL}. Dans ce cas T' est appelée différentiel de T par rapport à SALAIRE-TOTAL. La méthode différentielle proposée dans [KOEN 81] consiste à construire pour n'importe quelle transaction T son différentiel par rapport à un ensemble de relations dérivées {V1,V2,..,Vn}. On peut préconstruire les différentiels pour les primitives de mise à jour. T' est alors construite à l'aide de ces différentiels étant donné que toute transaction T peut être décomposée en une séquence de primitives de mise à jour.

Une autre méthode pour éviter la réévaluation globale est proposée dans [BUNE 79] mais pour les alerteurs. Dans cet article on a considéré deux sortes d'alerteur : alerteur-addition et alerteur-suppression correspondant à une addition ou une suppression de n-uplets sur telle ou telle relation dérivée. S'il y en a il faut effectuer l'opération d'alerte. L'exemple de l'alerteur-addition est : diffuser le nouveau tarif de taxe de revenu à tous les employés ayant un salaire supérieur à 20000. Ces employés constituent en fait une vue V(20K) et s'il y a des nouveaux employés avec salaire supérieur à 20000 l'alerte est signalée pour diffuser le tarif ci-dessus à ces nouveaux employés.

La méthode pour éviter la réévaluation globale dans [BUNE 79] est appelée méthode d'évaluation partielle, basée sur le parcours partiel d'arbre algèbrique de la relation dérivée associée à l'alerteur (qui est en fait le résultat d'une requête d'interrogation). A chaque nœud de cet arbre, en fonction de l'opération relationnelle associée, on peut dire que cette

opération peut additionner ou supprimer des n-uplets au résultat. Par exemple si des n-uplets sont ajoutés à la relation opérande de l'opération projection il l'est aussi pour le résultat. Si à la fin du parcours par cette procédure il n'a que des n-uplets qui sont ajoutés à la relation résultat final (la racine de l'arbre) l'alerteur-suppression n'est pas signalée et on n'a pas besoin de réévaluer la relation dérivée associée à l'alerteur.

Cependant cette méthode ne peut pas être appliquée aux photos car pour les photos la suppression ou l'adition de n-uplets sont les mêmes, c-à-d le contenu de la photo est changée et il nous faut la rafraîchir.

Nous allons développer dans ce qui suit une méthode pour détecter des transactions sur la base ne pouvant pas changer le contenu d'une photo dynamique à VS. Dans ce cas-là l'ancien contenu de la photo reste valable et le rafraîchissement n'est pas déclenché.

4.1- Détection des transactions ignorées

Une transaction Tr est dite **ignorée** pour la photo P si après la fin de Tr le rafraîchissement de P ne change pas le contenu de celle-ci. Par exemple un changement d'adresse d'un étudiant est une transaction ignorée pour la photo DEA-Informat parce que l'attribut adresse de l'entité ETUDIANT n'entre pas dans la liste des attributs de DEA-Informat. Notre approche est basée sur ce principe. En effet seulement les transactions portant sur des attributs concernant la photo peuvent ne pas être ignorées pour cette photo. Nous allons établir l'indépendance de la photo P des attributs, dits attributs source (attributs des types source), en terme de vecteur bit. Nous établirerons aussi les attributs dits touchés par la transaction Tr en même forme de vecteur bit. En comparant ces vecteurs bit on peut déterminer si la transaction Tr est ignorée pour P ou non.

Indépendance d'une photo par rapport aux attributs sources

D'abord étudions l'indépendance de chaque attribut de la photo par rapport aux attributs sources. Soit REQ (T1,T2,..,Tn) la requête de définition de P. Soit $i_1,i_2,..,i_n$ les nombres d'attributs de T1,T2,..,Tn respectivement et $i_0 = i_1 + i_2 + .. + i_n$. A chaque photo P on associe un vecteur bit (il ne se compose que de 0 ou 1) de longueur i_0. On le dénote VB(P) et appelle vecteur de dépendance de P des attributs source. Un attribut de Ti est dit concernant P s'il figure dans REQ (T1,T2,..,Tn) c-à-d dans la liste d'extraction de la clause SELECT ou bien dans les conditions logiques de la clause WHERE.

Si le j-ème attribut de Tk concerne P le $(v_k + j)$-ème composant de VB(P) est égal 1 sinon il est 0, où $v_k = i_1 + i_2 + .. + i_{k-1}$, $k >= 2$ et $v_1 = 0$.

Exemple 5.13 :

Considérons la photo DEA-Informat avec T1 = ETUDIANT, T2 = INSCRIPTION et T3 = ECOLE. Alors :

VB(DEA-Informat) = (1000011111000)

Vecteur de touche d'une transaction aux attributs des types de la base

Considérons maintenant une transaction Tr sur la base. On associe à Tr un vecteur bit VB(Tr) de longueur i_0 et l'appelle vecteur de touche de Tr. Un attribut de Ti est dit touché par Tr si :

- Tr modifie cet attribut d'un n-uplet de Ti ou
- Tr insère (supprime) un n-uplet de Ti

Si le j-ème attribut de Tk est touché par Tr le $(v_k + j)$-ème composant de VB(Tr) est égal à 1 sinon il est 0, où $v_k = i_1 + i_2 + .. + i_{k-1}$, $k >= 2$ et $v_1 = 0$.

Exemple 5.14 :

Soit Tr une transaction qui modifie l'adresse de l'étudiant Dupond. Alors :

VB(Tr) = (0010000000000)

Exemple 5.15 :

Tr est une insertion de l'inscription de l'étudiant Gervais. Alors :

VB(Tr) = (0000111110000)

Passons maintenant au lien entre une transaction Tr et une photo P. Formons d'abord le vecteur VB(Tr,P) = VB(Tr) ∧ VB(P) où ∧ dénote l'opérateur "ET" entre les vecteurs bit. On l'appelle vecteur de mesure d'influence de Tr sur P (VMI).

Propriété de VMI :*Si le VMI d'une transaction sur une photo est égal à 0 (comme vecteur) cette transaction est ignorée pour la photo.*

En effet VB(Tr,B) = 0 si et seulement si tous les composants correspondant de VB(Tr) et de VB(P) ne sont pas égaux 1 en même temps. Tout attribut de Ti est alors non touché par la transaction,ou ne concerne pas P. De ce fait le contenu de P reste inchangé après la transaction Tr ou bien Tr est ignorée pour P.

Exemple 5.16 :

Soit Tr une modification d'adresse de certain étudiant et P = DEA-Informat. Alors :

VB(P) = (1000011111000) et VB(Tr) = (0010000000000)

donc VB(Tr,P) = VB(Tr)∧VB(P) = (0000000000000)

Cette modification est, d'après la propriété de VMI ignorée pour DEA-Informat.

Exemple 5.17 :

Soit Tr une insertion d'une nouvelle inscription et soit P identique à DEA Informat. Alors :

VB(Tr) = (0000111110000) et VB(Tr,P) = (000001111000)

Cette fois Tr n'est pas une transaction ignorée pour DEA-Informat.

Le vecteur VB(P) pour une photo dynamique à VS est généré lors de l'interprétation de la définition de P. Il peut être stocké dans le catalogue TIGRE et il peut être stocké en mémoire centrale pendant une session TIGRE. Quant à VB(Tr) d'une transaction Tr il peut être généré à la phase d'analyse de la requête de Tr.

Notre proposition de détecter les transactions ignorées pour les photos n'est pas très efficace. D'une part la détection est assez grosse, elle se porte seulement au niveau global des attributs en laissant la côté de valeur concrète de ces attributs. Prenons une insertion d'un étudiant en MIAG. Cette transaction est en fait ignorée pour DEA-Informat mais son vecteur VMI sur cette dernière n'est pas 0. Autrement dit notre méthode ne permet pas de détecter toutes transactions ignorées, seulement les transactions **sûrement ignorées**. D'autre part notre approche est très relationnelle, c-à-d elle est mieux adaptée à une structure plate des données. Les documents TIGRE n'appartiennent pas à cette catégorie, ils ont une structure arborescente complexe. Malheureusement, dans le cas des documents

notre approche détecte mal les transactions ignorées. Les difficultés résident en ce que pour les chemins de la structure de document aboutissant à des portions à extraire pour des photos, sont soit des gros objets (présentation parenthesée), soit des relations (présentation relationnelle). Consulter ces chemins pour détecter les transactions ignorées s'avère prohibitif et même plus coûteux que la réévaluation des photos.

4.2- Notion de U-filtre

Considérons une transaction insérant seulement l'inscription d'un étudiant en DEA informatique de l'INPG. Cette modification n'ajoute qu'un n-uplet à DEA-Informat (voir l'exemple 5.7). Si l'on détruit tout le contenu de DEA-Informat et qu'on le recalcule, la réalisation du rafraîchissement est assez brutale. Cet exemple nous a conduit à élaborer une méthode utilisée dans certain cas comme l'exemple ci-dessus pour éviter la réévaluation globale des requêtes de définition de photo. Cet exemple nous a aussi suggéré qu'au lieu de tout rafraîchir ne peut-on rafraîchir que ce qui est ajouté (supprimé) ?.

Dans ce qui suit **un type classe T** est vu comme **un ensemble T** des faits.

Prenons d'abord le cas de l'insertion des faits dans un type T. Dans ce qui suit on dénote par **T.v et T.n** les ensembles des faits de T **avant et après** une transaction quelconque Tr respectivement. **T.a et T.s** dénotent les ensembles des faits **ajoutés** et des faits **supprimés** par Tr. Le contenu d'une photo P dynamique à VS (un film) avec la requête de définition REQ (T,T1,..,Tn) après Tr sera le résultat de la réévaluation de cette requête ou bien de REQ (T.aUT.v,T1,..,Tn). On va montrer que dans certaines conditions cette réévaluation peut être remplacée par l'évaluation de la requête REQ (T.a,T1,..,Tn) puis ajouter le résultat à l'ancien contenu de P.

Dans ce cas on dit que le **U-filtre** (pour dénoter l'union) a été utilisé pour rafraîchir la photo P. La technique de filtrage a été proposée dans [BUNE 79] pour les alerteurs. Nous avons mentionné dans le paragraphe précédent que la notion d'alerteur et celle de photos ne sont pas les mêmes. Nous développons ici une méthode adaptée aux photos dynamiques.

Nous commençons par l'examen de la propriété additive des opérateurs algèbriques relationnels.

Soit OP(A1,A2,..,An) un opérateur ensembliste n-aire sur les ensembles A1,A2,..,An dont le résultat est aussi ensemble. Cet opérateur est appelé **U-additif** pour le i-ème opérande si l'égalité :

OP(A1,..,Ai-1,B1∪B2,..,An) = OP(A1,..,Ai-1,B1,..,An) ∪ OP(A1,..,Ai-1,B2,..,An)

a lieu pour tout couple B1,B2 du i-ème opérande.

Un opérateur est appelé **U-additif** s'il est U-additif pour tous ses opérandes.

En particulier on peut considérer l'opérateur de photographie PH(T1,T2,.,Tn) portant sur les types Ti avec la requête de définition REQ (T1,T2,.,Tn) comme un opérateur ensembliste. Cette photo est alors U-additif pour T1 si l'égalité :

PH(A∪B,T2,.,Tn) = PH(A,T2,.,Tn)∪PH(B,T2,.,Tn)

a lieu pour tout couple A,B de T1. Si la photo est U-additive on a l'égalité :

PH(T1.v∪T1.a,T2,.,Tn) = PH(T1.v,T2,.,Tn)∪PH(T1.a,T2,.,Tn)

pour une insertion sur T1. De plus PH(T1.v,T2,.,Tn) et PH(T1.a,T2,.,Tn) représentent l'ancien contenu de la photo avant l'insertion et le contenu de la même photo s'appliquant seulement à l'ensemble de nouveaux n-uplets de T1. De ce fait si une photo est U-additive le U-filtre peut être utilisé. Les critères d'une photo additive seront établis plus loin.

Dans ce qui suit on ne s'interesse qu'aux opérateurs binaires ou unaires parce que tous les opérateurs algèbriques relationnels peuvent être ramenés à un opérateur soit binaire, soit unaire.

On va établir la U-additivité des opérateurs relationnels.

1/ Les opérateurs : selection, projection, jointure, produit cartésien, union et intersection sont U-additifs.

2/ La différence et la division sont U-additives pour le première opérande.

En effet nous avons les égalités suivantes :

E1 : (A1∪A2)[X] = A1[X]∪A2[X] (projection sur X)

E2 : (A1∪A2) : E = (A1 : E)∪(A2 : E) (selection par condition E)

E3 : (A1∪A2)⋈B = (A1⋈B)∪(A2⋈B) (jointure)

$A \bowtie (B1 \cup B2) = (A \bowtie B1) \cup (A \bowtie B2)$

E4 : $(A1 \cup A2) \times B = (A1 \times B) \cup (A2 \times B)$ (produit cartésien)
$A \times (B1 \cup B2) = (A \times B1) \cup (A \times B2)$

E5 : $(A1 \cup A2) \cup B = (A1 \cup B) \cup (A2 \cup B)$ (union)
$A \cup (B1 \cup B2) = (A \cup B1) \cup (A \cup B2)$

E6 : $(A1 \cup A2) \cap B = (A1 \cap B) \cup (A2 \cap B)$ (intersection)
$A \cap (B1 \cup B2) = (A \cap B1) \cup (A \cap B2)$

E7 : $(A1 \cup A2) \setminus B = (A1 \setminus B) \cup (A2 \setminus B)$ (différence)

E8 : $(A1 \cup A2) \div B = (A1 \div B) \cup (A2 \div B)$ (division)

Les preuves de ces égalités ne sont pas compliquées et nous n'entrons donc pas dans le détail des démonstrations.

Les opérateurs de document : project-doc, select-doc, extract-doc et groupe-doc [VELE 84] sont aussi U-additifs.

Quant aux opérateurs agrégat : MIN, MAX, SUM, COUNT ils sont également U-additifs seulement pour SUM et COUNT à condition que les deux opérandes de l'union soient disjoints.

En effet nous avons les égalités suivantes :

E9 : $MIN(A \cup B) = MIN\{MIN(A), MIN(B)\}$

E10: $MAX(A \cup B) = MAX\{MAX(A), MAX(B)\}$

E11: $SUM(A \cup B) = SUM(A) + SUM(B)$ Si $A \cap B = \emptyset$

E12: $COUNT(A \cup B) = COUNT(A) + COUNT(B)$ Si $A \cap B = \emptyset$

N.B : Ici l'opérateur U entre deux nombres est considéré comme l'opérateur d'addition entre ces nombres.

Nous montrons encore une propriété de la U-additivité nécessaire à la suite de ce

paragraphe.

Propriété de U-composition : *Une composition de deux opérateurs U-additifs est également U-additive.*

En effet considérons deux opérateurs U-additifs OP1 et OP2 et leur composition OP2*OP1. Quatre possibilités sont considérées :

1) OP1 et OP2 sont tous les deux unaires

2) OP1 et OP2 sont tous les deux binaires

3) OP1 unaire et OP2 binaire

4) OP1 binaire et OP2 unaire

Prenons le quatrième cas. On a alors :

$$OP2*OP1(A1 \cup A2,B) = OP2(OP1(A1 \cup A2,B)) = OP2(OP1(A1,B) \cup OP1(A2,B))$$

$$= OP2(OP1(A1,B)) \cup OP2(OP1(A2,B)) = OP2*OP1(A1,B) \cup OP2*OP1(A2,B)$$

Passons maintenant au problème de filtrage de photos et de transactions. Supposons que le type T figure dans la requête de définition REQ (T,T1,..,Tn) de la photo P et considérons les cas simples suivants :

- C1 : Il n'y a que des faits ajoutés au type T dans la transaction Tr c-à-d les T1,..,Tn restent les mêmes après Tr

- C2 : T n'apparait qu'une fois dans l'arbre algèbrique de la requête REQ (T,T1,..,Tn)

- C3 : Si dans l'arbre algèbrique, à partir du nœud correspondant à T on monte jusqu'à la racine sans rencontrer l'opérateur AVG et si l'on rencontre sur ce chemin la différence ou la division ce parcours doit se faire toujours à droite.

Si une transaction Tr satisfait la condition C1 on l'appelle **add-transaction** pour le type T. Si un type T satisfait les conditions C2 et C3 en même temps on l'appelle **singulièrement U-additif** pour la photo P.

Critère de la U-additivité d'une photo : *Si T est un type source de la photo P et T est singulièrement U-additif pour P, alors la photo P est U-additive pour le type T.*

En effet les conditions C2 et C3 signifient que le résultat de la requête REQ (T,T1,..,Tn) peut être obtenu par l'application successive des opérateurs U-additifs en réécrivant P(T,T1,..,Tn) = OP(T) où OP est une composition successive des opérateurs U-additifs. D'après la propriété de la U-composition notre photo P est U-additive pour T.

Remarque : Dans le contexte d'insertion de n-uplets les opérateurs SUM et COUNT sont U-additifs d'après E11 et E12 car on a toujours T.v∩T.a = ∅.

Maintenant on peut énoncer les conditions que nous avons évoquées au début de ce paragraphe :

Critère du U-filtre *: Soit P une photo dynamique à VS (ou un film) avec la requête de définition REQ (T,T1,..,Tn) et T singulièrement U-additif pour P. Soit Tr une add-transaction pour T. Alors le nouveau contenu de P après la transaction Tr est égal à l'union de l'ancien contenu avec le résultat de l'évaluation de REQ (T.a,T1,..,Tn), ou bien le U-filtre peut être utilisé pour rafraîchir immédiatement la photo P après la transaction Tr.*

La preuve de ce critère est évident. D'une part la photo P est U-additive pour T et d'autre part on a T.n = T.a∪T.v, d'où :

P(T.n,T1,..,Tn) = P(T.v∪T.a,T1,..,Tn) = P(T.v,T1,..,Tn)∪P(T.a,T1,..,Tn)

Mais P(T.n,T1,..,Tn) est exactement le nouveau contenu de P après Tr
P(T.v,T1,..,Tn) est son ancien contenu (avant Tr) et
P(T.a,T1,..,Tn) est le résultat de l'évaluation de REQ (T.a,T1,..,Tn).

Exemple 5.18 :

Les types ETUDIANT, INSCRIPTION et ECOLE sont singulièrement U-additifs pour la photo DEA-Informat (voir exemple 5.7).

En effet l'arbre algébrique de la requête de définition de DEA-Informat est montré dans la figure 5.2. Chacun de ces types ci-dessus n'apparait qu'une fois dans l'arbre algèbrique et tous les opérateurs de cet arbre sont U-additifs, donc DEA-Informat est U-additif (pour tous ses types source). De ce fait après toute add-transaction pour l'un des types source ETUDIANT, INSCRIPTION et ECOLE on peut utiliser U-filtre pour rafraîchir immédiatement DEA-Informat.

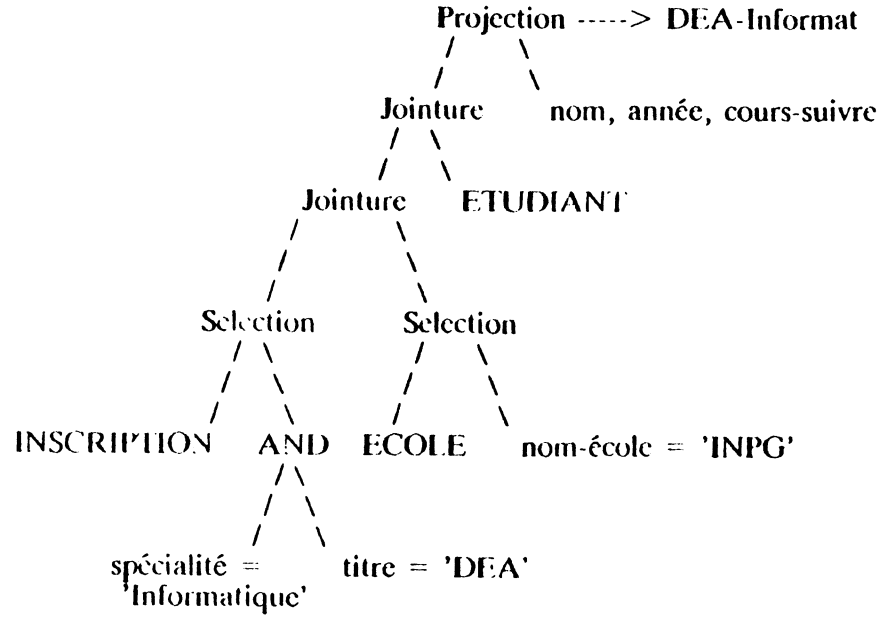

Figure 5.2 L'arbre algèbrique correspondant à la photo DEA-Informat

Remarques :

1/ L'unicité de T dans l'arbre algèbrique (condition C2) est importante. En effet, en supposant que OP est U-additif nous avons :

$OP(A1UA2,A1UA2) = OP(A1UA2,A1)UOP(A1UA2,A2) =$

$OP(A1,A1)UOP(A2,A1)UOP(A1,A2)UOP(A2,A2) \neq OP(A1,A1)UOP(A2,A2)$

2/ Ici le U-filtre est utilisé pour un seul type source de la photo P. Un problème peut se poser immédiatement : pourquoi un seul type ?. Peut-on l'appliquer pour plusieurs types sources ?. La réponse, à notre avis, est non. Supposons que la photo P est U-additive pour les deux types sources T1 et T2 et considérons l'égalité suivante :

$P(T1.vUT1.a,T2.vUT2.a) = P(T1.v,T2.v)UP(T1.v,T2.a)UP(T1.a,T2.v)UP(T1.a,T2.a)$

Si l'on veut appliquer le U-filtre pour plusieurs types sources on doit réévaluer la même requête plusieurs fois avant d'aboutir au résultat final. Ceci est aussi coûteux (même plus coûteux) qu'un rafraîchissement normal.

3/ L'absence de l'opérateur AVG dans la condition C3 pourrait être remplacée par deux opérateurs U-additifs SUM et COUNT. En effet on a toujours AVG(A) = SUM(A)/COUNT(A).

4/ Dans les applications réelles les photos U-additives sont nombreuses, par exemple les photos des exemples 5.1, 5.2, 5.3, 5.7 sont U-additives pour tous les types source et la photo SALAIRE-TOTAL de l'exemple 5.12 est U-additive pour EMPLOYE.

4.3- Notion de D-filtre

D'une façon analogue nous pouvons introduire la notion de D-filtre pour la suppression des faits d'un type source d'une photo dynamique à VS. L'idée de base reste la même : au lieu de recalculer toute la photo P(T,T1,..,Tn), on calcule seulement P(T.s,T1,..,Tn) où T.s est l'ensemble des faits supprimés par une transaction Tr, et puis la supprimer de l'ancien contenu de P. Dans ce cas on dit que le D-filtre est utilisé pour rafraîchir immédiatement P après la transaction Tr.

La démarche est identique à la précédente et nous n'évoquons que les points particuliers.

Nous nous arrêtons tout d'abord sur la **D-additivité** des opérateurs algèbriques relationnels.

Un opérateur ensembliste n-aire OP(A1,A2,..,An) est appelé D-additif pour le i-ème opérande si l'égalité :

$$OP(A1,..,Ai\text{-}1,B1\backslash B2,..,An) = OP(A1,..,Ai\text{-}1,B1,..,An) \setminus OP(A1,..,Ai\text{-}1,B2,..,An)$$

a lieu pour tout couple B1,B2 du i-ème opérande.

Un opérateur est appelé D-additif s'il est D-additif pour tous les opérandes.

Propriété de la D-additivité pour les opérateurs relationnels :

1/ Les opérateurs : selection, projection, jointure, produit cartésien et intersection sont D-additifs.

2/ La différence et la division sont D-additives pour le premier opérande.

3/ L'union n'est pas D-additive ni pour le premier opérande ni pour le deuxième.

Nous avons les égalités suivantes :

E13: $(A1 \setminus A2)[X] = A1[X] \setminus A2[X]$

E14: $(A1 \setminus A2) : E = (A1:E) \setminus (A2:E)$

E15: $(A1 \setminus A2) \bowtie B = (A1 \bowtie B) \setminus (A2 \bowtie B)$
$A \bowtie (B1 \setminus B2) = (A \bowtie B1) \setminus (A \bowtie B2)$

E16: $(A1 \setminus A2) \times B = (A1 \times B) \setminus (A2 \times B)$
$A \times (B1 \setminus B2) = (A \times B1) \setminus (A \times B2)$

E17: $(A1 \setminus A2) \cap B = (A1 \cap B) \setminus (A2 \cap B)$
$A \cap (B1 \setminus B2) = (A \cap B1) \setminus (A \cap B2)$

E18: $(A1 \setminus A2) \setminus B = (A1 \setminus B) \setminus (A2 \setminus B)$

E19: $(A1 \setminus A2) \div B = (A1 \div B) \setminus (A2 \div B)$

Les opérateurs de document sont aussi D-additifs. Quant aux opérateurs d'agrégat nous n'avons que :

E20: $SUM(A \setminus B) = SUM(A) - SUM(B)$ si $B \subseteq A$

E21: $COUNT(A \setminus B) = COUNT(A) - COUNT(B)$ si $B \subseteq A$

De ce fait on doit remplacer la condition C3 par la condition C3' suivante :

- C3' : *Si dans l'arbre algèbrique, à partir du nœud correspondant à T on monte jusqu'à la racine sans rencontrer AVG, MIN, MAX et l'union et si l'on rencontre sur ce chemin la différence et la division ce parcours doit se faire toujours à droite.*

Une transaction ne supprimant que des faits du type T est appelée **supp-transaction** pour T.

Si un type source T de la photo P satisfait C2 et C3' en même temps, on l'appelle **singulièrement D-additif pour P**

Critère de D-filtre : *Soit P une photo dynamique à VS, T un type source singulièrement D-additif pour P et Tr une supp-transaction pour T. Alors le D-filtre peut être utilisé pour rafraîchir immédiatement la photo P après la transaction Tr.*

Remarques :

1/ On a toujours T.s⊆T.v. De ce fait les conditions associées aux égalités E20 et E21 ne posent pas de réels problèmes.

2/ Comme pour la U-additivité, la D-additivité recouvre une large classe de photos. Les photos des exemples 5.1, 5.2, 5.3, 5.7 sont D-additives et la photo SALAIRE-TOTAL de l'exemple 5.12 est D-additive pour EMPLOYE.

5- Photo des photos

Jusqu'ici nous n'avons considéré que les photos pour lesquelles les types sources ne sont pas des photos.

Considérons l'exemple suivant :

Exemple 5.19 :

Construire la photo sur le nom, le dossier bourse, l'école et la spécialité des étudiants résidant à Grenoble :

type ETUD-GRENOBLE : **snapshot** (nom, spécialité, école, dossier)
AS SELECT i.nom, i.spécialité, i.école, e.dossier
FROM INFOR-ETUD i ETUDIANT e
WHERE e.adresse.ville = 'Grenoble'

Nous avons utilisé un trajet qui est un pas vers une photo, semblable à un pas vers une association n-aire [VEL 84b]. La requête de définition de ETUD-GRENOBLE porte sur ETUDIANT et INFOR-ETUD, ce dernier est une photo.

Supposons tout d'abord que la relation "*Le type T1 est un type source de la photo T2*" est une relation asymetrique. Par cette hypothèse nous incluons le problème recursif de la définition des photos.

Si tous les types sources d'une photo sont non-photo on dit que cette photo est une photo du premier ordre.

La définition d'une photo dynamique non du premier ordre peut porter sur des photos source dynamiques ou statiques comme nous l'avons mentionné dans le paragraphe 3. La photo porte toujours sur les versions courantes des types source.

Pour une photo dynamique à VS (un film) non de premier ordre, chaque rafraîchissement de l'un de ses types source provoque un rafraîchissement de cette photo, car le rafraîchissement est une opération de mise à jour. De ce fait si la photo porte sur des photos dynamiques à VS une mise à jour sur la base peut provoquer des rafraîchissements en cascade. Ceci peut dégrader les performances de notre système, il exécute des transactions plus longues qu'en cas d'absence de photos dynamiques à VS. Ici la méthode de détection des transactions ignorées et la méthode de U-filtre et de D-filtre pourraient être appliquées afin d'écourter la durée des transactions de ce type.

CHAPITRE 6

HISTORIQUE ET CHANGEMENTS DU SCHEMA CONCEPTUEL

Le problème de changement du schéma conceptuel a été abordé par la plupart des SGBD relationnels. En général on peut ajouter ou supprimer une relation. Certains SGBD par exemple SYSTEME R [CHAM 76] permet d'ajouter des attributs à une relation. Le SGBD MICROBE [MICR 82] permet de changer interactivement la structure des relations de la base, notamment on peut ajouter ou supprimer certains attributs d'une relation par les commandes du système conçues à cet effet.

Bien qu'une telle restructuration de données coûte assez cher, on ne peut pas l'ignorer. En effet, souvent dans les applications, la structure du monde réel change et le schéma conceptuel correspondant à la représentation de ce monde réel devra à son tour changer. On va montrer dans ce chapitre que l'on est capable de maintenir l'historicité de la base en tenant compte des changements du schéma conceptuel.

En effet le concept d'historique d'objets, proposé dans le chapitre 4, peut s'appliquer à n'importe quel type d'objets de la base. Une telle application aux photos a été décrite dans le chapitre 5. Considérons maintenant le schéma conceptuel comme un type d'objets, auquel on va appliquer le concept d'historique. D'une part, ici le type de version auquel on s'interesse est clairement un **historique à versions successives (HVS)** (on désire maintenir tous les états successifs du schéma), ou un historique à version manuelle (HVM) où l'on maintient d'une façon manuelle quelques états du schéma. D'autre part, la persistance de l'historique dépend de chaque application. Par exemple on peut s'intéresser au dernier état du schéma (en plus de l'état actuel) ce qui correspond à une persistance de 1. Dans ce qui suit nous parlerons des historiques du schéma conceptuel à VS et à persistance illimitée.

1- Changements du schéma conceptuel

Avant tout nous allons déterminer quels changements pourront être autorisés sur un schéma conceptuel. Nous nous limitons au cas de changement d'un type classe ou d'un type document. Ces opérations constituent un échantillon représentatif des transformations significatives que l'on peut être amené à effectuer sur une structure. On peut ajouter et supprimer un type classe quelconque, ajouter ou supprimer des attributs d'un type classe, supprimer, ajouter ou modifier un nœud d'un type document. Exemples:

1) Addition d'un nouveau type

add-schéma <déf-type>

2) Suppression d'un type classe

del-schéma <id-type-classe>

3) Addition d'un nouveau attribut à une classe

add-att <nouveau-att> *to* <id-type -classe>

4) Suppression d'un attribut d'une classe

del-att <id-att> *from* <id-type-classe>

5) Addition d'un nouveau nœud à un document

add-doc <nouveau-nœud> (*before* / *after*) <chemin-doc>

6) Suppression d'un nœud d'un document

del-doc <chemin-doc> *from* <id-type-doc>

7) Changement d'un nœud d'un document

change-doc <chemin-doc> *from* <id-type-doc> *by* <nouveau-nœud>

Pour la commande add-doc nous nous limitons au cas d'agrégat "begin...end", c-à-d un nouveau nœud n'est ajouté qu'à une séquence d'agérégat à la place indiquée par le mot-clé 'before' ou 'after'. Le bout du "chemin-doc" est un (des) nœud(s) d'un type document, de ce fait on doit ajouter un (des) nouveau(x) nœud(s) avant ou après ce(s) nœud(s).

Pour ajouter des constantes à un document on peut omettre 'before' ou 'after' car, par définition, toutes les constantes de document ne jouent aucun rôle dans la structure de document.

Exemple 6.1 :

Supposons que le type *population* :

```
type population : dynamic entity
                  each year
                  last 30
                  with time> month
                    nom-pays : string(15);
                    Nbr-d'habitants : integer;
                    taux-d'homme : (0..100);
                    taux-de-femme : (0..100);
        end;
```

a été prédéfini. Maintenant on veut ajouter la date de recensement au type *population* :

add-att date-recensement : time > hour *to* population

Exemple 6.2 :

Un type document *rapport-recherche* est un agrégat de page-couverture, introduction, corps et conclusion. Le corps est une liste de chapitres.Ces derniers sont des listes de paragraphes. Maintenant on veut ajouter le résumé de tous les chapitres après l'introduction :

add-doc résumé : list(1,*) *of* paragraphe *after* intoduction *of* rapport-recherche

Peut-on maintenir l'historicité d'une base de données TIGRE en tenant compte de tels changements du schéma conceptuel ? "l'historisation" du catalogue TIGRE permet une réponse positive.

2- Historique de la structure de données des types classe dynamiques

Le catalogue TIGRE constitue une BD proprement dite du SGBD relationnel (la méta-base). Cette méta-base contient toutes les caractéristiques des types TIGRE [PALA 83], [VELE 86]. Pour maintenir, par exemple l'historicité d'une classe dynamique dont la structure de données change au cours du temps, on peut historiser de la manière proposée dans le chapiter 4 la relation CATATT, qui contient caractéristiques des attributs des types TIGRE. Chaque fois qu'un attribut est ajouté à une classe dynamique, un n-uplet caractérisant cet attribut est inséré avec le temps associé dans CATATT. Une relation CATATT-H est créée pour stocker les n uplets caractérisant les attributs supprimés.

Notons pour chaque entité (association, agrégation) dynamique E, R(E) et R(E-H) les relations de stockage des données courantes et des données historiques de E. Chaque fois qu'un attribut est ajouté (supprimé) de E, deux relations R(E) et R(E-H) sont figées et remises dans la zone d'historiques. Deux nouvelles relations R(E) et R(E-H) correspondant à la nouvelle structure de données de E sont générées par le système avec ajout ou suppression d'un attribut par rapport aux relations antérieures. Chaque version de la structure de données de E corespond donc à deux relations R(E) et R(E-H) qui maintiennent respectivement les données courantes et les données historiques pour cette version de la structure de données de E.

De ce fait, pour gérer l'historique du schéma conceptuel il nous faut ajouter un catalogue CATRELHIS. Celui-ci gère les relations R(E) et R(E-H) de la classe dynamique E correspondant aux versions (successives ou manuelles) de la structure de données de E. Ainsi en consultant les relations CATATT, CATATT-H, CATRELHIS on peut reconnaitre la structure de données d'un type classe dynamique et les relations le stockant, aussi bien à l'heure actuelle que dans le passé, à une date déterminée.

3- Historique et changement d'un type document dynamique

La structure d'un type document est stockée dans la relation CATREGLE. Pour maintenir l'historicité des types documents dynamiques changeant de structure de données, on historise la relation CATREGLE. Ainsi on crée le catalogue CATREGLE-H pour stocker des règles historiques de documents. Cependant deux catalogues CATREGLE et CATREGLE-H ne sont pas suffisants pour maintenir le fait qu'un noeud peut être ajouté ou supprimé d'un agrégat de document. En effet la relation CATREGLE est organisée de telle façon que, si le document avec constructeur agrégat de cardinalité p est identifié par le surrogate k, les règles composant de cet agrégat sont identifiées par k+1,..,k+p. L'identificateur k+p+1 peut déjà avoir été utilisé, il n'est donc pas discriminant et ne peut être pris pour renuméroter les règles de l'agrégat dans le cas où l'on veut ajouter une nouvelle règle.

Pour cela nous proposons que la cardinalité d'un agrégat de document soit mémorisée dans CATTREGLE et créons un autre catalogue, CATAGRDOC (Règc, Dc, ordre-agr, règlec) signifiant pour chaque n-uplet (i1,i2,i3,i4) que la règle i4 est dans l'ordre de i3 de la règle agrégat i1 du type document identifié par i2, où 1 <= i3 <= cardinalité de l'agrégat. En historisant CATREGLE et CATAGRDOC, on peut gérer l'historicité des documents dynamiques.

4- Changement de la structure d'historiques d'un type TIGRE

La structure historique d'un type TIGRE peut,elle aussi, changer par rapport au temps. Par exemple on peut vouloir modifier la persistance d'un historique, historiser un type statique, ou même "dé-historiser" un historique. En effet on peut considérer les mots clé *'dynamic'*, *'each'*, *'last'*, *'with'* comme des attributs spéciaux d'un historique, dont les valeurs sont prises sur les domaines : booléen, ensemble des valeurs du type période, entier et ensemble des types restreints du temps. Ils sont stockés dans le catalogue CATHIS (voir le chapitre 7).

Ainsi avec, une commande telle que :

change-his <nouvelle-déf-his> *to* <id-type>

on peut faire une modification sur CATHIS. Ainsi pour conserver l'historicité des structures historiques on pourrait historiser CATHIS, cependant la sémantique de cette historicité et son lien avec les données historiques pourrait n'avoir aucun sens.

Examinons maintenant le cas de dé-historisation d'un historique. Ce type est devenu statique. Si l'on ne s'intéresse pas au passé de ce type on supprime le n-uplet correspondant de la relation CATHIS et sa relation stockant des faits historiques, c-à-d on ne garde pour ce type que la relation de la version courante sans l'attribut de temps associé. Cependant pour certaines applications nous nous intéressons à un type dé-historisé (par exemple avant l'instant t_0 le type était historique et à partir de t_0 il est devenu statique mais on veut garder son historicité avant t_0). Dans ce cas on ne supprime pas sa relation stockant des fait historiques. On doit encore ajouter deux attributs à la relation CATHIS qui disent si un historique est déshistorisé et le temps de la déshistorisation. De cette façon on peut maintenir l'historicité de ce type avant t_0 et sa version courante après t_0.

CHAPITRE 7

SGBD TIGRE ET GESTION DU TEMPS ET DES HISTORIQUES

Dans ce chapitre nous allons décrire des extensions de l'architecture du SGBD TIGRE [VELE 85a] qui permettent de prendre en compte la gestion du temps et des historiques.

1- Structure du catalogue TIGRE

La correspondance de schéma entre les modèles TIGRE et relationnel a été établie [PALA 83, PALA 84], [VELE 86]. Un schéma conceptuel et sa correspondance avec un schéma relationnel sont stockés dans un ensemble de relations (la méta-base) appelé le catalogue TIGRE. L'interprétation d'une définition de schéma TIGRE insère des n-uplets dans le catalogue et génère des relations vides correspondantes au schéma. Une extension du catalogue TIGRE pour la gestion des paramètres de documents a été proposée dans [TIGRE 21]. Dans ce chapitre nous allons décrire une autre extension du catalogue TIGRE - celle de la gestion du temps et des historiques.

1.1- Catalogue des types TIGRE

Tout type d'une base de données TIGRE est décrit par au moins un n-uplet dans la relation CATD :

```
CATD ( Dbc : integer;              (*surrogate de la base de donnée*)
       Dc : integer;               (*surrogate du type*)
       domname : string(12)        (*nom du type*)
       oftype : char;              (*spécification du type*)
       datatype : integer;         (*code du type de base simple sur lequel le
                                    type est défini du point de vue du SGBD*)
       min, elemax : integer; );   (*pour quelques types, bornes inférieures
                                       et/ou supérieures*)
```

Les types TIGRE se divisent en trois catégories : types de base, types construits et types classe. Les types de base sont "classiques" : integer, real, boolean, string de longueur fixe, et les types "multimédia" : texte, image, graphique (cercle, rectangle), symbole mathématique.

Les type de temps sont aussi des types de base (voir le chapitre 3) : type temps simple, cinq types temps restreints, type durée, type intervalle de temps et type période (9 au total), soi en tout 18 types de base, ayant comme surrogate 1,2,..,18 et prédéfinis dans le système.

1.2- Catalogue des historiques

Une relation CATHIS est ajoutée au catalogue TIGRE pour stocker les caractéristiques des historiques dans la base TIGRE. La relation CATHIS a la structure suivante:

```
CATHIS (Dbc : integer ;              (*surrogate de la base*)
        Dc : integer;                (*surrogate du type historique*)
        Type-version : char;         (*HVP, HVM ou HVS*)
        Période : string(32);        (*indique la périodicité si HVP*)
        Persistance : integer ;      (*persistance de l'historique*)
        temps-associé : integer;     (*surrogate du type de temps associé*)
        explicité  : boolean ; );    (*si la définition historique
                                              est explicite*)
```

Ainsi chaque n-uplet de la relation CATHIS est le descripteur d'un historique. En consultant CATHIS on peut récupérer toutes les caractéristiques des historiques : persistance, type de temps associé, type de version (HVM, HVP ou HVS)...C'est ainsi que nous stockons et manipulons les données historiques.

2- Relations de la zone d'historiques

Comme nous l'avons montré dans le chapitre 4, les données historiques sont stockées dans la zone d'historiques afin de ne pas pénaliser les accès aux versions courantes. Les relations de cette zone sont décrites dans ce qui suit.

Il faut noter avant tout que le système ajoute un attribut de temps associé à toutes les relations liées à des historiques (pour la version courante et aussi pour les données historiques). Cet attribut est invisible à l'usager. Pour les relations stockant des données historiques un attribut d'indication de suppression est aussi ajouté par le système.

2.1- Relation associée à un type renommé historique

Dans le cas d'absence d'historiques, un type renommé TIGRE ne génère pas de relations dans la base. Par contre un type renommé historique a une relation associée avec le même nom et un suffixe -H. Cette relation a trois attributs. Le premier représente le surrogate d'un fait qui possède une propriété représentée par la valeur du deuxième attribut. Le dernier attribut comporte le temps associé.

Exemple 7.1 :

Avec la définition :

type N-de-visiteurs : *dynamic integer*
with time> hour;

La relation N-de-visiteurs-H (Sc, N-de-visiteur, temps-ass) est générée dans la base.

2.2- Relations associées à des enregistrements ou tableaux historiques

La définition d'un type Array ou Record se traduit par la création d'une relation par type, comme le montre le schéma suivant :

```
type R = record                    Relation R (Rc : integer; (*surrogate*)
       c1 : T1;                                c1 : T1;
       c2 : T2;          ===>                  c2 : T2;
          .                                       .
          .                .
       cn : Tn;                                cn : Tn;) ;
    end;

type T : array[M] of S;  ===>  Relation T (Tc : integer; (*surrogate*)
                                           ordre: integer; (*de 1 à M*)
                                           valeur : S; ) ;
```

Pour un enregistrement historique une autre relation est générée

```
Relation R-H (Rc : integer;     (*surrogate*)
              c1 : T1;
              c2 : T2;
                 .
                 .
              cn : Tn;  ) ;
```

qui permet de stocker des valeurs historiques de ce type. Il en est de même pour la Relation T d'un tableau historique.

2.3- Documents historiques

Les occurrences de document d'une base de données TIGRE sont stockées de la façon suivante : une relation, DOCSTRUCT, stocke la structure logique des documents, à raison d'un n-uplet par élément de structure. Les unités (feuilles de l'arborescence) volumineuses des documents (texte, image), appelées "gros objet", sont découpées en fragments de taille égale à la taille maximale de n-uplets du SGBD sousjacent, et stockés comme des n-uplets. Ceci est représenté en relationnel par la relation PAVES, qui stocke les fragments (les pavés) des gros objets. Les unités qui ne sont pas volumineuses seront stockés dans les n-uplets de la relation DOCSTRUCT. La relation PAVES a la structure suivante :

```
PAVES (Docc : integer ;
       elem : integer;
       numpav : integer;        (*numéro du pavé*)
       contenu : LONG ; );      (*contenu du pavé*)
```

Pour gérer des documents historiques il nous faut historiser les deux relations ci-dessus, DOCSTRUCT et PAVES, par exemple la relation PAVES-H a la structure suivante :

```
PAVES-H (Docc : integer;
         elem : integer;
         numpav : integer;
         contenu : LONG;
         temps-ass : time ; );
```

De ce fait la gestion des documents historiques est faite via un sous-ensemble des relations du catalogue TIGRE concernant les documents et les gros objets. Il n'y a pas de zone d'historiques proprement dite pour les documents historiques. Pour retrouver un document historique (ou une partie de ce dernier) il faut d'abord retrouver les n-uplets concernant le document dans le sous-ensemble ci-dessus. Cependant les pavés de la version courante, qui ne sont plus utilisés, ne sont pas libérés à chaque modification. Ceci est effectué dans le cas d'absence d'historiques.

2.4- Relations associées à des classes historiques

Comme pour les enregistrements et les tableaux historiques, une relation avec le même nom et suffixe -H est générée pour une entité historique. Celle-ci stocke des faits historiques de l'entité. Pour les entités dérivées par agrégation et pour les associations, le principe reste le même : historiser les relations représentant et stockant ces historiques. Nous n'entrons pas dans les détails de cette historisation.

2.5- Exemple d'illustration

Exemple 7.2 :

Considérons maintenant l'entité historique EMPSAL de l'exemple 4.1, et supposons que l'on a les transactions suivantes sur EMPSAL :

1. A l'instant t1 insérer (Dupont, p1, 6000) et (Martin, p2, 7000)
2. A l'instant t2 modifier (Dupont, p2, 8000)
3. A l'instant t3 modifier (Dupont, p3, 9000)
4. A l'instant t4 modifier (Martin, p5, 8500)
5. A l'instant t5 insérer (Savio, p6, 6000)
6. A l'instant t6 supprimer Dupont

Les relations EMPSAL et EMPSAL-H finale sont alors illustrées par la figure 7.1 à la page suivante.

3- Architecture du SGBD TIGRE

L'architecture du SGBD TIGRE a été décrite dans le chapitre 2. Pour implanter les fonctionnalités concernant la gestion du temps et des historiques, il faut introduire les opérateurs de manipulation de données temporelles et de données historiques. Tous les opérateurs relationnels et opérateurs de document peuvent être appliqués sur les données temporelles (T-opérateurs) et sur les données historiques (H-opérateurs). Par exemple la liste parenthésée de la requête de l'exemple 3.1 est celle de la figure 7.2 et celle de l'exemple 4.4 est donnée par la figure 7.3

Nom	Poste	Salaire	Temps
Martin	p5	8500	t3
Savio	p6	6000	t5

Relation EMSSAL

Nom	Poste	Salaire	Temps	Indication de suppr
Dupont	p1	6000	t1	
Martin	p2	7000	t1	
Dupont	p2	8000	t2	
Dupont	p3	9000	t3	
Dupont	p3	9000	t6	supprimé

Relation EMPSAL-H

Figure 7.1 Relations EMPSAL et EMPSAL-H

```
Project (
        Select-doc (
                project-doc (
                        ETUDIANT, dossier,
                        parameter durée-bourse),
                T- > '12 month'),
        nom, adresse)
```

Figure 7.2 Liste parenthésée de la requête de l'exemple 4.3

```
H-Select (last version,
        Project (
                Select ( EMPLOYE'
                        nom = 'Dupont'),
                adresse),
                        )
```

Figure 7.3 Liste parenthésée de la requête de l'exemple 4.4

Le résultat de la traduction d'une requête se compose donc d'un ou plusieurs arbres constitués d'opérateurs relationnels, d'opérateurs document, de T-opérateurs et de H-opérateurs. Ces arbres sont interprétés à l'exécution par l'interpréteur TIGRE. Ce dernier comprend un module spécial : gestionnaire du temps et des historiques, chargé d'interpre-

ter les T-opérateurs et les H-opérateurs.

Cette architecture du SGBD TIGRE qui prend en compte la gestion du temps et des historiques est illustrée par la figure 7.4.

Le gestionnaire du temps et des historiques comprend des primitives pouvant interpreter les T-opérateurs et les H-opérateurs. Pour pouvoir interperter les T-opérateurs et les H-opérateurs, ces primitives appellent l'interface SGBD et le gestionnaire de documents. Les documents peuvent comprendre des données temporelles, donc au cours des interprétations des opérateurs document, le gestionnaire de documents peut appeler le gestionnaire du temps et des historiques.

Tous les modules du SGBD TIGRE doivent comprendre les énoncés associés au temps et à des historiques. Par exemple, lors de la définition d'un historique, l'interpréteur des énoncés de définition doit générer la relation correspondante dans la zone d'historiques et propager son historicité à ses types composants. Il en est de même pour l'analyseur des énoncés de manipulation et le générateur d'arborescences.

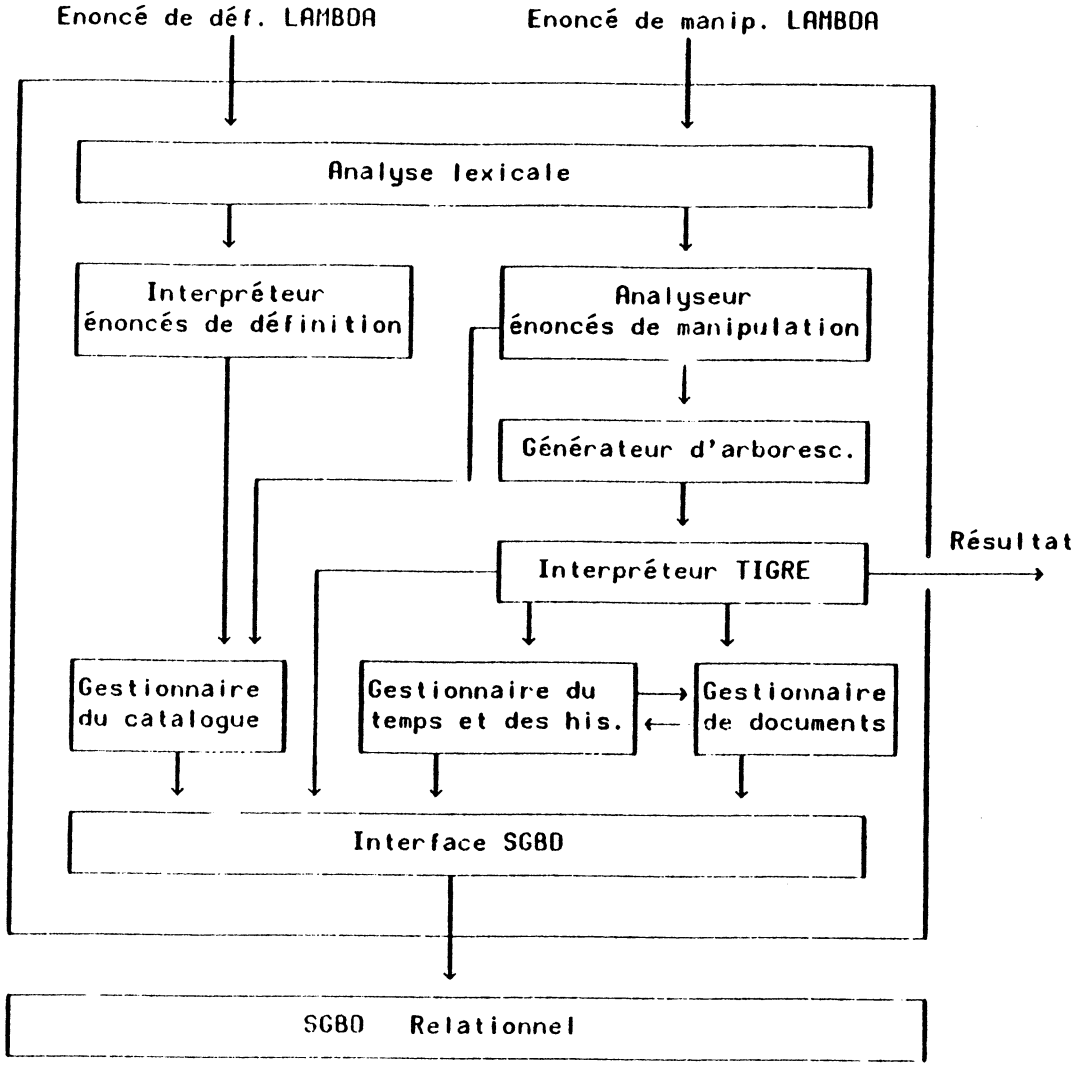

Figure 7.4 L'architecture du SGBD TIGRE étendu pour la gestion du temps et des historiques

CONCLUSION

Au long ce cette thèse nous avons étudié différents aspects liés à la notion de temps dans les bases de données généralisées.

Dans un premier temps, le modèle de données généralisées TIGRE a été étendu par l'inclusion de différents types de temps comme types de base. En prenant comme hypothèse que la bureautique avait été choisie comme application représentative, nous avons offert toutes les catégories de temps rencontrées dans la bureautique : *types de base (points de temps), intervalles de temps, durées et périodes*. Différentes *fonctions* et *opérations* sur le temps ont été définies pour exprimer les *liens sémantiques* entre les types de temps, et à travers ces fonctions et opérations nous pouvons manipuler les données temporelles depuis LAMBDA.

Nous avons ensuite défini une *structure d'historiques* pour les types TIGRE. Celle-ci est basée sur les concepts de *versions temporelles*, de *persistance* et de *modification*. Comme versions nous avons choisi trois types les plus représentatifs : *version manuelle, version périodique, version successive*. Nous avons appliqué la structure d'historiques au modèle *sémantique* et *généralisé* TIGRE. Ainsi un type TIGRE peut avoir deux structures : structure de données et structure d'historiques. Une fois qu'un type est défini comme historique, son historicité est *propagée* aux types composants selon un ensemble de règles de propagation. Un type statique possède une structure d'historiques vide et seule la valeur courante est retenue dans la base. Nous avons aussi proposé des extensions à LAMBDA permettant de manipuler les données historiques dans TIGRE.

La notion d'historique peut se généraliser, c'est ainsi que nous l'avons appliqué au *schéma* de la base pour gérer l'historique de son évolution et également à la notion de photographie d'une base de données. Les photographies sont introduites dans TIGRE comme un nouveau types de données. Associée à ce type on fournit l'opération de rafraichîssement avec référence au passé ou au futur. Par conséquent nous avons été amenés à définir la *notion d'album* (photo dynamique à VM ou à VP) et la *notion de film* (photo dynamique à VS), qui correspondent à la suite des vues et des photographies sur une base de données. Nous avons également étudié le problème de rafraîchissement de film afin d'éviter le plus possible la *réévaluation globale* de la requête de définition.

L'architecture du SGBD TIGRE est étendue par un module spécial pour la gestion du temps et des historiques. Ce dernier est chargé d'interpreter les T-opérateurs (sur les données temporelles) et les H-opérateurs (sur les données historiques) de l'arbre algèbrique d'une requête LAMBDA. Nous avons également décrit les relations de la zone d'historiques.

A notre avis, l'originalité de notre travail est la suivante :

1. Introduire les types de temps, les fonctions et les opérations associées dans un modèle de données généralisées.

2. Définir la structure d'historiques basée sur le concept de versions temporelles et l'appliquer sur un modèle de données sémantique et généralisé.

3. Offrir la possibilité de manipuler les données temporelles et les données historiques d'une façon homogène depuis le langage de manipulation de données.

4. Définir les notions de photographie dynamique, d'album et de film et étudier le problème de rafraîchissement de films.

En ce qui concerne l'implantation, nous avons réalisé un ensemble de procédures pour gérer les types de temps, ainsi que toutes les fonctions et opérations sur ces types. Nous avons incorporé la gestion du temps et des historiques dans la première version du langage de définition de données. On peut définir des types TIGRE sur les types de temps ainsi que définir des types historiques. Lors de la définition d'un historique, une relation est générée dans la zone d'historiques et la propagation d'historicité est effectuée sur ses types composants. Tout cela est écrit en PASCAL et tourne sous le système relationnel MICROBE de MULTICS.

Le changement du SGBD relationnel sousjacent et la contrainte du temps nous ont empêché d'incorporer la gestion du temps et des historiques dans le langage de manipulation de données qui est en cours de réalisation sur le système relationnel ORACLE.

Il est souhaitable de prolonger ce travail dans les directions suivantes :

- Réalisation complète de la gestion du temps et des historiques dans le SGBD TIGRE. Nous avons créé la zone d'historiques pour stocker les données historiques et à l'heure actuelle, cette zone est gérée par le SGBD relationnel sousjacent. Dans la mesure où les documents historiques sont en général

volumineux il faudrait étudier et évaluer des structures plus adaptées et offrant des performances acceptables.

- Etablissement du lien entre le concept de transaction et la gestion des historiques. D'abord il faudrait préciser l'usage des données historiques pour la reprise après panne et le contrôle d'accès concurrents aux données. Sur ce point on peut trouver des éléments dans [DADA 84]. Ensuite il faudrait étudier le rapport entre la transaction au sens du modèle TIGRE [TIGRE 30] et la gestion des historiques. En particulier, cela repose le problème des transactions longues.

- Définition formelle de l'algèbre temporelle pour un modèle Entité-Association. Dans [CLIF 85] on a tenté de construire cette algèbre pour le modèle relationnel (non en première forme normale) et de la comparer avec l'agèbre de CODD pour le modèle relationnel statique. De plus il faudrait songer à établir la complétude du langage de manipulation de données, étendu pour la manipulation des données temporelles et historiques, par rapport à cette algèbre.

- Application des concepts proposés dans ce travail au contrôle d'intégrité dynamique des données. Cela a été proposé pour le modèle Z d'Abrial par la logique temporelle dans [ABRI 78]. Plus généralement, comment le modèle d'objets avec des aspects dynamiques (opérations, comportements et événements associés, trigger et alerteur [ESWA 76], [ROLL 79], [BUNE 79], [BROD 81], [CAUV 86]) pourrait prendre en compte les concepts proposés dans ce travail?

BIBLIOGRAPHIE

[ABRI 78] J.R.Abrial "*Introduction d'une logique temporelle dans le langage Z (Z/8)*" Notes non publiées 1978

[ADIB 80] M.Adiba, B.Lindsay "*Database snapshots*"
Proceedings of 6th VLBD Montreal,Canada, Oct 1980

[ADIB 81] M.Adiba "*Derived relation : A unified mechanism for views,snapshots and distributed data*"
Proceedings of 7th VLBD Cannes, France, Sept 1981

[ADIB 83] M.Adiba "*La gestion du temps dans les SGBD*"
Dans "Bases de Données : Nouvelles perspectives"
Rapport du Groupe BD3 INRIA Janv 1983

[ADIB 84] M.Adiba, F.Azrou
"*An authorization mechanism for generalized DBMS*" 3rd Seminar
On distributed Data sharing Systems, Parma, Italy, March 1984

[ADIB 85a] M.Adiba, N.Bui Quang, J.Palazzo
"*Time concepts for generalized data bases*"
ACM Annual Conference, Denver Colorado, USA, Oct 1985

[ADIB 85b] M.Adiba, N.Bui Quang, J.Palazzo
"*Notion de temps dans les bases de données généralisées*"
AFCET-Modèles et Bases de Données, N 1 Oct 1985

[ADIB 86a] M.Adiba, N.Bui Quang
"*Aspects historiques dans les bases de données généralisées*"
Actes des Deuxièmes Journées "Bases de Données Avancées"
INRIA, Giens (Var) Avril 1986

[ADIB 86b] M.Adiba, N.Bui Quang
"*Historical multi-media Databases*"
Proceedings of 12th VLDB, Kyoto Japan, Aug 1986

[ADIB 86c] M.Adiba, N.Bui Quang
"Dynamic snapshots, albums and movies"
Soumis à publication

[ANDE 81] L.Anderson *"The Database semantics of time"*
Doctoral Thesis, University of Washington, Jan 1981

[ANDE 82] L.Aderson *"Modelling Time at the conceptual level"*
Proceedings of International Conference on DB
Academic Press , Israel 1982

[AVRI 83] G.Avriav, J.Clifford, M.Jarke
"Panel on time and Data bases"
Proceedings of ACM SIGMOD Conference 1983

[BEN 86] C.Ben Amouzegh, C.Chisment, G.Zurfluh
"Bases d'informantions généralisées - Concept de version"
Actes des Deuxièmes Journées "Bases de Données Avancées"
Giens (Var) Avril 1986

[BOLO 82] A.Bolour, L. Anderson, J.Dekeyser, T.Wong
"The role of Time in Information processing: A survey"
ACM SIGMOD Record 12.3 April 1982

[BOLO 83] A.Bolour, J.Dekeyser *"Abstraction in temporal information"*
Inform systems Vol 8 1983

[BRAB 85] F.Barbic, B.Pernici
"Time modeling in office information systems"
Proceedings of ACM SIGMOD Conference 1985

[BRAC 83] G.Bracchi, B.Pernici
"SOS: A conceptual model for office information systems"
ACM 1983 Vol 5

[BREU 79] B.Breuman, F.Falkenberg, R.Mauer
"CSL-A language for defining conceptual schemas"
Data Base Architerture, North Holland, June 1979

[BROD 81] M.Brodie *"On modeling behavioural semantics of databses"*
Proceedings of 7th VLDB Cannes, France, Sept 1981

[BUBE 77] J.A.Bubenko *"The temporal dimension in information modeling"*
Architecture and Models in Data Base Management Systems
North Holland 1977

[BUBE 80] J.A.Bubenko *"Information modeling in the context of system developement"* Proceedings of IFIP Congress 1980

[BUI 84] N.Bui Quang *"Notion de temps dans les bases de données généralisées"*
Rapport de DEA, IMAG Grenoble, Juin 1984

[BUI 85] N.Bui Quang *"Gestion des historiques pour la base de données généralisées TIGRE"*
Rapport de Recherche TIGRE 29, BULL-IMAG, Juin 1985

[BUNE 79] P.Buneman, K.Clemons
"Efficiently Monitoring Relational Databases"
ACM TODS, V 4, N 3, Sept 1979

[CAUV 86] C.Cauvet, J.Y.Lingat, P.Nobcourt, C.Rolland
"RUBIS : Une interface d'aide à la gestion des aspects dynamiques d'une base de données relationnelle"
Actes des Deuxièmes Journées "Bases de Données Avancées"
INRIA, Giens (Var), Avril 1986

[CHAM 76] D.D.Chamberlin and al.
"SEQUEL 2 : A unified Approach to Data Definition, Manipulation and Control"
IBM Journal of Research and Development Nov 1976

[CHEN 76] P.P.Chen "*The Entity Relationship Model - Toward a unified view of data*" ACM TODS, Vol 1, N 1, 1976

[CLIF 83] J.Clifford, D S.Warren
"*Formal semantics for time in database*"
ACM TODS June 1983 Vol 8 N 2

[CLIF 85] J.Clifford, A.Tansel
"*On an algebra for historical relational Databases: two views*"
Proceedings of ACM-SIGMOD Conference 1985

[CODD 70] E.F.Codd "*A Relational Model of Data for Large Shared Data Banks*"
Communications of ACM, V 4, N 4, Dec 1970

[CODD 79] E.F.Codd "*Extending the data base relational model to capture more meaning*"
ACM TODS, Vol 4, N 4, 1979

[COLL 85] C.Collet "*Les formulaires multi-média*"
Actes INFORSID 1985, Luchon Mai 1985

[DADA 84] P.Dadam, V.Lum, H.D.Werner
"*Integration of Time versions into a Relational Databases System*"
Proceedings of 10th VLDB, Singapour, Aug 1984

[DITT 85] K.R.Dittrich, R.A.Lorie
"*Version support for engineering database systems*"
IBM Research Report, RJ 4769 7/18/85

[ESWA 76] K.P.Eswaran "*Specifications, Implementations and Interaction of a Trigger subsystem in an integrated Data Base System*"
IBM Research Report RJ 1820, Aug 1976

[GADI 84] S.K.Gadia, J.H.Vaishnav
"*A Query Language for A Homogeneous Temporal Database*"
Proceedings of Conf on Principles of Database Systems, Apr 1985

[IBM 78] IBM Corporation
"Query-By-Example program description/operations manual"
Sept 1978

[KATZ 82] R.H.Katz, T.J.Lehman
"Storage structure for Versions and Alternatives"
Research Raport (479), University of Wisconsin, Madison, May 1982

[KLOP 81] M.R.Kloppioge *"TERM : An approach to include the Time dimension in the Entity-Relationship Model"*
Proceedings of 2nd Inter Conf on E-R Approach, Washington 1981

[KLOP 83] M.R.Kloppioge, P.C.Lockmann
"Modelling information preserving Database: Consequences of the concept of Time"
Proceedings of 9th VLDB, Florence 1983

[KOEN 81] S.Koenig, R.Paige
"A Transformational framework for The Automatic Control of Derived Data"
Proceedings of 7th VLDB, Cannes France, Sept 1981

[LOPE 83] M.Lopez, J.Palazzo Oliveira, F.Velez
"The TIGRE data model" Rapport de Recherche
TIGRE N 2, BULL-IMAG, Grenoble Nov 1983

[LUM 84] V.Lum and al.
"Designing DBMS support for the Temporal dimension"
Proceedings of ACM-SIGMOD Conference, June 1984

[MAIO 86] R.Maiocchi, B.Pernici
"Time reasoning in the office environment"
Notes internes, Politecnico di Milano, Italy 1986

[OVER 82] R.Overmyer, M.Stonebraker
"Implementation of a time expert in a database system"
ACM-SIGMOD Vol 12 N 3 April 1982

[PALA 83] J.Palazzo Oliveira, F.Velez
"La correspondance de schémas entre les modèles TIGRE et relationnel"
Rapport de Recherche TIGRE N 5, BULL-IMAG, Nov 1983

[PALA 84] J.Palazzo Oliveira
"Un modèle de données et sa representation relationnelle dans un systeme de données generalisées. Projet TIGRE"
Thèse de Docteur Ingénieur, INPG Grenoble, Juin 1984

[RIEU 86] D.Rieu *"Nature, état et dynamicité de l'objet CAO"*
Actes des Deuxièmes Journées "Bases de Données Avancées"
INRIA, Giens (Var) Avril 1986

[ROLL 79] C.Rolland, S.Leifert, C.Richard
"Tools for Information System dynamic Management"
Proceedings of 5th VLDB, Rio de Janicro 1979

[SCHI 83] U.Schiel *"An Abstract Introduction to the Temporal Hierarchic Data Model-THM"*
Proceedings of 9th VLDB Florence 1983

[SNOD 84] R.Snodgrass *"The Temporal Query Language TQuel"*
Proceedings of 3rd ACM SIGACT-SIGMOD Symposium on Principle Database Systems, Waterloo, Canada, April 1984

[SNOD 85] R.Snodgrass, I.Ahn
"A Taxionomy of Time in Databases"
Proceedings of ACM-SIGMOD May 1985

[STON 75] M.Stonebraker *"Implementaion of integrity contraints and views by query modification"*
Proceedings of ACM-SIGMOD, May 1975

[STON 76] M.Stonebraker,E.Wong, P.Kreps
"The Design and Impementation of INGRES"
ACM TODS, V 1 N 3, Sept 1976

[STON 80] M.Stonebraker, K.Keller
"Embeding Expert knowledges and hypothetical Databases into a Database system"
Proceedings of ACM-SIGMOD, May 1980

[TODD 76] S. Todd *"The Peterlee Relational Test Vehicle - A System Overiew"*
IBM Systems Jounal 15,4 1976

[VELE 84] F.Velez *"Un modèle et un langage pour les bases de données generalisées"*
Thèse de Docteur Ingénieur INPG Grenoble Sept 1984

[VELE 85a] F.Velez *"La prise en compte des documents structurés dans un SGBD : Aspects modèle, langage, et architecture du système"*
Actes Premières Journées "Bases de Données Avancées"
INRIA, St Pièrre de Chartreuse, Mars 1985

[VELE 85b] F.Velez *"LAMBDA : An Entity-Relationship based Language for the Retrieval of Structured Documents"*
Proceedings of 4th Int Conf on E-R Approach, Chicago, Oct 1985

[VELE 86] F.Velez *"Structure du catalogue TIGRE et représentation relationnelle d'un schéma TIGRE"*
Notes internes, Centre de Recherche BULL, Grenoble Mars 1986

[WIED 75] G.Wiederhold, J.F.Fries, S.Weyl
"Structured organization of clinical Databases"
Proceedings of AFIPS National Computer Conf, Anhein 1975

ANNEXE

A1- SYNTAXE DES ENONCES DE DEFINITION LAMBDA ASSOCIES AU TEMPS

Nous utilisons une gramaire BNF étendue dont les nouveaux symbolos sont :

[x] : 0 ou 1 instance de x ; x|y : x ou y

x^* : 0 ou plusieurs instances de x ; (x|..|y) : groupements des alternatives multuellement exclusives

x^+ : 1 ou plusieurs instances de x ; 'x' : x est un méta-symbole litéral

<schéma-def-LAMBDA>::= 'Define'<nom-db> <énoncé-def-type>$^+$ 'end.'

<énoncé-def-type>::= ('type' <def-type> ';')$^+$

<def-type>::= <id-type> ':' [<def-struct-hist>] <def-struct-données>

<id-type>::= <ident>

<def-struct-his>::='dynamic'|'each'(<date-period>|<jour-semaine>
|<élément-temps>] | ['last'<entier-sans-signe>]|['with'<type-temps-restr>]

<def-struct-données>::= (<type-de-base>|<type-construit>|<type-classe>)

<type-de-base>::= <type-de-base-simple>|<type-de-base-restr>|<type-temps>

<type-temps>::= <type-temps-simple>|<type-temps-restr>|
<type-intervalle-temps>|<type-durée>|<type-période>

<type-temps-simple>::= 'time'

<time>::='<year>/<month>/<day> <hour>h<minute>:<second>'|'now'

<year>,<month>,<day>,<hour>,<minute>,<second>::= <entier-sans-signe>

<type-temps-restr>::= 'time > '('month'|'day'|'hour'|'minute'|'second')

<time>month> ::= '<year>'

<time>day> ::= '<year>/<month>'

<time>hour> ::= '<year>/<month>/<day>'

<time>minute> ::= '<year>/<month>/<day> <hour>h'

<time>second> ::= '<year>/<month>/<day> <hour>h<minute>'

<type-intervalle-temps>::= 'time-interval'

<time-interval>::='<time>-<time>'|

'<value-time-restr-type>-<value-time-restr-type>'

<type-durée>::= 'duration'

<duration>::='<n1>y<n2>m<n3>d<n4>h<n5>:<n6>'

<n1>::= 0001|0002|...|9999

<n2>,<n4>,<n5>,<n6>::= 01|02|...|99

<n3>::= 001|002|...|999

<type-période>::= 'periodic'

<periodic>::='<date-période>'|'<intervalle-période>'|<jour-semaine>

<intervalle-période>::= <date-période>'-'<date-période>

<date-period>::= <élément-période>|<non-élément-période>

<élément-période>::=('month'|'day'|'hour'|'minute'|'second')

':='<entier-sans-signe>

<non-élément-period>::= <valeur-period-restr-type>

<period-restr-type>::=('year'|'month'|'day'|'hour'|'minute')

'>'('time'|<type-temps-restr>)

<jour-semaine>::= 'Su'|'Mo'|'Tu'|'We'|'Th'|'Fr'|'Sa'

<élément-temps>::= 'year'|'month'|'day'|'hour'|'minute'|'second'

CONTRAINTES SUR LES TYPES DE TEMPS

(second >= 0) AND (second <= 59) AND (minute >= 0) AND
(minute <=59) AND (hour >= 0) AND (hour < 24) AND (day > 0)
AND (day <= 31) AND (month >= 1) AND (month <= 12) AND
(year >= 1582) AND (year <= 9999) AND
((day <> 31) OR (month in (1,3,5,7,8,10,12))) AND
(((day <> 30) OR (month <> 2))
AND (((month <> 2) OR (day <> 29)) OR ((year MOD 400 = 0)
OR ((year MOD 4 = 0) AND (year MOD 100 <> 0))))

A2- SYNTAXE DES ENONCES DE MANIPULATION LAMBDA ASSOCIES AU TEMPS

<expression-de-sélection> ::= 'select'['unique']<expression-de
-valeurs>'from'<trajet-de-designation>[<clause-where>]
<expression-de-valeurs> ::= <valeur>(','<valeur>)*
<valeur> ::= <valeur-de-fait>|<valeur-inter-fait>|<valeur-de-temps>
<valeur-de-temps> ::= (<fonction-sur-intervalle>|<fonction-de-sélection>)
'('<valeur-non-derivée-primaire>|<valeur-de-temps>')'
| <valeur-tronquée>
<valeur-tronquée> ::= 'trunctime''('(<valeur-non-derivée-primaire>
|<valeur-de-temps>) ',' ('month'|'day'|'hour'|'minute'|'second') ')'
<valeur-non-derivée-primaire> ::= <valeur-attribut>|<valeur-attr-tableau>
|<valeur-attr-enregistrement>|'parameter'
<id-param> 'of'<valeur-attribut>

<fonction-sur-intervalle>::='duration'|'begin'|'end'

<fonction-de-sélection>::='year'|'month'|'day'|'hour'|
'minute'|'second'|'week'|

<enonce-de-mise-à-jour>::=<insertion>|<suppression>
|<modification>|<modification-de-temps>

<modification-de-temps>::=<affect-var><requête-un-fait>
<operation-de-temps><modification>

<operation-de-temps>::=<variable>':='(<oper-de-addition>|
<oper-de-soutraction>|<oper-de-multiplication>|
<oper-pseudo-ensembliste>|<oper-modif-granularité>)

<affect-var>:=<variable>['('<id>(','<id>)*')']':='

<oper-de-addition>::= ('move-foward'|'extend') <valeur-à-modifier>
'by' (<valeur-à-modifier>|<const-type-durée>)

<oper-de-soustraction>::= ('move-back'|'reduce') <valeur-à-modifier>
'by'(<const-type-durée>|<const-type-temps-simple>
|<const-type-temps-restr>|<valeur-à-modifier>)')'
|'time-distance' '(' <valeur-de-temps> ',' <valeur-de-temps> ')'

<oper-de-mutiplication>::='multime' '('<entier-sans-signe>
','<valeur-à-modifier>')'

<oper-modif-granularité>::='trunctime' '('<valeur-à-modifier>
',' ('month'|'day'|'hour'|'minute'|'second') ')'

<oper-pseudo-ensembliste>::= ('uniontime'|'insertime') '('valeur
-à-modifier>','(<const-type-intervalle-temps>|<valeur-à-modifier)')'
|('incltime'|'intime')'('<valeur-de-temps>','<valeur-de-temps>')'

<valeur-à-modifier>::=<variable>|<variable>'.'<ident>

<expression>::=<const-de-base>|<const-tableau>|<const-enreg>
|<variable>|<expr-reg-chaine>|'null'|<expr-de-temps>

<expr-de-temps>::=<const-type-temps>| ('incltime'|'intime'|'overlap')

'('(<valeur-non-dérivée-primaire>|<valeur-de-temps>)','
(<valeur-non-dérivée-primaire>|<valeur-de-temps>) ')'

La syntaxe des règles non modifiées et non décrites ici se trouvent dans [VELE 84].

A3- SYNTAXE DES ENONCES DE MANIPULATION LAMBDA ASSOCIES AUX HISTORIQUES

<expression-de-sélection> ::= 'selest'['unique']<expression-de-
-valeur>'from'<trajet-de-désignation>[<clause-where>]

<expression-de-valeur> ::= (valeur-courante>|<valeur-historique>)
(','(<valeur-courante>|<valeur-historique>))*

<valeur-courante> ::= <valeur>

<valeur-historique> ::= <désignation-histor> 'of' <valeur>

<désignation-histor> ::= <désignation-temps-explicite>|
<désignation-temps-implicite>

<désignation-temps-explicite> ::= 'version' ('at'|'after'|'before'|'during')
<valeur-de-temps>

<désignation-temps-implicite> ::= ('first'|'last'|'all'|
<indixe>'..'<indixe>|<indixe>|'last-'<indixe>) 'version'

<indixe> ::= <entier-sans-signe>

B- DEFINITION DES EXEMPLES

```
type date : time > hour ;
    id-étud : integer ;
    argent : (1000..40000) ;
    t-adresse : record
       numéro : integer ;
       rue : string(15) ;
       ville : string(15) ;
       code-postal : integer;
    end;

  unité : document
      structure case nature of
           T : texte
           I : image
           G : graphique
           RE: référence
           .
           .
         end;
      end;

  paragraphe : document
      structure
         list (1,*) of unité
      end;

  description : document
      structure begin
                formation : paragraphe
                liste-de-diplome : texte
                commentaire : list(1,*)  of paragraphe
               end;
      end;
```

```
cours-liste : document
    structure
        list(1,10) of paragraphe
    end;

t-dossier : document
    structure
        list(1,*) of paragraphe
    end;

liste-lire : array(1,10) of string(10) ;

dossier-bourse : document
    structure
      begin
       titre-de-bourse : texte
       durée : réf = durée-bourse
       taux : réf = taux-bourse
       titulaire : texte = T0
       corps : paragraphe
      end;
      constant
       T0 := 'CROUS de Grenoble'
        parameter
         durée-bourse : duration
         taux-bourse : argent
      end;
  end;

type ECOLE : entity
      nom-école : (INPG, U1, U2, U3) ;
      adresse : t-adresse ;
      telephone : integer ;
      enseignement : description ;
  end;
```

```
PERSONNEL : entity
    nom : string(20) ;
    adresse : t-adresse ;
    catégorie : (enseignant, ingénieur, secrétaire, technicien);
    dossier : t-dossier;
 end;

ENSEIGNANT : specilization of PERSONNEL
            where catégorie = 'enseignant'
           poste : (professeur, maitre-ass, assistant);
  end;

COURS : entity
   code-UV : string(10) ;
   nom : string(20) ;
   mode-examin : string(10) ;
   volume : integer ;
   bibliographie : liste-lire ;
   end;

ETUDIANT : entity
   nom : string(20) ;
   adresse : t-adresse ;
   date-nassance : date ;
   dossier : dossier-bourse ;
 end;

INSCRIPTION : relationship between
          ETUDIANT : etudiant(1,1) and
          ECOLE : école-de-étud(1,*)
      date-inscription : date ;
      année-scolaire : time-interval ;
      titre : (Thèse, DEA, MIAG, Licence, DEUG) ;
      spécialité : string(10) ;
      cours-suivre : cours-liste ;
      fin-étude : date ;
  end;
```

```
ENSEIGNER : relationship between
        ENSEIGNANT : prof-de-cours(1,4) and
        COURS : cours(1,3)
    année-scolaire : time-interval ;
    jour : periodic ;
    heure : periodic ;
    sale : string(10) ;
  end;

AFFECTATION : relationship bettween
        PERSONNEL : employé(1,2) and
        ECOLE : établissement(1,*)
    poste : string(20) ;
    date-affec : date ;
    contrat : t-dossier;
  end;

SUIVIE : relationship between
      ETUDIANT : étudiant(1,10) and
      COURS : cours-à-suivre(1,100)
    année-scolaire : time-interval ;
    note : (0..20) ;
  end;

rapport : document
  structure
    begin
     page-couverture
        begin
         nom : string(20);
         lieu-travail : string(10);
         titre : texte ;
         matière : string(20);
        end;
```

résumé : *list*(1,*) *of* paragraphe ;
corps : *list*(1,*) *of*
chapitre : *list*(1,*) *of* paragraphe ;
conclusion : *list*(1,*) *of* paragraphe ;
end;
end;

RAPPORT-RECHERCHE : *entity*
code-biblioth : string(10);
contenu : rapport;
end;

RAPPORTS TIGRE

TIGRE 1

Présentation générale du projet TIGRE
Janvier 1983

TIGRE 2

The TIGRE data model
M. LOPEZ, J. PALAZZO-OLIVEIRA, F.VELEZ - Novembre 1983

TIGRE 3

Proposition de modèle pour la normalisation des documents
G. BOGO, H. RICHY, I. VATTON - Mars 1983

TIGRE 4

Recommandations pour l'ergonomie d'un poste de travail attaché à un système bureautique
I. FORESTIER - Mars 1983

TIGRE 5

La correspondance de schémas entre les modèles TIGRE et relationnel
J. PALAZZO-OLIVEIRA, F. VELEZ - Novembre 1983

TIGRE 6

Description des éléments du modèle de document
G. BOGO, H. RICHY, I. VATTON - Septembre 1983

TIGRE 7

Programmation en logique pour une base de données généralisées
N'GUYEN G.T., J. OLIVARES, P. WINNINGER - Novembre 1983

TIGRE 8

Machine base de données - Etat de l'art
M. BURNIER - Novembre 1983

TIGRE 9

Caractérisation des formulaires pour une base de données généralisées
C. COLLET - Novembre 1983

TIGRE 10

Accès concurrent aux documents
S. BALTAS - Janvier 1984

TIGRE 11

Définition formelle du langage LAMBDA
F. VELEZ - Mars 1984

TIGRE 12

Logic programming for a generalized data mangement system
M. ADIBA, N'GUYEN G.T. - Janvier 1984

TIGRE 13

Modèle et fonctionnalités pour un système intégrant outils BD et outils de conception
D. RIALHE - Janvier 1984

TIGRE 14

SAGE : Un Système d'Autorisation GEnéralisé
M. ADIBA, F. AZROU - Janvier 1984

TIGRE 15

Coopération de Prolog et d'un SGBD généralisé :
Principes et applications
N'GUYEN G.T., J. OLIVARES, P. WINNINGER - Avril 1984

TIGRE 16

Two dimensional editing
V. JOLOBOFF, V. QUINT - Juin 1984

TIGRE 17

Aspects logiciels de la communication homme-machine sur les postes de travail individuels
V. JOLOBOFF - Juin 1984

TIGRE 18

Information processing for CAD/VLSI on a generalized data management system
M. ADIBA, N'GUYEN G.T. - Juin 1984

TIGRE 19

Représentation de la sémantique et des métas-connaissances dans une Base de Données Généralisées
N'GUYEN G.T., J. OLIVARES - Juillet 1984

TIGRE 20

Description de l'éditeur TIGRE
Fonctionnalités, architecture, interfaces
H. RICHY, I. VATTON - Juillet 1984

TIGRE 21

Gestion des paramètres d'un document TIGRE
C. COLLET - Novembre 1984

TIGRE 22

Les formulaires électroniques dans TIGRE
C. COLLET - Janvier 1985

TIGRE 23

Notion de temps dans les bases de données généralisées
M. ADIBA, BUI Q.N., J. PALAZZO-OLIVEIRA - Décembre 1984

TIGRE 24

L'intégrité sémantique dans une base de données généralisée
N'GUYEN G.T., P. WINNINGER - Décembre 1984

TIGRE 25

Prolog et bases de données relationnelles
N'GUYEN G.T., P. WINNINGER - Décembre 1984

TIGRE 26

SYCSLOG : Système logique d'intégrité sémantique
N'GUYEN G.T., J. OLIVARES - Janvier 1985

TIGRE 27

GRIF : Un éditeur interactif de documents structurés
V. QUINT, I. VATTON - Mars 1985

TIGRE 28

Semantic data organization on a generalized data management system
N'GUYEN G.T., J. OLIVARES - MAI 1985

TIGRE 29

Gestion des histotisques pour les bases de données généralisées TIGRE
BUI Q. N. - Juin 1985

TIGRE 30

Un mecanisme transactionnel pour le SGBD TIGRE
E. PEDRAZA - Novembre 1985

TIGRE 31

La prise en compte de documents structurés dans un SGBD : Aspects modèle, langage et architecture du système
F. VELEZ - Janvier 1986

TIGRE 32

Semantics of CAD objects for Generalized Databases
N'GUYEN G.T., RIEU - Mars 1986

TIGRE 33

CABD : Un SGBD pour la CAO
FAUVET M.C. - Mars 1986

TIGRE 34

Etude préliminaire pour le stockage et l'exploitation des comptes rendus médicaux
E. MUNOZ - Avril 1986

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de

. Madame C. ROLLAND, Professeur
. Monsieur F. BODART, Professeur

Monsieur BUI QUANG NGOC

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, Spécialité "Informatique".

Fait à Grenoble, le 31 octobre 1986

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

P.O. le Vice-Président,