Centre de Recherche en Informatique de Nancy

TYP – R: RÉALISATION EN TYP D'UN SYSTÈME MULTIBASE RELATIONNEL

THÈSE

soutenue publiquement le **19 avril 1983**

À L'UNIVERSITÉ DE NANCY I

pour l'obtention du grade de

DOCTEUR DE 3ème CYCLE EN INFORMATIQUE

par

Najib TOUNSI

devant la Commission d'Examen

Président: ..... *Jean-Claude DERNIAME*
Examinateurs: ..... *Jean-Jacques CHABRIER*
Jacques DUCLOY
Marion CRÉHANGE
Witold LITWIN

Université de Nancy I U.E.R. Sciences Mathématiques

Centre de Recherche en Informatique de Nancy

TYP – R:
RÉALISATION EN TYP
D'UN SYSTÈME MULTIBASE RELATIONNEL

THÈSE

soutenue publiquement le **19 avril 1983**

À L'UNIVERSITÉ DE NANCY I

pour l'obtention du grade de
DOCTEUR DE 3ème CYCLE EN INFORMATIQUE
par
Najib TOUNSI
devant la Commission d'Examen

Président : ..... *Jean-Claude DERNIAME*
Examinateurs : ..... *Jean-Jacques CHABRIER*
Jacques DUCLOY
Marion CRÉHANGE
Witold LITWIN

J'exprime ma profonde gratitude à Monsieur Jean-Claude DERNIAME qui m'a accueilli dans son équipe, et je lui suis particulièrement reconnaissant d'avoir situé le contexte de ce travail. Je le remercie très vivement de l'honneur qu'il me fait en présidant ce Jury.

Monsieur Jean-Jacques CHABRIER a suivi avec attention la réalisation de ce travail, et je le remercie très chaleureusement pour les conseils, les encouragements, les critiques qu'il m'a adressés au fur et à mesure, et qui ont permis d'achever ce travail.

Madame Marion CREHANGE a bien voulu trouver le temps de s'intéresser à cette thèse, et je la remercie beaucoup de l'attention particulière qu'elle y a portée, et des discussions très enrichissantes qu'elle a bien voulu m'accorder. Ses remarques critiques et suggestions m'ont été très utiles pour compléter l'exposé.

Je remercie Monsieur Jacques DUCLOY d'avoir porté un jugement sur cette thèse, des remarques et des suggestions qu'il m'a adressées, et d'avoir accepter de siéger dans le Jury.

J'adresse mes sincères remerciements à Monsieur Witold LITWIN de trouver le temps de venir participer à ce Jury, et je lui suis très reconnaissant de s'intéresser à ce travail.

Enfin, j'exprime mes plus vifs remerciements à Madame Danielle MARCHAND pour la réalisation matérielle de cette thèse, tout le soin, la patience et la gentillesse qu'elle y a apporté.

Je remercie également tous mes camarades d'équipe, anciens et actuels.

Sommaire

Introduction

Les systèmes de bases de données sont des systèmes logiciels. Durant longtemps les recherches dans le domaine des bases de données se sont situées en marge de celles relatives aux autres activités informatiques, en particulier les langages de programmation (sujet clé pour la construction des logiciels), et de manière plus générale les systèmes intégrés de développements de systèmes : logiciels fiables, évolutifs, faciles à maintenir et de moindre coût ; car pour améliorer la production d'un logiciel il faut améliorer la productivité des programmeurs avec notamment les outils nécessaires. Or, les systèmes bases de données sont des systèmes logiciels, qui sont aussi chers à développer et à maintenir. Par ailleurs, il est reconnu actuellement que la qualité d'un produit, si elle est fonction du langage de programmation mis en œuvre, elle dépend essentiellement de la méthodologie de conception utilisée.

Aussi une conception modulaire, structurée par niveaux d'abstraction successifs est probablement la bonne approche, car elle permet de diminuer de la complexité d'un système et d'aborder sa construction de manière progressive. D'où les tendances récentes dans la définition de langage de programmation supportant les notions de types abstraits de données, qui rendent l'utilisation d'un objet indépendante de sa représentation, et de modularité qui permet de décomposer un grand programme en petits morceaux fonctionnels écrits séparèment.

Le projet TYP va dans ce sens, et plus particulièrement dans le but de permettre la conception et le développement de systèmes bases de données, et d'applications bases de données.

Il a abouti dans un premier temps à la réalisation d'un système de développement de programmes comprenant :

- un langage ATM supportant la modularité des programmes et permettant la manipulation de types abstraits de données. Il offre en plus des possibilités de généricité et d'utilisation d'itérateurs abstraits, trés utiles pour les applications bases de données.

- Un interface de conservation d'objets.

- Un connecteur qui construit des programmes exécutables à partir de morceaux compilés séparément.

TYP - R est un système de bases de données relationnelles et réparties, écrit en TYP. Dans l'approche suivie, on considère de telles bases comme des ensembles de bases, appelés multibases.

Cette approche définit un cadre formel pour une méthodologie de conception de systèmes multibases, et elle permet de construire de telles bases de manière progressive, à partir de plusieurs bases élémentaires.

TYP - R se distingue par le fait qu'il est le seul projet actuel qui se propose d'étudier et d'expérimenter les nouvelles méthodes et techniques en matière de génie logiciel pour développer des systèmes multibases. Un tel système doit être modulaire de façon à s'adapter progressivement à la construction d'une multibase. Par ailleurs, la notion de type abstrait de données permet de ne pas se préoccuper des contraintes physiques d'implantation.

C'est dans ce cadre que nous avons réalisé une maquette qui a été présentée en Novembre 1981 à Paris lors des journées de présentation du projet SIRIUS sur les bases de données réparties. Cette maquette a permis notamment de montrer les résultats importants suivants :

- Il est possible de constituer automatiquement une information à partir de données situées dans plusieurs bases, en ne formulant qu'une seule requête.

- Il est possible de concevoir un système multibase sans se préoccuper de contraintes pouvant résulter de la répartition des données sur plus d'un site.

Le premier résultat est très utile à l'utilisateur, et le second très important pour le concepteur et le réalisateur.

L'exposé de cette thèse est mené progressivement en commençant (chapitre I) par décrire ce qu'est une base de données, et le modèle relationnel de données, avant d'introduire (chapitre II) les bases de données réparties, et l'approche base ensemble de bases (ou multibase) suivie en TYP - R. Nous entamons ensuite une description (chapitre III) de cette notion de multibase.

Dans le chapitre suivant, nous décirvons brièvement le système TYP, en insistant sur les aspects orientés bases de données et leur intérêt, avant de commencer (chapitre V) la description de la maquette réalisée : aspects fonctionnels, choix de réalisations, architecture, programmation en TYP des différents composants et principe de fonctionnement.

Signalons enfin, que TYP - R entre dans le cadre du projet B A BA de W. LITWIN, INRIA.

CHAPITRE I

LES BASES DE DONNEES ET LE MODELE RELATIONNEL DE DONNEES

I - 1 LE CONCEPT BASE DE DONNEES

Le terme Base de Données est apparu pour la première fois au milieu des années 60 dans les systèmes d'informations militaires pour désigner "une collection de données que partagent les utilisateurs d'un ordinateur muni d'un système de temps partagé"[McG 81]. Il faut peut-être chercher là l'origine de la terminologie Base.

Ce terme est ensuite entré progressivement dans le domaine de l'informatique et des organisations pour désigner finalement "une collection de données mémorisées dans un ordinateur, et utilisable par les applications des différents services d'une entreprise"[Eng 72]. Il faut prendre comme signification du mot donnée, une valeur contenant une information significative pour l'entreprise.

Cette nouvelle définition de Base de Données entraîne qu'une même donnée peut être utilisée par plusieurs applications.

En effet, avant l'apparition des Bases de Données, beaucoup de problèmes se posaient en informatique dans les organisations. D'une part les applications au sein d'une entreprise avaient chacune ses fichiers. Chaque service donc possèdait ses propres fichiers qui étaient conçus et maintenus indépendamment des fichiers des autres services. D'autre part, ces applications étaient programmées à un niveau très bas, c'est-à-dire, que les programmes (écrits généralement en PL1 ou COBOL) étaient très dépendants des fichiers qu'ils traitaient donc à la structure physique des données enregistrées.

Par conséquent, une même donnée pouvait figurer dans différents fichiers entraînant une certaine redondance qui faisait que souvent les données ne concordaient pas entre-elles. Il fallait donc unifier les données et en assurer la qualité. C'est ce qu'on a appelé l'intégration des données des

différents fichiers au sein d'une même de Base de Données qui les centralise et permet aux applications des différents services de les partager. Cela a permis aussi de développer certaines applications qui utilisent les données des différents services. Ce qui ne pouvait se faire que par la mise en commun de toutes les données.

L'intégration des données réalise l'unification des différents fichiers dans le but d'éliminer les redondances d'information et par conséquent de diminuer le risque de données incompatibles entre-elles.

Par ailleurs, cette intégration permet un partage des mêmes informations entre différents utilisateurs pour des besoins différents. On réduit ainsi le coût de stockage de ces informations. Cependant ce partage d'informations à pour conséquence aussi de diminuer le degré de liberté d'un service vis à vis d'une donnée et nécessite donc un système de contrôle et de protection.

Voilà qui résoud, du moins partiellement, le problème de redondance et d'incompatibilité des informations. Car, en effet les programmes restent toujours dépendants des fichiers et rendent ces informations inaccessibles aux usagers non programmeur. Il s'avère alors important d'élever le niveau des langages utilisés pour manipuler les données. Il faut donc d'une part rendre la Base de Données accessible à des utilisateurs non informaticien, et d'autre part assurer l'évolution des programmes d'application et des données sans coût excessif.

Pour cela, il faut rendre les programmes indépendants des structures physiques des données et des méthodes d'accès à ces données en fournissant un Logiciel qui transforme des structures de données de bas niveau en structures de haut niveau.

Il convient en effet de distinguer entre la structure physique des données, c'est-à-dire comment les valeurs sont enregistrées et de quelle manière on y accède, et la structure Logique de ces données c'est-à-dire comment les utilisateurs les "perçoivent" dans les différents

applications, car ce n'est pas la même vue selon les besoins.

Enfin, pour rendre les données accessibles à des usagers non informaticiens, il faut fournir un interface pour un langage de requête de très haut niveau.

Les bases de données visent donc les objectifs suivants :
- l'intégration des données pour en assurer la qualité et le partage ;
- garantir une indépendance entre données et programmes en développant des langages de haut-niveau pour distinguer : le niveau physique du niveau logique des données ;
- assurer le contrôle de l'intégrité des données et leur protection.

D'où le schéma fig. 1 correspondant à l'architecture d'une Base de Donnée, proposée par le groupe d'étude ANSI/SPARC [ANS 73] crée en 1972 pour standardiser la technologie Base de Donnée.

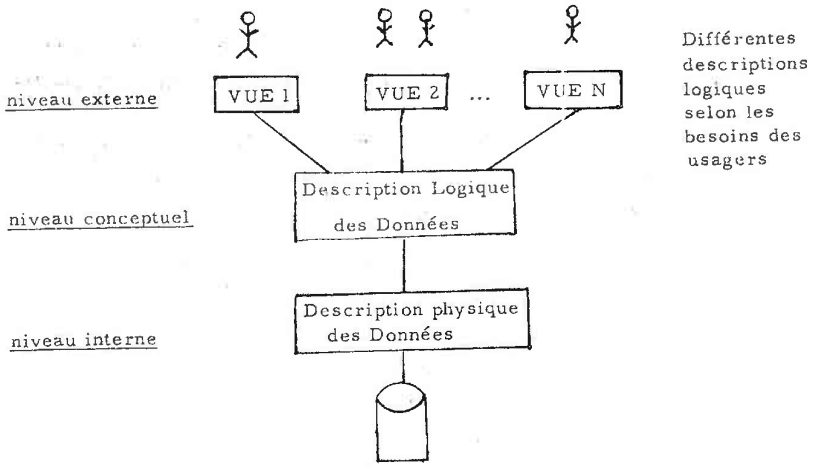


fig. 1 - Les différents niveaux dans la conception d'une B.D.

I - 2 LES SYSTEMES DE GESTION DE BASES DE DONNEES

Pour se concrétiser cette architecture nécessite le développem d'un logiciel qui assure la liaison entre les différents niveaux, et réali la correspondance ("Mapping") entre les descriptions logiques et physiq des données. Ceci pour transformer une requête usager exprimée sur des structures de données logiques en l'interprétant par une recherche sur des structures physiques.

Ce logiciel est appelé Système de Gestion de Base de Données, e doit donc permettre de gérer les données de la base, en assurer l'inté té et la protection, et fournir aux différents usagers une vue logique de données aussi variée que possible.

Pour répondre aux trois objectifs des Bases de Données, la réali tion d'un SGBD nécessite (d'après [McG 81]) :

- le développement de méthodes de structurations de données pou représenter l'intégration de toutes les données.
- Le développement de langages de haut niveau pour définir, et manipuler les données.
- Le développement de mécanismes de protection pour contrôler l'utilisation des données et en garantir la sécurité et l'intégrité.

Les SGBD sont actuellement classés selon leur méthode de struc ration de données ou Modèle de Données. Dans chacun des différents m le SGBD fournit généralement deux langages de haut niveau :

I.2.1 Le langage de Définition de Données

Il permet la description et la définition des objets de la base, et qui a trois niveaux de description. Dans le premier (niveau e terne) on décrit les données telles qu'elles sont vues par les différents usagers. Cette description peut ne concerner qu'une partie des donné

qui convient à une utilisation particulière. Une même description peut être partagée par plusieurs utilisateurs. Cela consiste en des déclarations des attributs et des types des objets en faisant abstraction de la manière dont ces objets sont décrits aux niveaux plus-bas. Cette description est appelée Schéma Externe (SE).

Dans le deuxième, dit niveau conceptuel, on décrit l'ensemble de toutes les informations contenues dans la base (Schéma Conceptuel SC.) tout en faisant encore abstraction de la description interne donnée dans le troisième niveau, appelé niveau interne, et qui elle consiste en la description des enregistrements physiques. On appelle Schèma Interne (SI) une telle description.

Entre ces différents niveaux il existe des correspondances qui lient les différentes descriptions d'un même objet.

Une première correspondance existe entre le SI et la SC, et qui définit comment les données perçues conceptuellement sont mémorisées. Si des changements interviennent sur les enregistrements physiques alors les correspondances, avec le SC doivent aussi changer pour préserver ce dernier, et garantir ainsi l'indépendance "données programmes". Ces changements sont l'affaire de l'administrateur de la base.

L'autre correspondance définit le lien entre un SE et le SC. Généralement le même type de différence peut exister entre ces deux schémas que celle existant entre le schéma interne et le conceptuel.

I.2.2 Le langage de manipulation de données :

Il donne à l'utilisateur la possibilité d'exprimer les accès ou les mises à jour sur les données de la base en utilisant la description qui en est faite dans le langage de définition.

Ces langages sont généralement de deux types qui fournissent d
possibilités :

1) on manipule les objets de la base à travers un langage procéd
classique (COBOL ou PL1 par exemple) dans lequel on a intercalé des
appels à des primitives qui permettent l'exécution de certaines opéra
sur la base (recherche, modification d'information...). De tels langa
sont destinés à des programmeurs d'application (cf. SYSTEMR avec
et SQL [CHA 80], ou INGRES avec C, un langage comme ALGOL,
QUEL [HEL 75]) ;

2) on manipule les objets de la base à travers un langage non-pr
cédural[1] et de très haut niveau qui permet d'exprimer des requêtes
recherche ou de modification, en termes très proches du domaine d'
application de la base des données.

L'utilisation de tels langages est généralement faite par des usa
non informaticien à travers un terminal conversationnel.

Nous reviendrons sur ce type de langages avec des exemples da
(II - 3).

II - LE MODELE RELATIONNEL

Ce modèle a été introduit par CODD E.F. [COD 70], et se ca
rise d'une part par sa simplicité, et d'autre part par une certaine syn
dans sa structure de données [CHA 76] qui en font que c'est le "mod
préféré" des SGBD. Car il réalise mieux l'indépendance entre donné
et programmes et offre ainsi la possibilité de définir des langages de
quêtes "naturels" comme SEQUEL par exemple [CHA 74].

(1) Si on convient d'appeler non-procédural un langage qui spécifie un
ment ce qui est désiré et non comment on l'obtient.

II.1 Définitions

Etant donné des ensembles $D_1, D_2, \ldots, D_n$, non nécessairement tous distincts, une relation R est un sous-ensemble du produit cartésien $D_1 \times D_2 \times \ldots \times D_n$.

Il ressort de cette définition qu'une relation est un ensemble de n-uplets $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ appelés tuples avec $\lambda_i \in D_i$ pour $i = 1 \ldots n$, et que tous les tuples sont différents.

Il est commode parfois de se représenter une relation comme une table dont les lignes représentent les tuples. Un exemple est donné par la figure II.1.a qui illustre une relation décrivant les employés d'une entreprise (on suppose qu'on les distingue par un nom).

Un employé est décrit par son nom, son salaire, le nom de son chef (qui peut être aussi un employé) et le numéro du département où il travaille.

On suppose qu'un employé a un seul salaire, un seul chef et travaille dans un seul département.

EMPLOYE

NOM	SALAIRE	CHEF	DEPT
DURAND	6 500 F	Pierre	1
MARTIN	7 000 F	Pierre	1
DUBOIS	8 000 F	Dupont	3

Figure II.1.a Relation EMPLOYE.

Dans cette représentation, les propriétés découlant de la définition d'une relation doivent être respectées :

1) Toutes les lignes sont différentes ;

2) l'ordre des lignes n'est pas significatif ;

3) l'ordre des colonnes est significatif, les tuples (Jacques, 4 00
Jean, 1) et (Jean, 4 000, Jacques, 1) ont des significations dif
rentes.

Le degré d'une relation est le nombre de colonnes de la table qu
la représente.

Dans la terminologie BD, on donne un nom à la table et un nom à
chaque colonne comme il est indiqué dans la figure II.1.a. Le nom de
table est appelé nom de la relation, et les noms des colonnes sont app
lés attributs. L'ensemble des valeurs qui définissent une colonne est a
pelé domaine. Par exemple, ici on a quatre attributs : NOM, SALAIRE
CHEF et DEPT, mais on n'a que trois domaines, des noms, des salai
et des numéros. Il est donc important de distinguer entre attribut et
domaine. Un attribut sert essentiellement à distinguer les colonnes (le
noms dans la lère colonne n'ont pas le même rôle que les noms de la
3ème colonne).

Les différentes valeurs qui composent un tuple sont appelées co
posants du tuple. Quand les composants dans une colonne (ou un ense
de colonnes) permettent à eux seuls de distinguer entre tous les tuples
la table, l'attribut (ou l'ensemble d'attributs) correspondant est appe
une clé candidate de la relation. NOM est une clé candidate de la relat
EMPLOYE. Quand il y a plusieurs clés candidates, on en choisit une q
on appelle clé primaire de la relation ou tout simplement la clé de la
relation.

On appelle clé secondaire, un attribut (ou ensemble d'attributs)
est clé primaire d'une autre relation. Considérons par exemple la rel
LOCALITE ci-après :

LOCALITE	DPT	ETAGE
	1	2
	3	4
	:	:

Fig. II.1.b Relation LOCALITE

qui indique où est situé chaque département. Si l'attribut DPT est clé primaire de LOCALITE, l'attribut DEPT de EMPLOYE est clé secondaire pour cette dernière relation.

Cette notion de clé secondaire est très utilisée dans l'expression de contraintes d'intégrité, appelées contraintes référentielles [DAT 81]. On peut remarquer aussi que l'attribut CHEF est clé secondaire pour la même relation EMPLOYE.

II.2 Normalisation des relations

Le concept de normalisation, introduit par CODD [COD 71], est basé sur la remarque que certaines relations se comportent mieux dans un environnement de mise à jour, que d'autres relations contenant pourtant les mêmes informations. Un phénomène, analogue à celui des effets de bord dans les langages de programmation, apparaît quant on met à jour un relation, et se caractèrise par une perte d'informations par exemple, ou par l'introduction de contradictions dans les données.

La théorie de normalisation sert donc à la conception de schémas de relations qui ont des propriétés favorables à la mise à jour. Nous allons montrer intuitivement sur quelques exemples certaines de ces propriétés.

II.2.1 Première forme normale (1 FN) :

Une relation est dite en 1FN si tous les composants de tous les tuples sont atomiques (c'est-à-dire indécomposables quant au

S.G.B.D.) ;

exemple : Relation fournisseur-pièces

R1

F	PS
F1	{P1, P2, P3}
F2	{P2, P4}

R2

F	P
F1	P1
F1	P2
F1	P3
F2	P2
F2	P4

La relation R1 n'est pas en 1FN car le composant {P1, P2, P3} n'est pas atomique, mais une liste qui peut être décomposable. D'où la relation R2 qui est en 1FN.

II.2.2 Deuxième et troisième Formes Normales (2FN, 3FN) :

Les relations en 1FN doivent être transformées en relations en 3FN à cause d'anomalies (pertes d'information etc...) dûes à leur mise à jour.

Considérons la relation en 1FN suivante :

E

NOM	SALAIRE	CHEF	DEPT	ETAGE
DURAND	6 500 F	PIERRE	1	2
MARTIN	4 500 F	DURANT	2	3
:	:	:	:	:

Figure II.2.a.

Qui pour chaque employé d'une entreprise donne son salairè, son chef, le département où il travaille et l'étage où est situé ce département, et suppose qu'un département est situé à un seul étage. Examinons quelques exemples "d'anomalies" :

- Insertion : supposons qu'on veuille insérer dans la base le fait qu'un département est situé à tel étage (Dept 3 est situé au 4ème étage par exemple) ; cela est difficile dans la relation E, puisque les autres composants, le nom, le salaire et le chef n'ont pas de valeur et par conséquent, on ne peut pas accèder à cette information puisque la clé NOM n'a pas de valeur ;

- Suppression : considérons le tuple (Martin, 4500, Durant, 2, 3) et supposons qu'on veuille supprimer ce tuple (l'employé Martin est parti en retraite). Si le fait que le département 2 est situé au 3ème étage ne figure que dans ce tuple, cette information est perdue en même temps. On supprime une information concernant un département par le fait de supprimer un employé ; ce qui peut avoir de sérieuses conséquences.

- Modification : soit maintenant à modifier le fait qu'un département est situé à tel étage. Si ce fait se trouve dans plusieurs tuples, cela nécessite une recherche exhaustive suivie de la modification. On court alors le risque de ne pas tout modifier et d'introduire des contradictions dans les données.

Ces quelques exemples, peut-être pas trop réalistes, montrent bien qu'une relation en 1FN n'a pas de propriétés suffisantes pour la consistance des données et la conception de schémas de BD.

L'un des objectifs de la théorie de normalisation des relations est donc d'éliminer ce genre "d'anomalies". Le résultat le plus important et le plus utilisé de cette théorie est la Troisème Forme Normale (3 FN). La deuxième n'étant qu'un intermédiaire entre la première et la troisième.

Pour essayer de comprendre comment cette 3FN évite les "an
malies" discutées ci-dessus, nous allons voir brièvement le
concept de Dépendance Fonctionnelle (DF) entre attributs d'une rela
On peut déjà remarquer que ces "anomalies" sont dues au fait que d
une même relation on mémorise deux ou plusieurs concepts apparen
indépendants : en l'occurrence ici, des informations concernant des
ployés (salaire, manager, département) et des informations concer
des départements (étage).

II.2.3.1 Dépendances fonctionnelles (DF)

On dit qu'un attribut Y (ou une collection d'attribu
d'une relation R est fonctionnellement dépendant d'un attribut X (ou
collection d'attributs) de R si, à chaque instant, chaque valeur de X
en relation avec une seule valeur de Y dans R. On note cette DF par
$X \longrightarrow Y$, et on appelle X un déterminant.

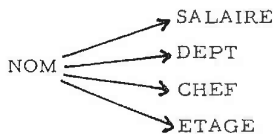
Exemple :

X	Y
x_1	y_1
x_1	y_2
x_2	y_2
x_2	y_1

X	Y
x_1	y_1
x_2	y_2
x_3	y_1

à gauche il n'y a pas une DF en X et Y (x_1 est associé à y_1 et y_2), à droite, on a $X \longrightarrow Y$.

Dans la relation E de la figure III.2.a, on a les DF suivantes :



DEPT ⟶ ETAGE

Les premières sont immédiates puisque, par définition de la clé, tous les attributs d'une relation sont fonctionnellement dépendants de la clé de la relation. NOM ⟶ ETAGE est aussi une conséquence de NOM ⟶ DEPT et de DEPT ⟶ ETAGE.

Par contre la DF DEPT ⟶ ETAGE exprime le fait qu'un département est localisé à un seul étage.

II.2.3.2 Définitions de la 3FN

La 3FN a été définie de plusieurs manières, nous en donnons ici deux définitions équivalentes :

1 - Définition de BOYCE - CODD [COD 74]

Une relation R est en 3FN si :

i) elle est en 1FN

et

ii) pour toute collection C d'attributs de R on a : si un attribut n'appartenant pas à C est fonctionnellement dépendant de C, alors tous les attributs de R sont fonctionnellement dépendants de C.

Dans notre exemple, la relation E n'est pas en 3FN car on a : DEPT ⟶ ETAGE, et on n'a pas, par exemple DEPT ⟶ SALAIRE.

2 - Définition de Sharman [SHA 75]

Une relation R est en 3FN si chaque déterminan
une clé. (Dans le sens clé candidate).

Dans l'exemple, DEPT est un déterminant (à cause de DEPT
DEPT ⟶ ETAGE), mais ce n'est pas une clé candidate.

Remarque : Soit X un ensemble d'attributs :

On dit que l'attribut A dépend totalement de X si A dép
fonctionnellement de X et ne dépend fonctionnellement d'aucu
partie de X. On dit alors, qu'une relation R est en 2FN SSI, e
est en 1FN, et si chaque attribut non clé est totalement dépen
de la clé. Non clé signifie n'est pas (ou ne fait pas partie d')
clé candidate.

Ces définitions de la 3FN expriment l'idée simple qu'une rela
doit décrire un seul "concept", c'est-à-dire un seul fait. Si elle en
décrit plusieurs, elle doit être décomposée en autant de relations c
cune décrivant indépendamment chaque fait. Nous avons déjà dit que
la relation E (figure II.2.a) décrit deux faits différents (reflète deux
formations indépendantes) qui sont : des données concernant des em
et des données concernant des départements.

Elle doit se décomposer en 2 relations, telles que celles-ci pa
exemple :

E' (NOM #, SALAIRE, MANAGER, DEPT)
D (DPT #, ETAGE).

Figure II.2.b relations employés et départe
elles sont en 3FN.

(1) Le symbole # indique la clé.

II.2.3 Autres Formes Normales :

Dans l'exemple précédent, nous avons décomposé la relation E, en deux relations E' et D. Mais, on aurait pu décomposer E en deux autres relations :

E1 (NOM #, SALAIRE, MANAGER, DEPT)
E2 (NOM #, ETAGE)

qui sont en 3FN. Mais, cette décomposition est moins satisfaisante que la précédente, car il est toujours aussi difficile d'insérer l'information que le département 3 est situé au 4ème étage, à moins qu'un employé ne travaille pas dans ce département.

Dans le processus de Normalisation, la théorie des DF a engendré la théorie de décomposition de relations. R. FAGIN [FAG 77] a introduit un autre type de dépendance appelé Dépendance Multivaluée qui lui, a permis de définir une 4ème Forme Normale et d'élargir ainsi le domaine de la conception de schémas de BD relationnelles. Ces dépendances multivaluées sont une généralisation des DF et fournissent un outil puissant qui permet d'étudier la décomposition de schémas de relations en une famille de schémas, sans aucune perte d'information. (Cf. aussi [DAT 81-a]).

II - 3 LES LANGAGES RELATIONNELS USUELS

L'un des intérêts majeurs du modèle relationnel est la complète séparation entre le niveau logique, et le niveau physique de description de données. Du point de vue langage, cette distinction se caractèrise de deux façons :

- par la possibilité de définir des "commandes" permettant de manipuler plusieurs enregistrements à la fois (opération ensembliste) évitant à l'utilisateur de penser en terme d'itérations sur des enregistrements interconnectés, et à travers un chemin d'accès prédéfini.

- Par la possiblité d'extraire de la base toutes les informations significatives par combinaisons de ces commandes.

II.3.1 L'Algèbre Relationnelle

Dans [COD 70] CODD a défini des opérateurs qui manipulent les relations en tant qu'opérandes. L'opérateur SELECT (appelé aussi RESTRICT), prend comme argument une relation et produit, une nouvelle relation obtenue en selectionnant certains tuples de la première. L'opérateur PROJECT transforme aussi une relation donnée en une nouvelle relation obtenue, en selectionnant certains attributs de la première. L'opérateur JOIN prend deux relations comme opérandes, et produit une troisième relation obtenue en "concatenant" des tuples de la première avec des tuples de la seconde, à chaque fois que les valeurs d'attributs spécifiés dans la première, correspondent à des valeurs d'attributs spécifiés dans la seconde.

L'algèbre relationnelle qui contient ces opérateurs avec ceux d'union, d'intersection ou de différence, n'est en fait pas un langage standard, mais est destinée à servir de base pour la définition de langages de manipulation relationnels de haut niveau.

En effet, toutes les informations qu'on peut extraire d'un ensemble de relations, peuvent s'exprimer par combinaison de ces opérateurs. Un langage relationnel de haut niveau est dit complet (au sens de recherche de toute information) s'il a au moins la puissance d'expression de JOIN, SELECT, PROJECT, UNION, DIFFERENCE et CART (produit cartésien).

Remarque : JOIN peut s'exprimer comme un produit cartésien suivi d'une sélection, et d'une projection (cf. exple 3 ci-dessous).

Exemples : Soit les relations dont le schéma est :

EMP (NOM #, SAL, CHEF, DEPT)
LOC (DPT #, ETAGE)

Figure II.3

1 - PROJECT (EMP, NOM, SAL) est la relation (NOM, SAL) obtenue en ne retenant de la relation EMP que les colonnes correspondant aux attributs NOM et SAL. On a ainsi, une relation qui donne le nom et le salaire de tous les employés.

2 - SELECT (EMP, SAL > 10 000 F) est la relation (NOM #, SAL, CHEF, DEPT) obtenue en ne retenant de la relation EMP que les employés dont le salaire est supérieur à 10 000 F.

3 - JOIN (EMP, LOC, DEPT = DPT) est la relation obtenue en concatenant les tuples de EMP et de LOC dont les valeurs correspondant aux attributs DEPT de l'une, et DPT de l'autre sont égales.
NB : dans le cas de l'égalité (DEPT = DPT) l'opérateur est dit EQUI - JOIN ou NATURAL JOIN.

Ce JOIN peut être obtenu par :

PROJECT (SELECT (CART (EMP, LOC), DEPT = DPT),
NOM, SAL, MGR, DEPT, ETAGE).

4 - Exemples de requêtes :

- Quel est le salaire du chef de l'employé JEAN :
  PROJECT (JOIN (EMP, SELECT (EMP, NOM = 'JEAN'),
  NOM = CHEF), SAL) ;
- Quels sont les noms de tous les employés et l'étage où ils travaillent :

PROJECT (JOIN (EMP, LOC), DEPT = DPT), NOM, ETAGE).

Le calcul relationnel : Avec l'introduction du modèle relationn CODD a défini deux types de langage : l'algèbre relationnel [COD 70 et le calcul relationnel [COD 71a].

Le calcul relationnel est une application de calcul de prédicat, permet à l'utilisateur de définir des variables pour désigner des tuple et de spécifier un prédicat sur ces variables, en vue d'obtenir les tup répondant à une requête donnée.

Un langage est, selon CODD, relationnellement complet s'il a u puissance d'expression équivalente à celle du calcul relationnel [CO 72]. L'algèbre relationnel est un tel langage [COD 72].

On peut donc se référer à l'un ou l'autre des deux langages pour les comparer à d'autres.

II.3.2 SEQUEL (Structured English QUEry Langage) [CHA [AST 75] [CHA 76a]

C'est le langage d'interface utilisateur de SYSTEMR [AST 76 - 79 - 81].

SEQUEL est un langage non procédural de trés haut niveau basé SQUARE (Specifying QUeries As Relational Expressions [BOY 73], langage de spécification de requêtes. Il est fondé sur une notion de "correspondance" ("Mapping") qui exprime la manière dont une perso utilise une table, si on assimile une relation à une table, un attribut à une colonne, et un tuple à une ligne de la table.

Pour chercher les noms des employés qui gagnent plus de 10 000 F, on regarde dans la colonne SAL les valeurs supérieures à 10 000 F, et on retient dans la colonne NOM, la valeur correspondante. En algèbre relationnelle, cette idée exprime une sélection suivie d'une projection.

Exemple : PROJECT (SELECT (EMP, SAL > 10 000), NOM)
ce qui est naturel car généralement le résultat d'une requête est une portion de relation de base ou virtuelle (résultat d'un opérateur).

Le "mapping" consiste donc en un nom de table (EMP), un nom de colonne de départ (SAL), un nom de colonne d'arrivée (NOM), et un argument (> 10 000). Le résultat est un ensemble de valeurs dans la colonne d'arrivée, correspondant aux valeurs de colonne de départ qui satisfont un argument donnée.

Cette idée généralisée permet d'exprimer des requêtes pour extraire la plupart des informations significatives d'un ensemble de relations considérées comme des tables. Le résultat produit étant lui-même une table.

SQUARE ayant une syntaxe plus mathématique qu'orientée utilisateurs, SEQUEL a été défini avec une syntaxe proche de l'expression parlée par l'utilisation de mots clés de la langue anglaise.

II.3.2.1 Exemples

Considérons les relations EMP et LOC (figure II.3).

R1 - Lister les noms des employés dont le salaire est supérieur à 10 000 F.

```
SELECT    NOM
FROM      EMP
WHERE     SAL > 10 000
```

Cet exemple de requête la plus simple consiste en trois mots clés: SELECT, FROM et WHERE et en trois paramètres : la relation (EMP), la colonne des valeurs résultats (NOM) et la condi à satisfaire (SAL > 10 000).

En algèbre relationnel cette requête s'écrit :
PROJECT (SELECT (EMP, SAL > 10 000), NOM).

R2 - Lister les noms de tous les employés avec le département l'étage où ils sont situés.

```
SELECT    NOM, DEPT, ETAGE
FROM      EMP, LOC
WHERE     EMP.DEPT = LOC . DPT
```

en algèbre relationnelle :

```
PROJECT (JOIN (EMP, LOC, DEPT = DPT),
          NOM, DEPT, ETAGE).
```

Remarque : En SEQUEL il y a plusieurs manières d'exprimer une requête, et en particulier le JOIN. Par exemple, pour savoir les noms des employés travaillant au 2ème étage :

```
SELECT  NOM
FROM    EMP
WHERE   DEPT = SELECT   DPT
               FROM     LOC
               WHERE  ETAGE = 2
```

ou

```
SELECT  NOM
FROM    EMP, LOC
WHERE   EMP.DEPT = LOC.DEPT
        AND LOC.ETAGE = 2
```

Ceci est possible en particulier, quand le résultat est une projection une relation existante.

Certaines requêtes demandent des informations qui ne peuvent être obtenues par une simple sélection de valeurs dans une table, mais par un calcul. SEQUEL offre la possibilité d'utiliser des fonctions prédéfinies (AVR, SUM, MAX, MIN, COUNT ...), qui calculent respectivement la moyenne, la somme, le maximum, le minima et le nombre d'occurrences d'un ensemble de valeurs. En plus de ces fonctions, l'utilisateur peut définir arbitrairement des variables appelées variables calculées et les utiliser dans une requête.

R3 - Pour chaque département lister le numéro du département et la moyenne de salaires des employés qui y travaillent :

```
SELECT    DPT,      Q
FROM      LOC
WHERE     Q = SELECT   AVR (SAL)
              FROM     EMP
              WHERE    DEPT = LOC.DPT.
```

La variable calculée Q et la fonction prédéfinie AVR dans cet exemple permettent d'exprimer des calculs pour obtenir des informations non stockées dans la base.

En utilisant la clause GROUP BY du langage SEQUEL, on peut écrire cette requête beaucoup plus simplement :

```
SELECT    DEPT  ,  AVR (SAL)
  FROM    EMP
GROUP     BY      DEPT
```

qui s'énonce en paraphrasant : "dans la relation EMP, calculer la moyenne des salaires des employés groupés par département, et lister le numéro de ce département avec cette moyenne".

SEQUEL possède ainsi une grande puissance d'expression, et de calcul avec une certaine souplesse d'utilisation.

II.3.2.2 Comparaison avec les opérateurs

JOIN, PROJECT et SELECT de l'algèbre relationnelle.

N.B. : L'opérateur SELECT de l'algèbre relationnelle et le mot clé SELECT de SEQUEL n'ont aucun rapport.

Soit deux relations dont le schéma est :

R (RW, RX, RY) et S (SY, SZ)

ayant les attributs RY et SY définis sur un même domaine.

Algèbre Relationnelle	SEQUEL
- PROJECT (R, RX, RY)	- SELECT RX, RY FROM R
- SELECT (R, RX = 'x')	- SELECT * FROM R WHERE RX = 'x'
	Pour sélectionner toute une ligne, on remplace les attr par une *
- JOIN (R, S, RY = SY)	- SELECT * FROM R, S WHERE RY = SY

Dans les deux cas le résultat est une nouvelle relation (ou table

II.3.2.3 Conclusion

SEQUEL est un langage simple, non proc ral qui n'utilise aucun concept mathématique, et qui est fondé sur une

étude théorique menée dans SQUARE un langage de spécification de requêtes. Son format structuré en blocs définis par des mots clés du langage naturel rendent sa compréhension, et son apprentissage très faciles [REY 75] . Ainsi SEQUEL est destiné à une large communauté d'utilisateurs pour un usage en mode itéractif de recherche d'information dans une base de donnée relationnelle.

II - 4 CONTRAINTES D'INTEGRITE ET VUES

Dans une base de données relationnelle, une relation est définie par deux choses : son intension et son extension. L'intension d'une relation (ou signification) c'est la déclaration de son schéma et des contraintes d'intégrité qui y sont attachées. L'extension d'une relation c'est l'ensemble des tuples qui la composent à un moment donné.

Les schémas sont déclarés en donnant le nom des relations, des attributs, et des domaines. Les contraintes d'intégrité par contre comportent plusieurs indications : les attributs composant la clé, les dépendances fonctionnelles entre attributs d'une relation, et les propriétés que doivent vérifier les données pour en garantir la cohérence et l'intégrité.

Les contraintes attachées à la définition de la clé sont de deux sortes : une clé primaire doit toujours être initialisée, (intégrité d'entité). Une clé secondaire peut ne pas être initialisée, mais si elle reçoit une valeur, cette dernière doit exister comme valeur de la clé primaire d'une relation dans la base (intégrité référentielle).

Les contraintes attachées à la définition des dépendances fonctionnelles servent surtout à la définition de relations en 3FN.

Les autres contraintes sont toutes celles qui imposent des contrôles lors des mises à jour comme par exemple : "les employés du département

5 ont un salaire > 10 000 F"
"les nouveaux salaires sont toujours supérieurs aux anciens".

Si l'intension d'une relation ne varie généralement pas au cours du temps, son extension par contre est soumise à toutes les modifications, et permet de créer des vues.

Une vue est un ensemble de relations dites virtuelles, c'est-à-dire non physiquement stockées dans la base, mais définies comme dérivées de l'ensemble des relations de la base. Chacune de ces relations virtuelles est obtenues par une requête d'interrogation.

Une vue est une "fenêtre dynamique" sur la base. Elle offre la possibilité à des différents utilisateurs d'avoir d'autres visions de la même base. On peut donc considérer qu'un schéma externe est une vue

Pour la mise à jour des vues, il faut propager la modification jusqu'au niveau des relations de base dont elles sont dérivées, et qui elles sont partagées par plusieurs utilisateurs. Ce qui est parfois dangereux et même impossible [COD 74] .

Des exemples de tels problèmes avec leurs conséquences sur les restrictions d'utilisation des vues sont décrits en détail dans [DAT 81 et les différentes solutions de plusieurs systèmes réalisés dans [KIM

Les vues sont appelées copies (ou instantanés) si elles sont créée et physiquement stockées dans la base. Elles reflètent les données comme elles étaient au moment de la copie.

Cette copie peut donc être mise à jour sans risque pour les relations de base. Par contre, elle doit être recréée périodiquement, su tout pour les applications temps-réel, pour refléter les dernière modifications intervenues dans la base [ADI 81] . Mais, il se pose alors le problème de tenir compte des modifications propres à la copie.

II - 5 CONCLUSION

Les avantages de l'approche Modèle Relationnel de données dans la conception, et la réalisation des Bases de Données ont été largement et éloquemment détaillés dans la littérature [COD 74 - 79 - 81] , [DAT 81 d] , [DEL 80] , [CHA 76] principalement. Citons simplement les principaux arguments qui sont développés : (1) l'indépendance entre données et programmes, (2) la simplicité de la structure de donnée qui permet de définir (3) des langages de manipulation de très haut niveau utilisables aussi bien par des programmeurs que par des non informaticien.

En effet, le Modèle Relationnel de données permet aux programmes d'application et aux formulations de requêtes d'interrogation d'être complétement indépendants des structures physiques de stockage des données, et leur accès. Il est possible de changer ces structures et les stratégies d'accès, sans recompilation des programmes, ou changement dans l'interface usager. Ceci est dû principalement à la structure de donnée, au niveau logique qui présente les informations comme des valeurs dans des tables, et adressables par le contenu. Ce qui implique qu'aucune méthode d'accès n'est à préconiser ou à spécifier à ce niveau. Toutes les informations significatives (réponse à une interrogation par exemple) peuvent être déduites de ces tables de la même manière, et par un ensemble de commandes, ou d'opérations (SELECT, RETRIEVE, ...) à travers certains langages.

Ces langages, SEQUEL [CHA 76] , QUEL [HEL 75] , QBE [ZLO 75] ..., sont des langages de très haut niveau, non procéduraux, et qui ont en plus l'avantage d'être accéssibles aussi à des non informaticien.

Dans [COD 82] CODD reprend en détail tous ces arguments, et définit ce que doit être une Base de Données Relationnelle. Il la présente comme étant le meilleur fondement, par le biais du Traitement Relationnel, pour l'amélioration de la productivité des programmeurs et des usagers des grands systèmes de partage de données, et pour la réduction des efforts consacrés à la maintenance de ces systèmes.

Parmi les futurs développements, CODD cite le "rapprochement" entre les langages de manipulation de Bases de Données, et les langages de programmation, et plus particulièrement, l'introduction de la notion de type abstrait de données dans les langages de Bases de Données, et les facilités de traitement de ces derniers dans les langages de programmation.

CHAPITRE II

BASES DE DONNEES REPARTIES

I - INTRODUCTION

On a commencé à parler de la répartition des données et des traitements, dès qu'il a été possible de travailler sur des ordinateurs distants et interconnectés pour échanger des informations entre programmes par des moyens tels que lignes téléphoniques, réseaux locaux ou nationaux, . . .

Des diverses interrogations suivantes [LEB 81] :

- Ne serait-il pas souhaitable de conserver à l'information sa répartition naturelle ? Peut-on adapter le système d'information à l'organisation et non l'inverse ? (raison sociologique).

- Comment pallier à la vulnérabilité des centres de traitements informatiques, où toute l'information est concentrée ? (problème de sécurité).

- Existe-t-il un moyen d'alléger la charge d'un ordinateur de la capacité de stocker, et de traiter de grands volumes de données (problème de performance) ?

- Peut-on exploiter les liens qui peuvent exister entre des informations dispersées ? ou rapprocher ces informations ?

Est née l'idée d'associer les bases de données et les réseaux d'ordinateurs (figure II.1).

Des recherches sont nées ensuite, visant à expérimenter l'implantation d'une base de données sur plusieurs ordinateurs, (on parle aussi de site), et à en étudier les problèmes.

Les plus connues sont celles du projet pilote SIRIUS de l'INRIA en France [LEB 80a], [SIR 81].

D'autres recherches ont été menées aux USA : R*, INGRES réparti, SSD-1 [VLD 80].

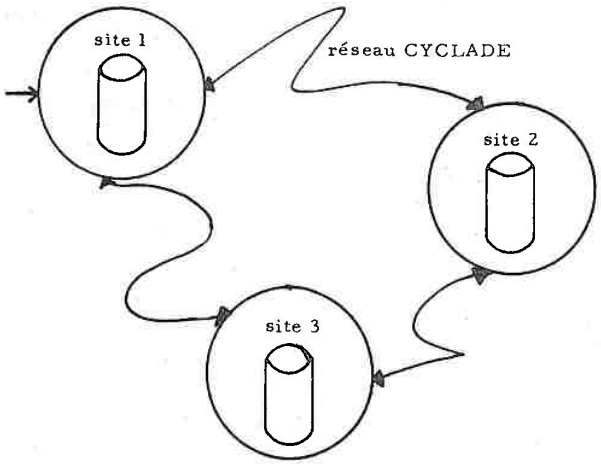


Figure II.1 (source [POL 79]).

II - PREMIERE APROCHE

II.1 Qu'est-ce qu'une BDR :

D'après [DDB 80], [VLD 80], [SIR 81] une base de données répartie (BDR) apparaît comme une base de données "classique" a néanmoins la particularité d'être constituée d'un ensemble de données mémorisées sur plusieurs ordinateurs interconnectés par un réseau. Un sous-ensemble mémorisé sur un seul ordinateur peut constituer lui-même une base de données (appelée locale par rapport à la base répartie) sur cet ordinateur.

Exemple : - base de données répartie sur des ordinateurs IRIS 80 connectés à travers le réseau national CYCLADE comme dans les projets POLYPHEME [POL 79] et ETOILE [MEY 78].

- base de données répartie sur un réseau local d'ordinateurs REALITE 2000 comme dans le projet SIRIUS-DELTA. [LEB 80]

II.2 Qu'est-ce qu'un SGBDR ?

Un SGBDR (Système de Gestion de Bases de Données Réparties) est un système qui doit :

1) Permettre de définir et de manipuler des ensembles de données réparties sur différents sites comme un seul ensemble logique ayant toutes les caractéristiques d'une base de données (par définition d'une BDR). Cela veut dire que les utilisateurs manipulent celle-ci, sans avoir à connaître la localisation des données.

2) Assurer l'intégrité des données et posséder des dispositifs de protections d'informations confidentielles et de reprise en cas de panne.

3) Offrir la possibilité de modifier la répartition des données sans remettre en cause les programmes utilisant la BDR.

Exemples :

- Le SGBDR du projet POLYPHEME [POL 79] est un système qui gère une BDR construite à partir de plusieurs bases existantes et gérées par des SGBD. Aucune supposition n'est faite sur ces SGBD notamment sur le modèle de données utilisé. Cette méthode de construction est dite ascendante et correspond pour POLYPHEME à une coopération de bases de données hétérogènes, réparties dans un réseau général d'ordinateurs. Toutes les opérations concernant la BDR passent par un site particulier où est défini le SGBDR.

- Dans le projet SIRIUS DELTA [LEB 80], le SGBDR est complétement réparti , dans le sens où il n'est pas nécessaire de définir un site "Maître". Le BDR est accessible de n' importe quel site du réseau, et tout se passe comme si l'utilisateur a, à sa disposition toutes les données du réseau sur ce site.

- Dans le projet ETOILE [MEY 78] le SGBDR gère une base de données répartie sur plusieurs sites, pour une organisatio étoilée : site central, sites régionaux. Ces derniers contiennent des données du site central, dont les utilisateurs peuvent avoir besoin sur ces sites. Ils sont "indépendants" entre-eux et ne communiquent qu' avec le site central qui joue un rôle de coordinateur.

II.3 Description d'une BDR

Par définition, une BDR se présente à ses utilisateurs comme une base de données puisque le système qui la gère (le SGBDR) a justement pour rôle de donner une vue intégrée de plusieurs ensembl

d'informations stockées sur différents ordinateurs. Il faut donc constituer une description globale de la BDR, et qui sera utilisée par le SGBDR.

Cette description globale sera scindée par niveaux de fonctionnalités, afin de constituer les schémas de la base qui seront dans une forme exploitable par le SGBDR [LEB 81] :

- schéma global : ce schéma est de même nature que le schéma conceptuel d'une base de données non répartie (cf. I.I.2). Il a pour rôle de représenter l'ensemble des informations constituant la BDR sans référence aux structures de mémorisation de celles-ci, et à leur localisation sur les sites de répartition.

- schémas externes (globaux) qui sont pour chaque application de la BDR, des vues adaptées du schéma global.

On voit donc que les 2 niveaux, externe et conceptuel proposés par l'architecture ANSI-SPARC sont présents dans une BDR. Ce n'est pas le cas pour le niveau interne. La base de données est répartie sur plusieurs ordinateurs, chacun contenant un sous-ensemble de celle-ci, et qui constitue une base locale. De ce fait, cette base locale sera aussi définie par des schémas locaux (externes, conceptuels, internes) conformément à l'architecture ANSI-SPARC.

Ce niveau interne pour la BDR décrit notamment :
- la répartition des données du schéma global sur les bases locales ;
- les règles de transformation données globales / données locales.

Cette description est appelée schéma interne global.

Un SGBDR est considéré comme un "utilisateur" pour une base locale. Il utilise donc un schèma externe défini sur celle-ci.

D'où la figure :

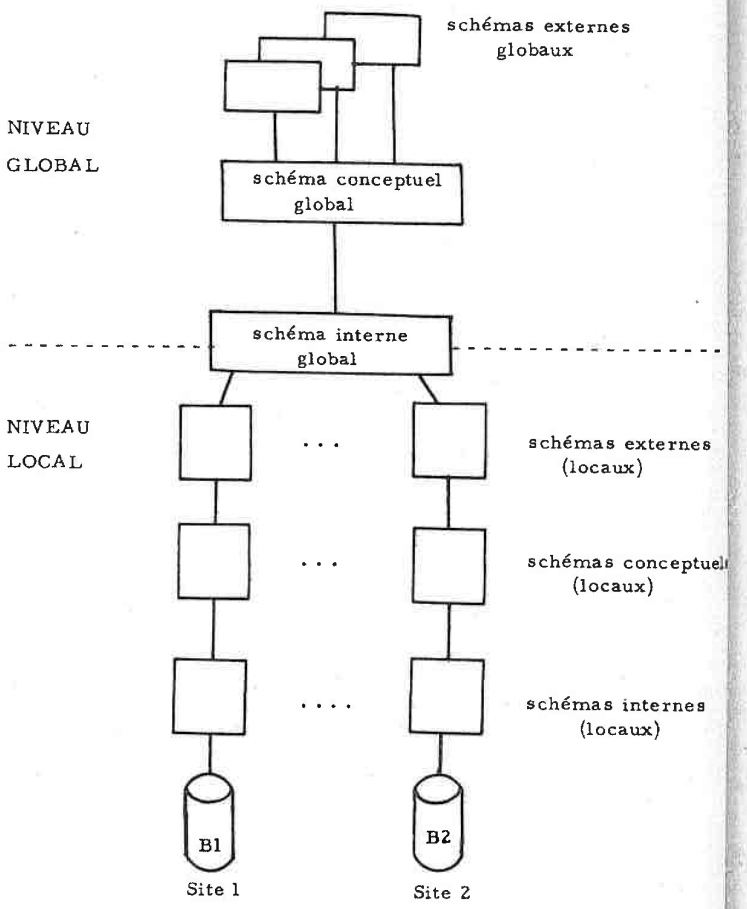


Figure II.2 - Architecture fonctionnelle d'une BDR
(source [LEB 81])

II.4 Restrictions de l'approche :

Le fondement de cette approche est l'hypothèse suivante : une BDR est UNE base de données qui est physiquement implantée sur plusieurs sites.

Des difficultés ont été mises en évidence par les maquettes développées dans le cadre du projet SIRIUS, notamment en ce qui concerne la conception et l'implantation, l'utilisation et l'évolutivité.

II.4.1 Problèmes de conception et d'implantation

Il a été reconnu que la définition d'un schéma conceptuel (global) de la BDR est en général compliquée et parfois impossible [LIT 81], car il se pose le problème de correspondance entre la description globale et les différentes descriptions locales des données.

Exemple : en termes relationnels.

Deux relations peuvent avoir le même nom, et des significations différentes dans deux bases différentes. Il faut, alors les distinguer dans la description globale. Inversement, deux relations peuvent avoir la même signification, et être décrites différemment dans deux bases différentes. Il faut alors, au niveau global en donner une seule description, sinon savoir reconnaître, qu'il s'agit de deux descriptions différentes d'une même entité. Ce qui aboutirait à un schéma conceptuel de peu d'intérêt.

Par ailleurs, l'architecture généralement admise pour les bases de données est celle en 3 niveaux d'ANSI-SPARC qui présente toutes les garanties en ce qui concerne l'indépendance entre données et

programmes. Or, toutes les architectures proposées pour les BDR différent de celle-ci en raison de la présence de données logiques,(1) en niveau "interne" de la BDR, niveau censé être du domaine de l'implantation [LIT 81] (cf. figure II.2). Plus particulièrement, le concepteur d'une BDR est confronté d'une part au problème de définition de "règles de répartition" des informations sur les bases locales où elles sont décrites, et stockées, et d'autre part au problème de définition de "règles de localisation" qui déterminent pour le SGBDR comment localiser les données qui sont réparties.

Exemple : les informations concernant les empoyés sont stockés dans la base A et celles concernant les départements où ils travaillent sont stockés dans la base B, ou bien les employés d'un âge > 50 sont stockées dans la base A et les autres dans la base B, ou bien encore, les informations concernant un employé sont stockées dans la base qui contient les informations concernant le département où il travaille etc...

Il faut donc pouvoir définir pour le SGBDR des algorithmes très complexes et très variés, en fonction du type de répartition choisi (partitionnement, duplication des données et aussi des structures), et des règles de localisation définies. Compléxités qui résultent de la prise en compte des volumes de données à échanger entre les sites, de différentes possibilités d'exécutions parallèles, ou des évaluations des temps de réponse, etc... (cf. par exemple [NGU 78], [ESC 81]

(1) Celles décrites localement.

II.4.2 Problèmes d'utilisation

Il est de bon sens de penser que pour l'utilisateur d'une base locale, les services offerts par un SGBDR devraient être au moins identiques à ceux offerts par "son SGBD habituel" ; Ce n'est pas toujours le cas. Par exemple :

- le langage de manipulation de la BDR peut ne pas être le même que celui de l'une des bases la constituant ;
- la BDR peut ne fournir que des vues partielles sur certaines bases locales, si l'administrateur de l'une de celles-ci désire cacher des informations confidentielles aux usagers de la BDR.

D'un autre côté, un problème important risque de mettre en conflit un SGBD local, et le SGBDR. En effet, au niveau de la BDR, on peut avoir des contraintes d'intégrité qui mettent en jeu des données appartenant à plusieurs bases. Ces contraintes sont celles que le SGBDR doit maintenir. Par ailleurs, d'autres contraintes peuvent exister localement, qu'un SGBD local doit maintenir. Il arrive alors, des situations où les deux types de contraintes sont incompatibles [DAR 79] , c'est-à-dire que l'une ou bien l'autre est violée.

II.4.3 Problèmes d'évolutivité

Les problèmes d'évolutivité des BDR sont en fait liés aux problèmes de conception et d'implantation. Il est très difficile de rajouter une nouvelle base sans une remise en cause totale du schéma global, et donc des programmes existants.

Il ressort de cette discussion que des problèmes certains se posent pour la conception et la réalisation d'une BDR, fiable et évolutive.

Ces problèmes sont dûs à notre avis à ce que l'approche prise
dans la plupart des cas, assez pragmatique étant donné qu'elle n'est
basée sur des définitions précises, ou des hypothèses stables, et not
ment une clarification de la notion de répartition [LIT 79], qui per-
mette aux concepteurs des BDR de ne pas se préoccuper des contrai
tes liées à l'implantation (plusieurs sites au lieu d'un, implications
sur la nature des partitionnements des données, et sur les taux de tr
ferts de celles-ci entre les sites, etc...).

Il est alors difficile de bien cerner les problèmes réellement n
veaux (définition et manipulation de plusieurs bases) qui se posent, d'
étudier les méthodes et développer les outils, et les techniques perm
tant d'élaborer les solutions efficaces.

III - APPROCHE BASE-ENSEMBLE-DE-BASES

L'idée de cette approche est qu'une base de données peut être co
tituée d'un ensemble de bases de données. Celles-ci sont des bases
usuelles simples, ou des ensembles de bases.

III.1 Définition [LIT 81]

Une base de données est répartie si elle est constituée
un ensemble de bases ; on parle alors de multibase de données. Le
schéma conceptuel de cette multibase peut être décrit par l'ensemble
des schémas conceptuels des bases qui la composent.

Exemple 1 : Une base de données sur des restaurants, et une b
de données sur des cinémas, peuvent former une base de données ré
tie de loisirs (figure II.3), si on considère que l'univers des loisirs e
constitué de celui des restaurants, et de celui des cinémas.


```
Multibase     LOISIRS
    base      RESTAURANTS
              :
    base      CINEMAS
              :
fin
```

Figure II.3 - Schéma d'une base de données répartie.

Exemple 2 : On peut adjoindre une base THEATRES à la BDR des loisirs, et constituer une autre BDR à partir de cette dernière, et d'une base METROS sur les transports :

a)
```
Multibase     PARIS-GUIDE
    Multibase     LOISIRS
        base      RESTAURANTS   } sous-univers
        base      CINEMAS       }   des loisirs
        base      THEATRES      }
    fin                         }
    base      METROS            } sous- univers des
                                }       transports
fin                             } Univers du Tourisme
```

Figure II.4.a

ou encore avec une base MUSEES :

b)
```
Multibase     TOURISME
    Multibase     LOISIRS
        base      RESTAURANTS
        base      CINEMAS
        base      THEATRES
    fin
    base      METROS
    base      MUSEES
fin
```

Figure II.4.b.

Dans cette approche, la répartition pour une base est considérée comme une propriété logique pour celle-ci. En d'autres termes, le fait d'être un ensemble de bases, constitue une nouvelle propriété pour une base de données, et qui en fait une BDR. Par conséquent, toutes considérations relatives à l'implantation (plusieurs sites ou un seul site) sont secondaires, et sans rapport, quant à la conception d'une BDR. Ce qui ne veut nullement dire sans importance, mais cela permet de bien séparer les problèmes.

Exemples : Reprenons la figure II.3

a) Des liens sémantiques peuvent exister entre données appartenant à différentes bases. Il faut alors formaliser ces liens [KAB 82].

b) On voudrait pour la BDR des loisirs pouvoir interroger les deux bases RESTAURANTS et CINEMAS simultanément. Par exemple, chercher les salles de cinéma et de restaurant situées dans une même rue. Il faudrait alors définir un langage qui puisse permettre d'exprimer des requêtes adressées à plusieurs bases.

c) Il faut aussi pouvoir définir des contraintes mettant en jeu des données appartenant à plusieurs bases. Par exemple, il est interdit de rapprocher des informations concernant le gérant d'un restaurant, et celui d'une salle de cinéma.

Ou bien (cf. figure II.4.a.) avant d'insérer une salle de restaurant ou de cinéma, vérifier qu'il existe une ligne de métro qui passe à proximité de l'endroit où est située cette salle.

A la lumière de ces exemples, on peut constater qu'ils ne concernent pas le fait qu'une BDR est implantée sur 1 ou plusieurs sites.

Remarques :

1) Un schéma global au sens de l'approche précédente, c'est à dire donnant de la BDR la vue d'une seule base, n'est plus nécessaire. On peut concevoir une BDR même si la définition d'un tel schéma est impossible. Toutefois, on peut le construire en tant que schéma d'une vue, si cela se révèle utile pour certains utilisateurs.

2) L'architecture en 3 niveaux d'ANSI-SPARC s'applique à une BDR, puisque la nouvelle propriété de répartition pour une base est une propriété logique.

3) La construction d'une base ensemble de bases est une opération qui préserve les propriétés, et les caractéristiques de chacune de ces bases. (Par exemple, on peut les interroger dans "leur" langage).

III.2 Avantages de l'approche :

L'hypothèse : une base de données est répartie si elle est un ensemble de bases de données, clarifie la notion de répartition ; En effet, on n'utilise que le concept mathématique d'ensemble et on suppose défini, celui de base de données [LIT 79].

De ce fait, cette approche nous semble plus méthodologique, vu que d'une part elle sépare les problèmes de conception de ceux relatifs à la réalisation, et d'autre part, elle permet une construction de BDR de manière progressive et structurée. Progressive, car la construction d'une base (répartie) se fait à partir de bases plus "élémentaires" qui

peuvent être faciles à réaliser, et structurée car une base élémentai peut elle-même être composée de bases plus élémentaires (cf. exem 2, figure II.4).

Ce sont là les points qui nous ont semblé essentiels dans cette approche.

III.3 Projet B A BA [LIT 81.a - 81.b]

B A BA signifie "Base Ansemble[1] de BAses"

Ce projet se situe dans le cadre de cette approche, et se donne pour objectif de démontrer la faisabilité d'un système permettant de gérer une multibase de données.

Des maquettes ont été développées pour cela. La maquette MUQUAPOL [KAB 82] (MUltibase QUel Adapté POLyphème) qui e une extension du SGBDR Polyphème [KAB 81], permet de constituer une multibase dont les bases sont relationnelles. Le langage de manipulation QUELM est une extension du langage QUEL [HEL 75] et permet de formuler des requêtes s'adressant à plusieurs bases.

MUQUAPOL, présente la particularité de permettre la définitio des liens, appelés dépendances, entre les bases de la multibase : dé dances sémantiques, d'intégrité, de confidentialité, et de traitement [KAB 81].

Une autre maquette MRDSM a été développée pour manipuler une multibase en prenant un SGBD classique, en l'occurrence MRDS, et en y ajoutant un interface tel que l'ensemble "interface + SGBD" puisse traiter une requête multibase [GUE 81].

(1) Selon l'auteur : "il n'y a aucune justification, sauf celles coutum à ce que le mot "ensemble" commence par un "e" au lieu d'un "a comme il n'y a aucune justification, sauf celles coutumières à ce une base soit un ensemble de données autres que des bases elles

Il faut donc noter que ces maquettes utilisent des SGBD (R)s existants.

III.4 Situation de TYP-R et ses motivations

III.4.1 Objectifs

TYP-R se situe dans le cadre de ce projet, et se distingue par le fait qu'il se propose d'étudier, et d'expérimenter les tendances récentes en matière de construction de logiciels fiables, évolutifs, faciles à maintenir, et de moindre coût, pour la construction et la gestion de bases de données réparties considérées comme des ensembles de bases[1]. Car en effet, nous considérons que cette approche définit un cadre formel pour une méthodologie de conception de Systèmes MultiBases (SMB) : ensemble de ces logiciels.

Ainsi, une conception modulaire et structurée par niveaux d'abstraction successifs [WIR 71] [DIJ 72] permet à de tels systèmes de s'adapter progressivement à la construction d'une multibase et d'être, en outre, "insensibles" quant à la répartition de celle-ci sur plus d'un site. Les concepts de modularité et d'abstraction nous semblent alors nécessaires pour développer de tels systèmes.

En effet, on peut considérer que globalement un SMB se compose de "machines" (figure II.5) structurées de façon à ce que chacune d'elles gère un ensemble de données. Celles-ci appartiennent à une seule base, exemple C pour la base CINEMAS, M pour METROS, etc..., on a plusieurs bases, exemple L pour LOISIRS, P-G pour PARIS-GUIDE.

Exemple : Considérons le SMB du point de vue interrogation.

(1) Nous dirons multibase dorénavant.

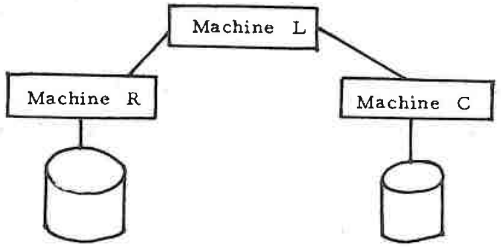


Figure II.5.a (cf. exemple 1 paragraphe II.III.1)

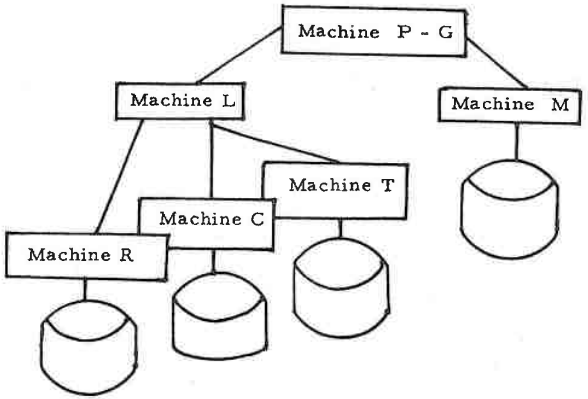


Figure II.5.b (cf. Exemple 2.a paragraphe II.III.1)

Figure II.5 - Schémas de principe d'un SMB.

Une machine se comporte alors comme un distributeur d'inform
Elle fournit des informations directement à un utilisateur, comme la
machine L (figure II.5.a), ou à une autre machine, comme la machine
M (figure II.5.b). Les informations qui constituent en quelque sorte, l
matière première d'une machine, proviennent de données enregistrées
sur un support physique (via un SGBD, un SGF, etc...) comme les mac
nes R, C, T et M, ou bien d'autres machines. Les machines L et P -
puisent leurs informations dans d'autres machines.

Précisons maintenant les notions d'abstraction, et de modularité, au regard de cette figure.

Ici la modularité, c'est la composition du système en éléments (ou modules) réalisant une fonction bien déterminée, et l'abstraction, la propriété qui fait qu'un module est utilisable en sachant ce qu'il fait sans avoir à connaître comment. Une machine est alors un module muni de cette propriété.

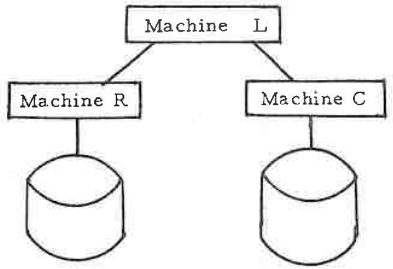
Ainsi, la machine P-G fournit des informations provenant des machines L et M, sans connaître comment celles-ci composent ces informations, et à fortiori à partir de quelles autres informations.

Il faut noter enfin que les considérations relatives à la répartition physique sur plusieurs sites, sont secondaires quant à la figure II.5.

III.4.2 Maquette d'un SMB réalisée

Nous avons réalisé une maquette, et l'avons présentée, avec celles décrites au paragraphe II.III.3, durant les journées de présentation du projet SIRIUS à PARIS, en Novembre 1981.

Pour cette maquette, nous nous sommes placés dans le cas simple suivant :



où les machines R et C sont identiques, et n'utilisent aucun SGBD existant.

Cela correspond à la multibase :

Multibase LOISIRS (L)

Base RESTAURANTS (R)

Base CINEMAS (C)

fin

où les bases RESTAURANT et CINEMA sont relationnelles, car en effet nous avons préféré le modèle relationnel (cf. I.II).

Cette maquette a permis de montrer qu'il était possible :

1) d'une part, de concevoir un système multibase sans se préoccuper notamment de contraintes pouvant résulter de la répartition des données sur plusieurs sites. Résultat important pour le concepteur et le réalisateur.

2) D'autre part, de rapprocher automatiquement des informations à partir de données situées dans plusieurs bases. Résultat utile pour l'utilisateur qui pourra formuler une requête s'adressant simultanément à plusieurs bases.

Pour la réalisation de cette maquette, nous avons utilisé le système TYP [CHA 82], qui nous a permis de disposer d'un environnement de programmation modulaire offrant la possibilité de manipuler des types abstraits de données. Ce qui va dans le sens de nos motivations. Le mécanisme d'abstraction de TYP, nous a donné le moyen justement de séparer l'utilisation d'une donnée des facteurs relatifs à leur structure de représentation, fichier en particulier et sa localisation.

Ces concepts étant nouveaux dans le domaine des bases de données relationnelles, nous avons dû dans un premier temps trouver une méthode [TOU 82], qui nous permette de prendre en compte les trois caractéristiques d'un système relationnel :

1) la définition de schémas relationnels ;

2) l'insertion, suppression et modification de tuples avec les règles d'intégrité correspondantes (d'entités et référentielles) ;

3) la manipulation des relations avec des opérateurs algébriques.

Le travail présenté dans cette thèse est organisé comme suit : dans le chapitre III, nous discuterons de ce qu'est une multibase en examinant quelques points essentiels, et fixerons les hypothèses prises pour la réalisation de la maquette.

Le chapitre IV, rappelle brièvement le système TYP, en particulier les aspects orientés bases de données.

Le chapitre V, décrit la réalisation de cette multibase sous TYP. On y trouvera aussi un développement des points 1), 2), 3) ci-dessus.

CHAPITRE III

DESCRIPTION D'UNE MULTIBASE ET HYPOTHESES PRISES DANS TYP-R

I - INTRODUCTION

Une multibase de données (ou multibase) est un ensemble de bases de données.

On peut concevoir une multibase pour modéliser le monde réel par plusieurs bases de données. En effet, les conceptions actuelles de bases de données ont toujours été fidèles au principe de donner à l'utilisateur la possibilité de manipuler une seule base de données, image d'un certain univers qu'on a modélisé [KAB 82]. Or un utilisateur peut éprouver le besoin de manipuler plusieurs bases en même temps [LIT 81], par exemple pour extraire des informations à partir de données appartenant à deux ou plusieurs bases. Si les SGBDR actuels permettent cela, un utilisateur croit néanmoins avoir affaire à une seule base définie par le schéma global de la BDR, et qui impose des restrictions telles que celles que nous avons citées.

Les bases d'une multibase peuvent être de bases de données existantes que l'on désire "faire coopérer" [GOD 81], un système multibase pourra alors permettre de gérer l'ensemble de ces bases, ou bien, des bases de données que l'on désire créer et manipuler ensemble (ou séparément, nous verrons plus loin ce que nous entendons par là). Un système multibase pourra alors permettre de créer ces bases, et de les manipuler ensemble.

Les critères pour définir une multibase peuvent être les suivants :

- l'ensemble des bases porte un nom

Exemple : une base de restaurants, une base de salles de cinéma , et une base de lignes de transport en commun peuvent former une multibase LOISIRS. On peut alors interroger cette dernière par

une requête du genre : quels sont les restaurants et les cinémas situés dans une même rue, désservie par une ligne de transport.

- Il existe des liens sémantiques entre les bases :

Exemple : à partir de la base de personnes abonnées à France-Loisirs et de la base de personnes abonnées à DIAL, on peut constituer une multibase et l'interroger pour, par exemple, sonder le niveau culturel, et les goûts artistiques des abonnés.

Par ailleurs, l'intérêt d'une telle multibase est extrême, si les deux organismes mettent en commun leur organisation.

- A partir de plusieurs bases servant à des applications différentes au sein d'une même entreprise, on constitue une multibase pour des applications "globales" au sein de l'entreprise. Ce dernier critère peut être l'un des plus importants.

II - SCHEMA D'UNE MULTIBASE RELATIONNELLE

Dans la suite, on ne va considérer qu'une multibase relationnelle c'est-à-dire un ensemble de bases relationnelles au sens de E.F. COD [COD 70] : ensemble de relations. (Cf. cas particulier ci-dessous).

Le schéma conceptuel d'une multibase relationnelle sera pour nous constitué de l'ensemble des schémas conceptuels normalisés (cf. I.II.2) de chaque base de la multibase.

III - EXEMPLE DE MULTIBASE

```
Multibase LOISIRS
    base RESTAURANTS
        SALLES (NUMR #, NOMR, TYPE, RUE, TEL)
        PLATS  (NUMP #, NOMP, NCAL)
        MENUS  (NUMR #, NUMP #, PRIX)
    fin
    base CINEMAS
        SALLES (NUMC #, NOML, RUE, TEL)
        FILMS  (NUMF #, NOMF, GENRE)
        SEANCES (NUMC #, NUMF #, HEURE, PRIX)
    fin
fin
```

Figure III.1 - Schéma d'une multibase relationnelle

Le langage de manipulation doit permettre d'exprimer des requêtes adressées simultanément à plusieurs bases. En particulier, les relations doivent être nommées dans une requête avec leur contexte, c'est-à-dire la base à laquelle elles appartiennent. Ceci pour éviter toute ambiguïté pouvant résulter de relations ayant un nom identique dans deux bases différentes (exemple : SALLES).

Avec une telle extension, tout langage relationnel, peut servir à manipuler une multibase relationnelle.

Exemples : Liste des noms de restaurants, et de cinémas situés dans une même rue. (En algèbre relationnelle cf. I. paragraphe II.3.1) :

```
PROJECT (JOIN (RESTAURANTS • SALLES • CINEMAS, SALLES,
                RUE = RUE), NOMR, NOMC) ;
```

- liste des noms de tous les restaurants

(PROJECT (RESTAURANTS . SALLES, NOMR) ;

Le schéma de la multibase LOISIRS n'est pas relationnel. Les base
RESTAURANTS et CINEMAS sont appelées bases locales.

Cas particulier : Une base locale peut elle-même être une multiba

Exemple : Multibase LOISIRS
Multibase RESTAURANTS
base KLEBER-REST
base MICHELIN-REST
fin
base CINEMAS
fin

Figure III.2

Cela est particulièrement intéressant, car la multibase resta
peut exister avant la création de la base cinémas. L'adjonction de c
dernière peut alors, se faire sans remettre en cause la multibase e
tante, donc sans modification de son schéma, du langage, ou de la c
respondance avec des schémas externes [LIT 81], et des program
existants.

Les bases locales qui ne sont pas des multibases sont appelée
bases élémentaires. Dans la suite on ne s'intéressera qu'aux multi
formées de bases élémentaires.

IV - ARCHITECTURE DE LA MULTIBASE

L'approche multibase est compatible avec les recommandations données par l'architecture ANSI-SPARC.

En effet, pour la multibase le fait d'être constituée de plusieurs bases est une propriété logique qui apparait dans son schéma conceptuel, qui décrit l'ensemble de toutes ces bases. Si on déduit de ce schéma des schémas externes décrivant des vues sur la multibase, il faudra définir les correspondances entre ces vues, et le schéma conceptuel.

Le schéma interne de la multibase est relatif à l'implantation, et est constitué de l'ensemble des schémas internes de chaque base.

D'où l'architecture en 3 niveaux :

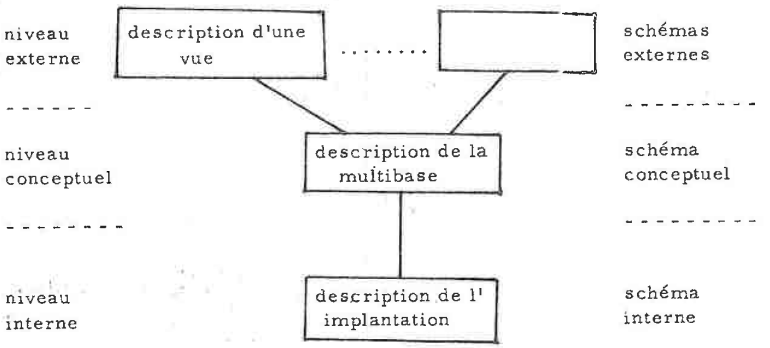


Figure III.3 - Architecture en 3 niveaux d'une Multibase.

Les requêtes sont adressées, soit en terme du schéma conceptuel, soit en terme de schémas externes. Dans ce cas, elles nécessitent un traitement préalable pour être traduites en terme du schéma conceptuel.

V - DEFINITION DE SCHEMAS EXTERNES (et de vue globale)

La définition des schémas externes, pour une multibase, est u problème de recherche à part entière, et est actuellement à l'étude p NASSIF R. Un nouveau problème se pose, dû au fait que le schéma c ceptuel est décrit comme un ensemble de bases qui sont elles-même des ensembles de relations.

On peut néanmoins noter les remarques suivantes :

1 - il est très utile de définir pour certains utilisateurs désirant manipuler séparément une seule des bases, un schéma exte donnant une vue sur chacune des bases de la multibase. (La multibas considérée est celle de la figure III.1).

Exemple : cas le plus simple, chaque base est décrite par une vue donnant le schéma complet de la base

```
vue U1
  base RESTAURANTS
       SALLES (...)
       PLATS (...)
       MENUS (...)
fin
```

Figure III.4 - Vue sur 1 base de la multibase

```
vue U2
  base CINEMAS
       SALLES (...)
       PROJECTION (...)   correspond à la relation
       FILMS (...)                SEANCES
fin
```

Figure III.5

La correspondance entre ces schémas externes, et le schéma conceptuel de la multibase est alors évidente, au changement des noms près. L'utilisateur peut en effet, éprouver le besoin de nommer différemment une base, une relation ou un attribut.

N.B. : Ce type de vue peut se généraliser à un sous-ensemble de la multibase, c'est-à-dire :

soit $MB = \{B_1, B_2, \ldots, B_n\}$

une multibase formée de bases B_i.

Toute combinaison $\{B_{i_1}, B_{i_2}, \ldots, B_{i_p}\}_{1 \leq p \leq n}$

est une vue possible.

L'exécution d'une requête sur ces vues ne nécessite aucun traitement préalable, sinon un changement de nom d'identificateurs éventuellement.

Exemple : SELECT (PROJECTION, ...) sera traduite par
SELECT (SEANCES, ...)

2 - Certains utilisateurs désirent avoir une vue partielle sur une ou plusieurs bases :

Exemple :

```
vue U3
  base  RESTAU-CHINOIS
    REST (NUMR #, NOMR, RUE, PHONE)   restaurants de
                                      type chinois
    CARTE (NUMR #, NUMP #, NOMP, PRIX) plats servis
                                       par ces res-
  fin                                  taurants.
  base SALLES-ART-ESSAI
    :
  fin
fin
```

Figure III.6 vue partielle sur la multibase.

La correspondance avec le schéma conceptuel (figure III-1) est la suivante (formulée à l'aide des opérateurs de l'algèbre relationnelle) :

```
REST = PROJECT (SELECT (RESTAURANTS.SALLES, TYPE=chinois),
                NUMR, NOMR, RUE, TEL)
CARTE = PROJECT (JOIN(JOIN (MENUS, SELECT (RESTAURANTS-
                                SALLES, TYPE = chinois),
                            NUMR = NUMR),
                        PLATS, NUMP = NUMP),
                    NUMR, NUMP, NOMP, PRIX) ;
   :
```

Pour éxécuter une requête d'interrogation sur cette vue, le traitement préalable serait de remplacer dans la requête les noms de relations par l'expression algébrique les définissant.

3 - Des utilisateurs désirent avoir une vue "globale" intégrée (une seule base) de la multibase :

Exemple : Soit la multibase (seule une relation est présentée pour chaque base) dont le schéma est :

```
Multibase RESTAURANTS
  base PETITS
    SALLES-1(NUMR #, NOMR, NB-PLACES, GERANT)
      :        restaurants avec un nombre de places ≤ 40.
  fin
  base GRANDS
    SALLES-2 (NUMR #, NOMR, NB-PLACES, GERANT)
      :        restaurants avec un nombre de places > 40
  fin
```



```
  base RESTAU                                  } restaurant
    REST (NUMR #, RUE, TYPE, TEL)              } du guide
      :                                        } PETIT-MALIN
  fin
fin
```

Figure III.7

On suppose que certains restaurants des bases PETITS et GRANDS sont répertoriés dans la base RESTAU.

Pour définir dans une vue "globale", une vue unique (une relation) sur tous les restaurants des 3 bases, il faut :

d'une part qu'un même restaurant soit identifié par le même numéro NUMR dans les différentes bases, et d'autre part que pour tous les restaurants, il existe une clé unique (l'attribut NUMR est défini sur 1 seul domaine commun aux 3 bases).

On peut alors définir la vue :

```
vue U4
  base RESTAURANTS
    RES (NUMR #, NOMR, NBPLACES, GERANT, RUE,
                 TYPE, TEL)
      :
fin
```

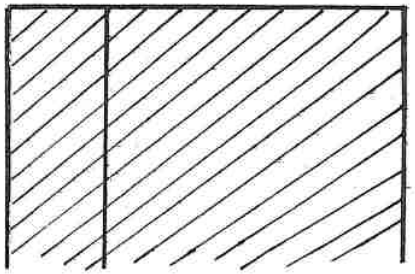
Figure III.8

Des attributs peuvent avoir des valeurs indéfinies dans cette relation RES. C'est le cas pour NOMR, NBPLACES et GERANT, s'il existe des restaurants dans les bases PETITS et GRANDS (figure III.7), qui ne figurent pas dans la base RESTAU.

C'est le cas aussi pour RUE, TYPE et TEL dans la situation inverse.

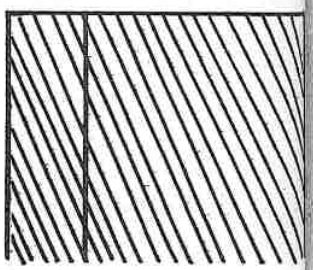
On peut illustrer cela comme suit :

NUMR NOMR NBPLACES GERANT



relation SALLE 1

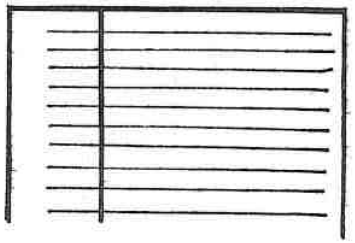
NUMR .....



relation SALLE 2

relation RESTAU

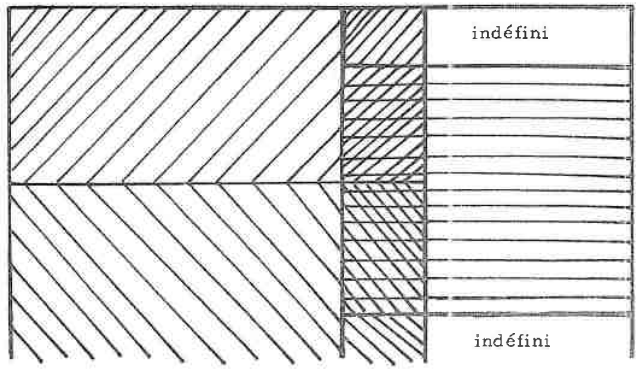
NUMR RUE TYPE TEL



on a alors :

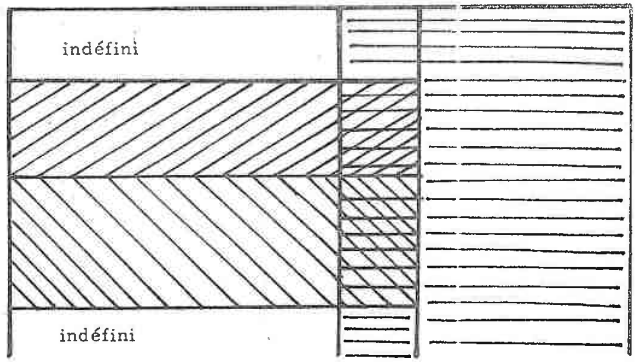
relation RES

NOMR NBPLACES GERANT NUMR RUE TYPE TEL



ou aussi

NOMR NBPLACES GERANT NUMR RUE TYPE TEL



etc...

Figure III.9

Si on considère maintenant que dans la base GRANDS, on a les restaurants de plus de 30 places, on aura des restaurants répertoriés dans les deux bases GRANDS et PETITS (ceux pour lesquels le nombre de places est compris entre 30 et 40). Le schéma U4 (fig. III-8) reste valable à condition de s'assurer qu'un même restaurant existe dans les deux bases GRANDS et PETITS avec la même valeur pour tous les attributs.

La correspondance entre la vue U4 et la multibase RESTAURAN sera :

RES = JOIN (REST, UNION (SALLE1, SALLE2), NUMR = NUMR

Cette vue U4 est très utile si on veut par exemple connaître le nom de tous les restaurants de la multibase. En effet, il suffirait de ne nommer dans la requête qu'une seule relation RES.

Mais, dans le cas général, il est difficile, sinon impossible de définir une telle vue [LIT 81]. Car, on peut supposer qu'il n'existe pas de clé unique pour tous les restaurants(1). C'est notamment le cas, si les bases existaient avant la création de la multibase, et si elles étaient conçues indépendamment l'une de l'autre.

Autre exemple :

Considérons une multibase de personnel d'une entreprise, créée à partir des bases suivantes :

```
base  SECTEUR 1
   EMP (INSEE #, NOM, NB-H, TAUX-H)
    :
    :
fin
```

où la relation EMP décrit des employés d'un secteur avec un numéro INSEE, un nom, le nombre d'heures et le salaire horaire.

(1) La jointure sur le critère NUMR = NUMR n'a alors aucun sens.


```
base  SECTEUR2
   E (INS# , NOM, SALAIRE)
    :
    :
fin
```

Figure III.10

où la relation E décrit des employés d'un autre secteur, avec un salaire mensuel, au lieu du nombre d'heures, et du salaire horaire.

Supposons que l'on ait :

EMP

INSEE	NOM	NB - H	TAUX - H
E1	SMITH	45	120

E

INSEE	NOM	SALAIRE
E2	SCHMIDT	5 500

Si on veut définir une vue "globale" sur tout le personnel de l'entreprise, on aura une relation PERS :

PERS (INSEE #, NOM, SALAIRE)

où SALAIRE sera obtenu pour les employés du secteur 1 par NB - H * TAUX - H.

Mais, dans cette vue, on ne peut plus accèder par exemple au taux-horaire de SMITH, et il n'est plus possible de le modifier lors d'une mise-à-jour. La propriété, un employé a un taux horaire, de la base SECTEUR1 est perdue.

Cet exemple montre bien qu'il n'est pas toujours utile, dans une multibase, de fournir une vue globale sur celle-ci.

Les relations EMP et E, bien qu'elles décrivent la même entité, ne sont pas identiques, car cette description est différente, de l'une à l'autre. Cette différence réside dans le fait qu'il existe un calcul pour passer d'une description à l'autre.

Les relations suivantes ne sont pas identiques non plus, bien qu'elles décrivent la même chose, une table.

TABLE1 (LGCM, DESCRIPTION)
TABLE2 (LGM, DESCRIPTION) (1)

où

LCGM est la longueur de la table en centimètres ;
LGM est la longueur de la table en mètres.

VI - INTERET ET DIFFICULTES DE DEFINITION DE VUE GLOBA

Quand plusieurs relations dans des bases différentes décrivent u même entité, il est intéressant de définir une vue globale de façon à pouvoir nommer une seule relation pour extraire des informations, à partir de toutes, ou de quelques unes des relations.

Les difficultés résultent du fait que ces relations décrivent la m entité de façon plus ou moins homogène, comme nous venons de le voi sur les quelques exemples précédents. K. KABBAJ [KAB 82] a étudi une autre approche qui consiste à définir formellement dans le schém conceptuel de la multibase, une relation d'équivalence entre deux schém relationnels : deux schémas sont dits équivalents s'ils décrivent la même chose.

(1) On peut remarquer qu'ici les relations TABLE1 et TABLE2 contiennent les mêmes informations, contrairement aux relations EMP et E de l'exemple précédent.

On définit ainsi des classes d'équivalence entre relations. Il suffit alors, dans une requête, de ne nommer qu'une seule relation pour avoir toutes les informations répondant à la même requête, et qui peuvent être extraites des autres relations appartenant à la même classe d'équivalence que la relation nommée.

Ces équivalences sont définies par l'administrateur de la base.

Néanmoins, les mêmes difficultés subsistent quand deux relations décrivent la même chose, mais de façon différente (par des schémas différents).

De telles relations sont alors dites hétérogènes.

VII - PRESENTATION DE LA MAQUETTE REALISEE

TYP - R est un système qui actuellement permet de manipuler une multibase. Il supporte la définition de vues sur une ou plusieurs bases de la multibase, (vues de type 1 étudiées précédemment).

Le schéma de la multibase est un ensemble de schémas de bases relationnelles. La multibase est celle de l'exemple du paragraphe III.

La définition de la multibase est en cours de réalisation [GR 82] : un langage de définition est proposé qui, traité par un générateur de programmes, permet de définir un ensemble de bases relationnelles.

Les liens sémantiques inter-bases sont laissés à la charge de l'utilisateur qui doit savoir exprimer les requêtes correspondant à ses besoins, et connaître l'univers décrit par le schéma conceptuel, ou la vue qui le concerne.

Les liens d'intégrité et de confidentialité inter-bases ne sont pas abordés dans cette thèse. Ils sont actuellement étudiés par R. NASSIF. Nous supposerons donc qu'il n'y a pas de contraintes inter-bases.

La caractéristique essentielle de TYP-R, et qui fait l'objet de cette thèse, est sa réalisation à l'aide du système TYP [CHA 82]. TYP est un système de programmation modulaire supportant la notion de type abstrait de données. Nous essayerons de montrer l'intérêt de cette réalisation dans un contexte de bases de données relationnelles.

Avant d'aborder la description de TYP-R, nous rappellerons TYP en mettant l'accent sur les caractéristiques essentielles qui ont servi dans la réalisation de TYP-R.

CHAPITRE IV

RAPPEL DU SYSTEME TYP

I - DESCRIPTION DU SYSTEME TYP [CHA 82]

Le système TYP est un système intégré de développement de programmes utilisant deux concepts étroitement liés : le concept de modularité, et le concept de type abstrait de données.

La modularité des programmes résulte du besoin de diviser un grand programme en plusieurs morceaux, compilés séparément. Cela pour en faciliter l'écriture, et donc la mise au point.

Décomposer un programme en plusieurs morceaux est un comportement basé sur la reconnaissance d'abstractions dans l'écriture des actions à accomplir. Chaque morceau de programme, appelé module, constitue une réalisation d'action. Ce morceau sera utilisable en faisant abstraction de cette réalisation, et par l'intermédiaire d'un interface.

On peut grouper dans un même module la réalisation de plusieurs actions individuelles ayant des caractéristiques, ou des propriétés communes (par exemple : partage d'une même ressource). Un tel module constitue vis à vis de l'utilisateur une sorte de "boîte noire" dont on sait quelles sont les fonctions. Un module de Parnas [PAR 72], en est un exemple. On verra plus bas qu'il en est de même pour un module du langage ATM.

La notion d'abstraction a été introduite pour les données par [LIS 74] et a donné naissance aux types abstraits de données. Il a été observé que pour l'utilisation d'une donnée on est intéressé uniquement par l'information qui y est contenue, et par son comportement qu'on peut spécifier par un ensemble d'opérations significatives vis à vis de cette donnée. Ce qui en permet l'utilisation de façon indépendante de la structure de représentation.

Un type abstrait peut donc être défini comme un ensemble d'opérations caractérisant une catégorie d'objets, et qui peuvent leur être appliquées. Ces objets sont dits, alors de ce type.

I.1 Le Langage ATM de TYP

ATM (abstraction, type, modularité) [MIN 79] est le composant principal du système TYP. C'est un langage modulaire qui possède l'originalité de bien séparer l'aspect concernant la spécification, et l'aspect concernant l'implantation aussi bien pour les données que pour les programmes. Ce qui fait distinguer quatre unités de compilation.

I.1.1 L'unité TYPE

Elle est définie par une partie syntaxique comprenant le nom du type, la liste d'opérations le caractérisant avec leur profil et une partie sémantique sous forme de commentaires spécifiant le comportement de ces opérations.

Exemple : soit une application qui traite de la gestion des restaurants d'une ville :
dans cette application un objet restaurant peut-être décrit par un type abstrait en ATM.

```
Type Restaurant ;
 build créer (integer, text, text, text, integer) ;
    #  opération de création d'objet avec en paramètre :
    #  le numéro du restaurant, son nom, la rue, son type
    #  de cuisine et son numéro de téléphone
 function numr  returns integer ;
 function nomr  returns text ;
```



```
 function rue   returns text ;
 function type  returns text ;
 function tel.  returns integer ;
   # fonctions d'accés aux différents constituants.
 modif changer tel (integer) ;
   #  opération de changement de numéro de téléphone
 consult convtext ⟶ text ;
   #  opération de conversion en type text
   :
   :
end
```

Les opérations d'un type sont de natures différentes. Ainsi, Build introduit une opération de création d'objet, function une opération de type fonction, modif et consul des opérations de type procédure : l'une avec modification d'objet, l'autre sans modification d'objet. Le symbole # introduit un commentaire.

A partir du moment où cette unité est compilée le type restaurant est connu du système. Il est alors possible de déclarer des variables de ce type :

```
r : restaurant ;
```

et d'écrire des appels d'opérations sur cette variable :

```
n := ∂r numr ;
```

où n est une variable de type entier (integer). Elle reçoit la valeur du numéro de restaurant désigné par r ;

```
∂r changertel (393 28 28) ;
```

changement du numéro téléphone de r en 393 28 28.
La notation '∂ variable opération' est un appel d'opération de type abstrait sur une variable déclarée de ce type.

La réalisation des objets de ce type restaurant peut ne pas être choisie, et donc définie ultérieurement, avant l'exécution de programmes utilisant ce type.

Types de base ou prédéfinis en ATM

Un certain nombre de types sont disponibles en ATM :

- les types de base : Integer, Real, Boolean et Char, dont les opérations sont reconnues par le compilateur (opérations usuelles : -, /, *, AND, OR, NOT, etc...).
- Les types prédéfinis :

  text : chaîne de caractères

  Sbit : chaîne de bits

  FDIR : fichier à accès direct

  FSEQ : fichier à accès séquentiel

  FIND : fichier à accès séquentiel indexé

  dont les opérations sont prédéfinies et compilées.

(exemple : get text ⟶ text, boolean pour un fichier indexé).

I.1.2 L'unité CAPSULE :

Cette unité permet de préciser le choix de réalisation des objets d'un type.

Exemple : (cf. Type restaurant)

```
Capsule restau for restaurant ;
  rep     numr : integer ;
          nomr : text ;
          rue  : text ;
          type : text ;
          tel  : integer ;
  end
build créer ...
      :
      :
```



```
function nomr returns text ;
  begin
      return nomr ;
  end
:
:
end
```

Une capsule définit la structure choisie pour représenter les objets d'un type abstrait (zone rep ... end) et le code représentant les algorithmes d'implantation des opérations définies sur ce type.

On peut définir plusieurs capsules pour un même type, correspondant à plusieurs choix de réalisation. C'est avant une exécution que l'on précise quelle capsule doit représenter un type. En particulier, on peut "changer" de capsule entre 2 exécutions d'un même programme. On a ainsi une indépendance totale entre la description abstraite d'un objet, donc sa manipulation dans un programme, et la représentation de cet objet.

I.1.3 L'unité MODULE et l'unité MACHINE

Comme nous l'avons dit au début de ce chapitre, un module constitue un morceau de programme écrit et compilé séparément dans un certain contexte[1].

En ATM, un module est constitué d'un ensemble de procédures pouvant partager une ressource commune. Cette ressource est constituée d'un ensemble de déclarations de variables de types prédéfinis ou de types abstraits appartenant au contexte. Les procédures contiennent des déclarations de variables locales et d'une suite d'instructions

(1) Celui constitué des unités déjà compilées et constituant un environnement de compilation.

décrivant un algorithme correspondant à une action. Les instructions se composent d'un certain nombre de structures de contrôle de bases fournies par le langage, d'appels d'opérations d'autres modules ou sur des variables déclarées d'un type abstrait, et d'instruction d'itération de haut niveau, et d'un genre nouveau (cf. IV.I.3).

Exemple : module qui gère l'ensemble des restaurants de la base. La ressource de ce module est constituée d'un fichier indexé. On suppose qu'un restaurant est identifié par son numéro.

```
Module  restaurants for  restaurants ;
  ressource
      f : find ;
  end
  :
  :
  proc  ajouter (r : restaurant) ;
     begin
        :
        :
        # corps de l'opération d'adjonction d'un restaurant
        # par  écriture sur f.
     end
proc supprimer (clé : integer) ;
     :
     :
proc  modifier (clé : integer, r : restaurant) ;
     :
     :
function existe (clé : integer) returns boolean ;
:          :
:          :
end
```

Pour pouvoir utiliser correctement ce module, qui constitue pour l'application une "boîte noire", il faut en fournir un interface où sont spécifiés les profils syntaxiques des opérations (procédures ou fonctions

effectuées et leur comportement.

Le texte constituant cet interface est appelé machine :[1]

```
machine  restaurants ;
  :
  :
  proc  ajouter  (restaurant) ;
        # adjonction d'un restaurant dans la base
  proc  supprimer  (integer) ;
        # suppression de restaurant dont le numéro est en paramètre
  proc  modifier (integer, restaurant) ;
        # modification du restaurant dont le numéro est en 1er
        # paramètre par le restaurant en 2ème paramètre
  function existe (integer)  returns  boolean ;
        # text d'existence d'un restaurant de numéro donné
  :
  :
  end
```

Un appel d'opération de machine s'écrit en ATM :

```
$  nom-de-machine nom-d'opérations Liste-de-para-
   mètre ;
$  restaurants ajouter (r) ;
```

on peut aussi définir plusieurs modules pour une même machine.

Après compilation des types et des machines avec les capsules et les modules correspondants, on peut constituer un programme exécutable. Celui-ci est le résultat, d'une connexion des différentes capsules et modules nécessaires à une exécution.

(1) Une machine doit cependant être compilée avant le module correspondant.

I.2 Le connecteur de TYP [HEN 80]

C'est un éditeur de liens dynamique qui permet lors de l'exécution d'un programme, de ne charger de mémoire centrale que les unités nécessaires à celui-ci et au fur et à mesure du besoin.

En particulier on peut, entre deux exécutions consécutives, remplacer une unité de programme (capsule ou module) par une autre, et cela grâce à une directive au connecteur.

Ceci est particulièrement important dans un contexte base de données : on peut changer le choix de connection d'une unité, pour avoir un meilleur temps de réponse par exemple, lors d'une deuxième exécution d'un programme si une première exécution n'a pas été satisfaisante. On réalise ainsi une indépendance totale vis à vis des structures physiques. On sait que c'est l'un des principaux problèmes des SGBD.

D'un autre côté, l'idée d'édition de lien dynamique permet de ne charger en mémoire centrale que les programmes nécessaires à une exécution donnée (d'une requête par exemple).

La technique de connexion sera aussi utilisée pour réaliser des contrôles de contraintes d'intégrité.

Des propriétés à vérifier pour des objets de la base pourront être contrôlées avant de connecter une unité de programme qui modifie cet objet (cf. fig. IV . 1)

Ainsi, les tests à effectuer avant une modification, ne sont pas écrits dans le programme qui fait cette modification mais ailleurs. Un tel programme ne change pas en même temps qu'une contrainte d'intégrité qui elle est susceptible d'évoluer.

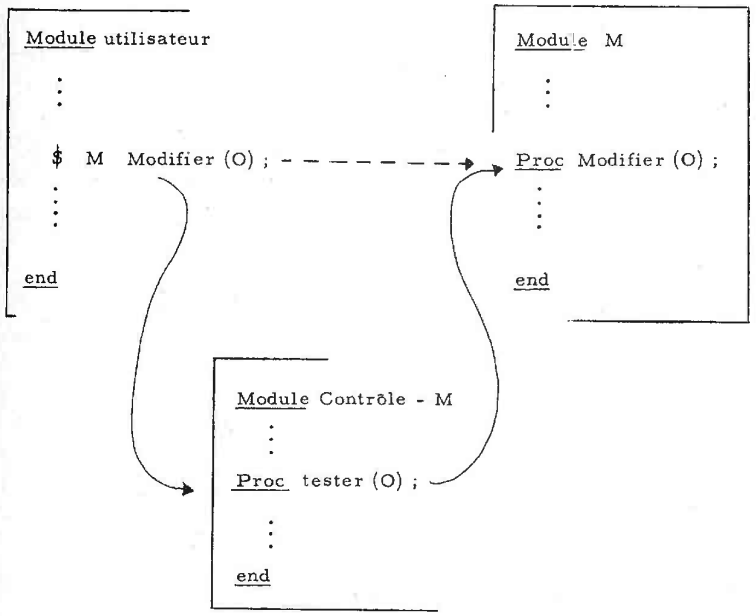


Fig. IV.1 Connexion avec contrôle

----------→ connexion directe

——————→ connexion avec contrôle

Cette technique est largement détaillée dans [HEN 80].

I.3 Les itérateurs dans TYP [TOU 79]

Pour manipuler un ensemble, il est parfois nécessaire de pouvoir écrire des instructions d'itération sur cet ensemble sans faire d'hypothèse sur la manière d'accèder aux différents éléments.

Si on veut lister par exemple tous les restaurants de la base, on a envie d'écrire :

```
pour chaque restaurant r
    faire
        écrire (r) ;
    fin
```

En ATM, nous avons la possibilité de définir un mécanisme d'itérateurs abstraits qui permet le traitement des éléments d'un ensemble, un par un comme une séquence d'éléments.

Pour cela, on définit une opération de nature particulière Iter qui doit être utilisée par des instructions foreach et first. Celles-ci sont des structures de contrôle de haut niveau.

1) Exemple d'utilisation d'un itérateur

On complète l'exemple de machine (du IV.I.1.3), avec une opération Iter :

```
Machine restaurants ;
   :
   :
   function existe (integer) returns boolean ;
   Iter extraire yields restaurant ;
     # production d'un restaurant à la demande d'une itérati
end.
```

Si maintenant, on veut lister le nom de tous les restaurants chinois, on peut écrire en ATM :

```
foreach r in $ Restaurants extraire
    where ∂ (∂ r type) eq ('chinois')
    then write (∂ r nomr) ;
endforec ;
```


Cette instruction foreach décrit un traitement (partie then) à effectuer pour chaque restaurant, désigné par la variable r implicitement déclarée du type restaurant, délivré par l'itérateur extraire défini dans la machine restaurants, et qui vérifient la propriété d'être chinois (partie where qui est facultative).

Si maintenant, on désire savoir s'il existe un restaurant chinois, on a une instruction first qui permet cela :

```
first r in $ restaurants extraire
    where ∂(∂ r type) eq ('chinois')
    then
        write (∂ r nomr) ;
    else
        write ('une pizza est aussi bonne')
endfirst
```

N.B. : ici la partie where est nécessaire.

Cette instruction first opère sur un restaurant vérifiant la propriété d'être de type chinois. La partie then de cette instruction décrit le traitement à faire s'il en existe un, et la partie else[1] décrit le traitement à faire si aucun restaurant ne satisfait cette propriété qui est exprimée dans la partie where. Dans le premier cas, on imprime le nom de ce restaurant, et l'itération s'arrête. Dans le second cas (l'ensemble est épuisé sans qu'aucun élément ne satisfasse la propriété) on imprime un message et l'itération s'arrête.

Les itérateurs permettent de faire abstraction de la manière dont les éléments d'un ensemble sont accèdés. Dans les instructions foreach et first on n'a pas besoin de connaître la stratégie d'accès aux objets qu'on traite, notamment l'ordre de parcours, stratégie qui est définie dans l'unité d'implantation par le corps de l'opération Iter (module restaurants ici).

---

(1) Nous n'avons pas jugé utile de définir else pour une instruction foreach.

En particulier, on peut remettre en cause cette stratégie sans modifier les instructions d'itérations. Ce qui contribue à réaliser l'indépendance entre les données et les programmes.

2) Exemple de réalisation d'un itérateur

Le texte ATM réalisant l'itérateur extraire pourra être : en reprenant le module restaurants

```
Modules restaurants for restaurants :
   ressource
     f : find ;          #  fichier à accès séquentiel indéxé
   end
:
:
Iter extraire yields restaurant ;
 Var
     r : restaurant ; t : text ;
     b : boolean ;
 end
 begin
   @ f openg                        #  ouverture du fichier
   @ f get ⟶ t, b                   #  accès au ler enregistrement
 while b ;
       :                  #  conversion d'un objet de type text
       :                  #  (enregistrement lu t) en objet de
                          #  type restaurant (r).
     yield r ;
     @ f get ⟶ t, b ;
 end while ;
 @ f close ;
 end
end
```


Ici la stratégie d'accès est simple : c'est le parcours séquentiel d'un fichier f choisi pour représenter l'ensemble des restaurants conservés dans la base.

L'instruction yield a pour effet de délivrer un objet à l'instruction foreach ou first qui a invoqué l'itérateur.

II - PROGRAMMATION EN T Y P D'UNE APPLICATION ORIENTEE BASE DE DONNEES

Dans ce paragraphe, nous allons décrire brièvement, comment nous pouvons utiliser le système TYP pour développer une application utilisant une base de données. La programmation est faite en ATM.

Nous supposons définies, sans nous préoccuper comment, les différentes entités représentées dans la base, leurs différents constituants ainsi que les différentes opérations de manipulation.

Il convient pour nous de distinguer deux niveaux de description.

II.1 Description d'un objet

Un objet vu comme une association d'objets élémentaires est décrit à ce niveau. Par exemple, un restaurant est un objet composé d'un numéro, d'un nom, d'un type de cuisine, d'une rue et d'un numéro de téléphone. Cette description a pour effet de faire abstraction de cette association et de sa représentation en ne fournissant qu'une liste d'opérations qu'on désire effectuer sur un objet.

On définit ainsi des types abstraits de données.

Exemple :

a - Type restaurant (cf. IV.I.1.1)

b - Type plat

```
        # description d'un plat
    build  créer (integer, text, integer) ;
        # création d'un objet plat à partir, respectivement, d'u
        # numéro de plat, d'un nom de plat, et du nombre de
        # calories du plat.
    function  nump  returns  integer ;
        # accès au numéro d'un plat.
    function  nomp  returns  text ;
        # accès au nom d'un plat
    function  ncal  returns  integer ;
        # accès au nombre de calories qu'il y a dans un plat.
      :
    end
```

on suppose qu'un plat est identifié par son numéro.

c - Type menu

```
        # description d'un menu
    build  créer (integer, integer, integer) ;
        # création d'un objet menu à partir du numéro du res-
        # taurant servant ce menu, le numéro du plat servi,
        # et son prix.
    function  numr  returns  integer ;
        # accès au numéro du restaurant.
    function  nump  returns  integer ;
        # accès au numéro du plat servi.
```



```
    function  prix  returns  integer
        # accès au prix de ce menu.
      :
    end
```

On suppose qu'un menu est identifié par le numéro du restaurant qui le sert, et par le numéro du plat servi. Autrement dit, deux plats différents, servis par un même restaurant sont dans deux menus différents, et deux restaurants différents servant un même plat sont dans deux menus différents aussi.

II.2 Description d'une classe d'objets

C'est la description abstraite de chaque classe d'objets d'un même type (restaurant, menu, plat) qu'il y a dans la base.

Cette description doit rendre compte des objets actuellement stockés et doit donc permettre des manipulations du genre adjonction, suppression, modification, accès, parcours, etc...

Exemple : Machine restaurants ;
(cf. IV.I.1.3).

```
machine plats ;
    proc ajouter (plat) ;
    proc supprimer (integer) ;
    proc modifier (integer, plat) ;
    proc accès (integer) → plat, boolean ;
    Iter extraire yields plat ;
    function existe (integer) returns
                                boolean ;
end
```

```
machine menus ;
  :
proc modifier (integer, inte-
                ger, menu) ;
proc accès (integer, interger)
        → menu, boolean ;
  :
end
```

II.3 Exemples de manipulations sur la base

La manipulation de cette base se fait par programmes où l'on déclare des variables de type restaurant plat et menu.

```
Exemple : r : restaurant ; n : integer ; b : boolean ; ...
    ...   $ restaurants accès (n) ⟶ r, b
          if b then
                ∂r changer tel (393 28 28)
                $ restaurants modifier (n, r) ;
          endif ;
          # modification du numéro de téléphone d'un restaurant.
```

Ces programmes constituent des applications des utilisateurs de la base. En particulier, l'interrogation de la base se fait par programmes, écrits ponctuellement en vue d'une exécution occasionnelle, ou déjà compilés et stockés dans une bibliothèque.

Regardons quelques exemples de programmes d'interrogation :

R1 - Quels sont les noms et rues de tous les restaurants :

```
foreach r in $ restaurants extraire
    then
        write (∂ r nomr, ∂ r rue) ;
endforec ;
```

R2 - Quels sont les noms et rues de tous les restaurants chinois :

```
foreach r in $ restaurants extraire
    where ∂ r type = 'chinois'
    then
        write (∂ r nomr, ∂ r rue) ;
endforec ;
```


R3 - Quels sont les restaurants servant des menus d'un prix inférieur à 50,00 F

```
foreach r in $ restaurants extraire
  then
      first m in $ menus extraire
          where (∂ m numr = ∂ r numr and
                 ∂ m prix < 50)
          then
              write ( ∂ r nomr) ;
          else ;
          endfirst ;
endforec ;
```

ou encore :

```
foreach m in $ menus extraire
    where ∂ m prix < 50
    then
        $ restaurant accès (∂ m numr) ⟶ r , ok ;
        if ok #  toujours vrai
        then write (∂ r nomr ) ;
        endif ;
endforec ;
```

NB : ok toujours vrai veut dire qu'il n'existe pas de menu, qui n'est servi par aucun restaurant. On suppose que c'est le cas.
Ici on peut sortir plusieurs fois le nom d'un restaurant.

R4 - Même requête que R3, avec en plus le nom des plats servis à ce prix.

```
foreach m in $ menus extraire
    where ∂ m prix < 50
      then
           $ restaurants accès (∂ m num r)→ r, okr ;
           $ plats        accès (∂ m nump)→ p, okp ;
        if (okr and okp)  ## toujours vrai
        then
             write ( ∂ r nomr, ∂ p nomp) ;
        endif ;
endforec ;
```

R5 - Quels sont les restaurants qui servent chacun tous les plats (de la base) en menu.

Autrement dit, si un restaurant r sert des plats en menu, et s'il n'existe pas de plat p dans la base qui n'est pas servi par r, alors sortir le nom de r.

```
foreach r in $ restaurants extraire
  then first m in $ menus extraire
       where (∂ m numr = ∂ r numr)
       then  ## r sert des plats en menu
           first p in $ plats extraire
               where not ($menus existe ( ∂ rnumr, ∂ pnump))
               then
                    ## p existe qui n'est pas servi par r en m
                    ;
               else
                    write ( ∂ r nomr) ;
               endfirst ;
         else  ## r ne sert aucun plat en menu
       endfirst ;
endforec ;
```

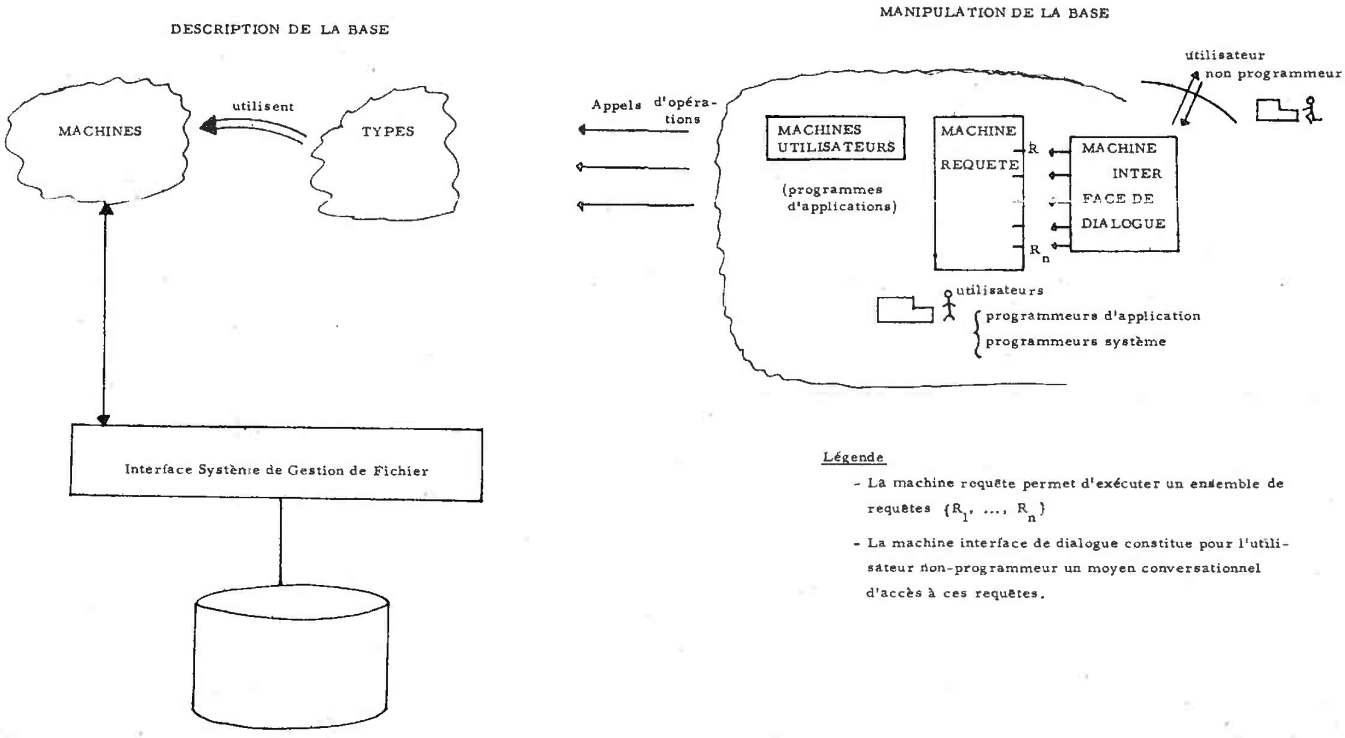

II.5 Utilisation de la base

Cette base peut être utilisée par des utilisateurs programmeurs ou des utilisateurs non-programmeurs.

Les utilisateurs programmeurs, sont ceux qui développent des programmes d'application écrits en ATM, où qui interrogent la base par programmes. Ils ont alors un environnement de programmation sous TYP contenant les types et les machines qui décrivent les objets de la base.

Les utilisateurs non-programmeurs sont ceux qui disposent d'un ensemble de programmes de "bibliothèques" constituant des requêtes. Ils les utilisent par l'intermédiaire d'un interface de dialogue, programme écrit en ATM. Ils n'accèdent à la base qu'à travers cet interface qui leur fournit une sorte de "menu" de toutes ou certaines (pour raison de confidentialité) requêtes mises à leur disposition. Les requêtes R1, ..., R5 vues précedemment en sont des exemples d'interrogation.

II.6 Schéma de description d'une base en TYP



Légende

- La machine requête permet d'exécuter un ensemble de requêtes $\{R_1, \ldots, R_n\}$
- La machine interface de dialogue constitue pour l'utilisateur non-programmeur un moyen conversationnel d'accès à ces requêtes.

Fig. IV.2 - ARCHITECTURE D'UNE BASE EN TYP

CHAPITRE V

REALISATION D'UNE MULTIBASE RELATIONNELLE : DESCRIPTION DE LA MAQUETTE

V. I - INTRODUCTION

La multibase que nous avons implantée est celle décrite par le schéma de la fig. V.1.

Nous rappelons qu'une multibase est un ensemble de bases, et que nous nous intéressons au cas où ces bases sont relationnelles, c'est-à-dire décrites par un ensemble de relationsen 3 F N (cf. ch. I - paragraphe II.2) supposée établie par ailleurs. Nous rappelons aussi que nous ne prenons pas en compte les différents liens pouvant exister entre les données de différentes bases. En particulier, l'utilisateur doit connaître la signification des relations de chaque base et savoir exprimer sa requête de la manière appropriée.

```
Multibase LOISIRS
   base RESTAURANTS
      SALLES (NUMR #, NOMR, RUE, TYPE, TEL)
      PLATS  (NUMP #, NOMP, NCAL)
      MENUS  (NUMR #, NUMP #, PRIX)
   fin
   base CINEMAS
      SALLES (NUMC #, NOMC, RUE, TEL)
      FILMS  (NUMF #, NOMF, GENRE)
      SEANCES (NUMR #, NUMF #, HEURE, PRIX)
   fin
fin
```

Fig. V.1 Multibase de Loisirs.

V.I.1 Le langage de définition de la multibase

C'est le langage par lequel on décrit le schéma de la multibase.

Cette description commence par le nom de la multibase, exemple :

Multibase LOISIRS

suivie de la description de chaque base constituante.

Description d'une base : exemple de la base RESTAURANTS

on commence par donner le nom de la base, différent pour chaque base :

base RESTAURANTS

suivi des déclarations de domaines : nom de domaine et représentation (text ou integer), introduites par le mot clé domaine et séparées par des virgules :

```
Domaines
        NUMERO        :  integer,
        NOM           :  text,
        RUE           :  integer,
        NB-CALORIES   :  integer,
        MONNAIE       :  integer,
        TELEPHONE     :  integer,
        CUISINE       :  text,        # type de cuisine
        COMMUN        :  text,        # nom commun
fin
```

fin indique la fin de la liste.

Les noms de domaines servent à vérifier la cohérence des jointures entre relations.

Ensuite, on déclare les attributs sous la forme :

```
Attributs
    NUMR, NUMP  :  NUMERO,
    NOMR        :  NOM,
    NOMP        :  COMMUN,
    NCAL        :  NB-CALORIES,
    PRIX        :  MONNAIE,
    RUE         :  RUE,
    TEL         :  TELEPHONE,
fin
```

Le nombre d'attributs doit être au moins égal à celui des domaines.

Enfin, on décrit les relations de la base RESTAURANTS en donnant des noms, tous différents, aux relations et pour chaque relation on définit une liste d'attributs, tous différents, parmi ceux déclarés. Dans les attributs définissant ainsi une relation, on déclare ceux qui forment la clé primaire et ceux qui forment les clés secondaires, le cas échéant

Cela est donné sous la forme :

```
Relations
    SALLES (NUMR, NOMR, RUE, TYPE, TEL),
        clé primaire NUMR ;
    PLATS (NUMP, NOMP, NCAL), clé primaire NUMP ;
    MENUS (NUMR, NUMP, PRIX),
      clé primaire (NUMR, NUMP),
      clé secondaire NUMR, NUMP ;
    fin
```

le couple (NUMR, NUMP) constitue la clé primaire pour la relation MENUS. NUMR et NUMP sont clés secondaires pour cette même relation.

Les clés secondaires servent à introduire des contraintes référentielles à respecter (cf. V.III.5). Ainsi par exemple, NUMR est clé primaire pour SALLES et en même temps clé secondaire pour la relation MENUS. Cela implique que toute valeur associée à cet attribut dans cette dernière relation, existe comme valeur pour ce même attribut dans la relation SALLES.

Le langage de définition présenté ici est en cours d'implantation. Un traducteur [GRI 82] traduit ce langage en générant des textes de programmes ATM. Ceux-ci sont actuellement écrits "à la main". Nous essayerons de montrer (paragraphe V.II.2) que cette écriture est systèmatique.

V.I.2 Le langage de manipulation de la multibase

C'est le langage avec lequel l'utilisateur formule une requête à la multibase. Selon le type de celle-ci, on distingue :

V.I.2.1 Le langage d'interrogation

C'est un langage algèbrique utilisant les opérateurs de l'algèbre relationnelle : PROJECT, JOIN et SELECT, (cf. ch. I, paragraphe II.3.1) pour formuler une requête d'interrogation. Celle-ci peut-être adressée à plusieurs bases. Il faut pour cela qu'une relation soit nommée dans la requête avec son contexte, c'est-à-dire la base à laquelle elle appartient. Cela pour éviter toute ambiguïté pouvant résulter du fait de l'exitence de relations ayant même nom dans deux bases différentes.

La notation choisie est :

RESTAURANTS . SALLES pour la relation SALLES de la base RESTAURANTS par exemple. Néanmoins cette notation est facultative si le nom d'une relation est unique dans toute la multibase :

PLATS au lieu de RESTAURANTS . PLATS par exemple.

Syntaxe du langage :

Opérateur PROJECT :

PROJECT (< Relation > , < Critère de projection >) ;

< Relation > est un nom de relation éventuellement préfixé par le nom de la base à laquelle elle appartient.

< Critère de projection > est une liste d'attributs (sans duplication) appartenant à la relation spécifiée et séparés par des virgules.

Opérateur SELECT :

SELECT (< Relation > , < critère de sélection >) ;

< critère de sélection > est de la forme 'attribut' θ constante' où attribut est un nom d'attribut appartenant à la relation spécifiée, et θ un des symboles : $\{=, \neq, \geq, \leq, >, <, \}$. La constante est numérique ou alphanumérique selon la représentation du domaine de l'attribut spécifié.

Opérateur JOIN :

JOIN (< Relation >$_1$, < Relation >$_2$, < critère de jointure >) ;

< critère de jointure > est de la forme 'attribut$_1$ θ attribut$_2$ où attribut$_i$ appartient à la relation$_i$. θ a la même signification que pour SELECT. Attribut$_1$, et attribut$_2$ sont définis sur le même domaine.

Ces opérateurs ont pour résultat une relation, ils peuvent être imbriqués les uns dans les autres.

Nous reviendrons sur ce langage, et sur le traitement d'une requête au paragraphe II.4.1.

V.I.2.2 Langage de mise à jour

Ce langage concerne les opérations de suppression, de modification ou d'insertion d'un tuple dans une relation. Ces opérations sont respectivement :

- DELETE (< Relation > , < clé >) ;
- MODIFY (< Relation > , < clé >, < Tuple >) ;
- INSERT (< Relation > , < Tuple >) ;

et qui signifient dans l'ordre :

- supprimer dans la relation en paramètre le tuple de clé donnée ;
- modifier dans la relation en paramètre le tuple de clé donnée selon le tuple donné ;
- insérer dans la relation en paramètre le nouveau tuple donné.

Nous détaillerons ce langage de mise à jour au paragraphe V.II.4.2, avec les contraintes à vérifier à chaque exécution de l'une de ces opérations.

V - II CONCEPTS MIS EN ŒUVRE POUR LA REALISATION DE TYP-R

V.II.1 Les concepts

Nous avons utilisé le système TYP (cf. ch. IV) pour cette réalisation. Les caractéristiques d'abstraction et de modularité dans TYP, nous ont semblé essentielles pour développer un logiciel de bases de données, relationnel notamment.

Ceci pour plusieurs raisons :

- Le langage ATM de TYP permet une séparation entre la définition abstraite (fonctions de manipulation) et la représentation (structure de données et algorithmes réalisant les fonctions de manipulation) d'un objet. Il est possible d'avoir plusieurs représentations pour une même unité d'abstraction, et changer l'une par l'autre avant toute exécution de programme.

On réalise ainsi une "data indépendance" au niveau interne de la base[1]. Quelques rares systèmes permettent cela, exemple PIVOINES [CRE 74] .

- L'écriture modulaire des programmes implantant la base, rendent cette implantation facile à écrire, mettre au point et à maintenir. La fiabilité et l'évolutivité de la base sont donc renforcées.

Par ailleurs, le fait d'encapsuler les structures de données, et les algorithmes associés dans un "module" permet de mieux concevoir l'architecture de l'implantation, et de bien localiser les différentes actions.

- La fabrication dynamique de programmes exécutables, par la technique de connexion dans TYP, permet de ne charger en mémoire centrale que les unités de programme nécessaires à une exécution donnée, en particulier d'une requête, et de modifier à tout moment, entre deux exécutions, certains choix d'implantation (algorithmes, représentations).

- Le traitement relationnel [COD 82] est rendu possible en manipulant les relations comme des séquences de tuples grâce aux itérateurs. L'interprétation et l'exécution d'une requête sont relativement rapides.

---

(1) Quand il n'y pas d'ambiguïté, on dira base tout court, au lieu de multibase.

V.II.2 Choix de réalisation

Dans une base de données relationnelle les objets manipulés sont des relations. Il est donc évident qu'il se pose le problème de choix de représentation d'une relation.

A) Situation du problème :

Rappelons que l'utilisateur de la base crée une relation en déclarant son schéma et doit pouvoir le manipuler avec des opérateurs algébriques, la construire par des insertions de tuples et éventuellement la mettre à jour.

Comment représenter en TYP cette relation ? ou plutôt doit-on la "représenter" comme étant un type abstrait de donnée, ou la représenter avec un (ou des) type (s) abstrait (s) de donnée ?

Nous sommes enclin vers cette deuxième possibilité. Car une relation est à première vue un objet assez complexe du fait qu'il existe une sorte de double dimension. Dimension qu'on peut appeler 'verticale' qui est dûe à l'existence d'attributs, et dimension 'horizontale' qui elle, est liée à l'existence d'un certain nombres de tuples dans la relation. Ces deux dimensions, en fait, sont liées aux notions d'"intension" et d'"extension" d'une relation. [DAT 81].

Dans le premier cas, utile au niveau conceptuel car il a trait au concept même de la relation, et à toute les relations potentiellement possibles, on traite de ces dernières comme étant constituées d'une liste d'attributs. On parle alors en terme de normalisation, de dépendances fonctionnelles et d'opérations algébriques de projection ou de jointure qui opèrent sur les colonnes d'une relation.

Le second aspect par contre, est nécessaire à l'implantation et à la programmation car il a trait aux relations actuellement existantes dans une base. On traite alors des relations comme étant constuées d'un ensemble de tuples et essentiellement des opérations d'accès à un tuple (insertion, modification etc...).

B) Différentes solutions :

Il est en fait indispensable pour implanter une relation de tenir compte de ces deux aspects. Pour le réalisateur il se pose alors le problème de savoir s'il faut en tenir compte dans une même spécification, ou dans deux spécifications différentes, et qui seraient complémentaires.

En effet, une première possibilité serait de définir un type abstrait relation, paramétré par le nombre de tuples et la liste des attributs. Nous aurons donc un seul niveau de paramétrisation puisqu'on tient compte des deux dimensions dans une même spécification de type abstrait avec des opérations de création, insertion ou suppression de tuple, projection, sélection, jointure, etc... qui ont pour résultat un objet de type relation(1).

Toutes les relations stockées dans la base sont de ce type. Il est alors immédiat d'interpréter les requêtes de mise à jour ou d'interrogation. (Ce sont des opérations définies sur le type). Mais dans la pratique on rencontre de sérieux problèmes :

1 - la spécification de type relation est très lourde et difficile à décrire complétement et avec précision et clareté [PAO 80]. Le nombre d'opérations introduites devient arbitrairement grand au fur et à mesure des spécifications. Ce qui s'avère inutilisable à l'implantation pour devenir plutôt un exercice très pratiques de spécification de types abstraits comme le signale [THO 80].

(1) Un exemple de telle spécification est donné [MAI 81] et [THO 80].

2 - Les relations résultats d'opérateurs relationnels imbriqués doivent être représentées entièrement en mémoire.

3 - Les performances seraient alors prohibitives.

En fait, cette approche est utilisée par un analyste qui veut spécifier abstraitement le comportement d'un système de base de données relationnelles. Il doit considérer une relation comme un type abstrait muni de certaines (beaucoup) opérations [MAI 81].

Une deuxième possibilité consisterait à tenir compte des deux dimensions d'une relation en deux étapes différentes distinguant ainsi, les deux niveaux de paramétrisation.

L'idée est la suivante : un tuple est d'un certain type, et la relation est du type 'ensemble de tuples de ce type'.

C'est ce qui ressort d'ailleurs d'une discussion à ce sujet parue dans [SIG 80].

En voici une description (simplifiée pour l'exemple) :

type Ensemble-de-tuples : E

créer :	⟶ E ;
ajouter : E X Tuple	⟶ E ;
supprimer : E X Tuple	⟶ E ;
accèder : E X Clé	⟶ Tuple ;
extraire : E	- - - - - ⟶ Tuple ;
⋮	

fin

- - - - - ⟶ signifie que l'opération est un itérateur.
Ceci correspond au prenier niveau de paramétrisation. Une relation est un ensemble de tuples.

Au second niveau, celui de la définition du type du tuple constituant les éléments de la relation, nous avons les opérations d'accès aux composants du tuple, et des "pseudo-opérations" de l'algèbre relationnelle qui vont permettre d'interpréter les opérateurs JOIN, PROJECT et SELECT en les évaluant tuple par tuple.

En voici une description (simplifiée pour l'exemple) :

type tuple ;

créer	:	entier	⟶ tuple ;
const	:	tuple X entier X attribut X valeur	⟶ tuple ;
att	:	tuple X entier	⟶ attribut ;
val	:	tuple X entier	⟶ valeur ;
Project	:	tuple X critère-projection	⟶ tuple ;
Join ?	:	tuple X critère-de-jointure X tuple	⟶ booleen ;
Concat	:	tuple X tuple	⟶ tuple ;
Select ?	:	tuple X critère-de-sélection	⟶ booleen ;
rg	:	tuple X attribut	⟶ entier ;

fin

L'opération extraire du type relation est un itérateur qui transforme une relation en une séquence de tuples :

Critère-de-projection, critère-de-seléction, critère-de-jointure sont des types prédéfinis qui représentent respectivement (une liste d'attributs), (attribut θ valeur), (attribut θ attribut), où $\theta \in \{=, \neq, \leq, \geq, <, >\}$.

L'interprétation des opérateurs relationnels se fera alors en deux stades. Par l'itérateur extraire on accède, un par un, à tous les tuples et par les opérations concat, select ?, join ?, et project, on réalise la fonction désirée.

Exemple :

relation restaurants

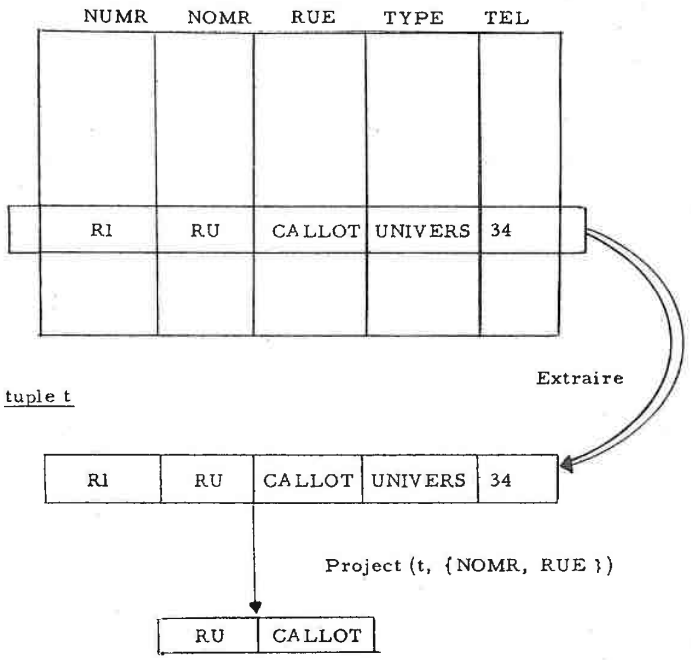


Fig. V.1 - Interprétation de PROJECT (restaurants, NOMR, RUE)

V.II.3 Solution TYP R :

En TYP R nous avons choisi cette deuxième solution qui nous a paru plus simple à réaliser. Les types relation et tuple ci-dessus

constituent pour nous un modèle d'implantation de relations.

Dans ce qui suit, nous étudierons en détail comment les choix de réalisation retenus sont mis en œuvre, avec les concepts retenus.

V - III DESCRIPTION DES IMPLANTATIONS DES RELATIONS EN TYP - R

Les textes ATM des programmes implantant une relation seront générés [GRI 82] par un générateur qui accepte en entrée les déclarations de schéma de relations.

Ces textes sont actuellement écrits "à la main".

Ils sont constitués de quatre unités : un type et sa capsule, une machine et son module.

Le type décrit les éléments de la relation.

La machine décrit l'ensemble des objets de ce type conservés dans la base.

Soit la relation R (A_1 #, A_2 #, ..., A_k #, A_{k+1}, ..., A_n) où A_i sont des attributs définis sur des domaines D_i $i \in [1, n]$. Pour simplifier l'exposé on confondra nom de domaine et sa représentation. On suppose que les k premiers attributs ($1 \leq k \leq n$) constituent la clé primaire de R.

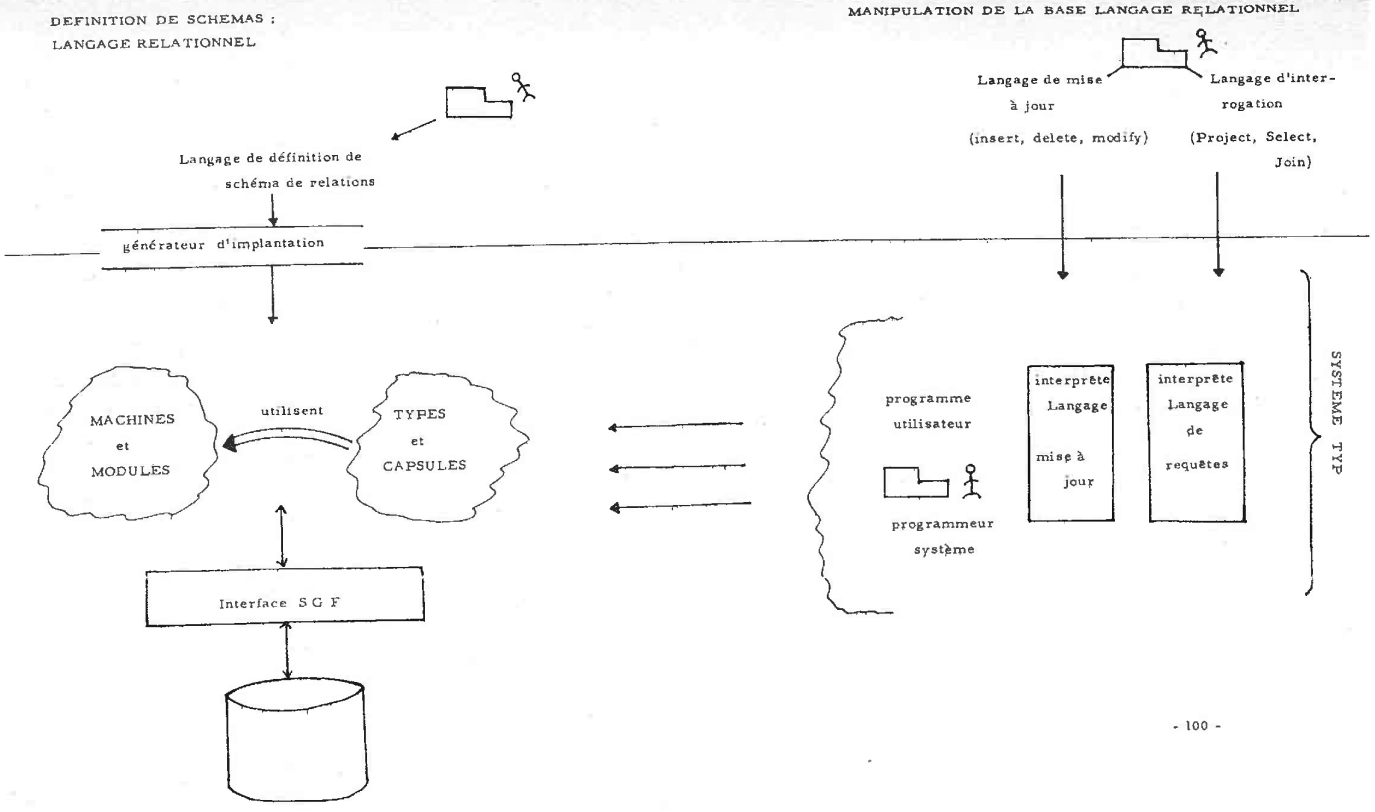


Fig. V.3 - ARCHITECTURE D'UN SYSTEME RELATIONNEL EN TYP.

V.III.1 Unité type appelée le t-type de la relation :

```
type rb ;
build créer ( d1, d2, ..., dn) ;
function a1   returns   d1
function a2   returns   d2
   :
function an   returns   dn
fin
```

Les fonctions sont déduites du schéma de la relation. Elles ont pour nom le nom d'attribut, et retournent des valeurs du type représentant le domaine de cet attribut.

On a :

$$a_i(\text{créer } (v_1, v_2, \ldots, v_n)) = v_i$$

pour tout $i \in [1, n]$;

Exemple : pour les relations de la base RESTAURANTS nous avons les t-types : (en ATM).

```
type salle ;
   build créer (integer, text, text, text, integer) ;
   function numr returns integer ;
   function nomr returns text ;
   function rue  returns text ;
   function type returns text ;
   function tel  returns integer ;
   function equal (plat) returns boolean ;
   consult convtup ⟶ tuple ;
end
```

```
type plat ;
    build créer (integer, text, integer) ;
    function nump returns integer ;
    function nomp returns text ;
    function prix returns integer ;
    function equal (plat) returns boolean ;
    consult convtup ⟶ tuple ;
end

type menu ;
    build créer (integer, integer, integer) ;
    function numr returns integer ;
    function nump returns integer ;
    function prix returns integer ;
    function equal (menu) returns boolean ;
    consult convtup ⟶ tuple ;
end
```

Les opérations equal et convtup, décrites communément pour les t-types ci-dessus, ont été introduites pour un besoin interne, en particulier l'opération convtup. Comme nous le verrons au paragraphe (IV.4) lors de l'interprétation d'une requête, on a besoin de gérer dynamiquement des schémas intermédiaires de relations. Disons pour l'instant que le type prédéfini, tuple, est nécessaire pour manipuler les objets de différentes relations de la même manière.

V.III.2 Une unité machine, que nous appellerons la r-machine de la relation.

Cette machine est chargée de gérer les objets

(éléments de la relation) physiquement stockés. Ces objets sont ceux du t-type de la relation.

Considérons la relation R (cf. V.III) et son t-type rb.

```
machine rbs ;
    proc init ;
    proc ajouter (rb) ;
    proc supprimer (d1, d2, ..., dk) ;
    proc modifier (d1, d2, ..., dk, rb) ;
    proc accès (d1, d2, ..., dk) ⟶ rb, boolean ;
    function existe (rb) returns boolean ;
    iter extraire yields rb ;
end
```

init : effectue l'initialisation (assignation de fichier par exemple) ;

ajouter : effectue une adjonction de l'objet passé en paramètre ;

supprimer : effectue une suppression de l'objet dont la clé est donnée en paramètre ;

modifier : effectue un remplacement de l'objet dont la clé est donnée en paramètre, par l'objet passé en paramètre ;

accès : effectue l'accès à un objet à partir d'une clé ;

existe : teste l'existence d'un objet dans la base ;

extraire : itérateur permettant de parcourir une relation.

Exemple : pour les relations de la base restaurant nous avons les r-machines : (en ATM).

```
machine salles
    proc init ;
    proc ajouter (salle) ;
```

```
    proc  supprimer (integer) ;
    proc  modifier (integer, salle) ;
    proc  accès (integer) ⟶ salle, boolean ;
    function  existe (salle) returns boolean ;
    iter  extraire  yields  salle ;
end

machine  plats ;
    proc  init ;
    proc  ajouter  (plat) ;
    proc  supprimer (integer) ;
    proc  modifier (integer, plat) ;
    proc  accès (integer)⟶ plat, boolean ;
    function existe (plat)  returns boolean ;
    iter  extraire  yields  plat ;
end

machine  menus ;
      .
      .
      .
    proc  supprimer  (integer, integer) ;
    proc  modifier  (integer, integer, menu) ;
    proc  accès  (integer, integer)⟶ plat, boolean ;
      .
      .
      .
end
```

V.III.3 Capsule et module

Exemple de capsule correspondant à un t-type r b


```
Capsule rb for  rb ;
  rep
        a1  :  d1 ;
        a2  :  d2 ;
        .
        .
        an  :  dn ;
  end
build  créer (v a1 : d1 , ..., v an : dn) ;
  begin
        a1 := v a1 ; ... ; an := v an ;
  end
function  a1  returns  d1 ;
begin  return  a1 ; end
  .
  .
  .
```

Dans la zone rep end on déclare des variables (autant que de fonctions définies sur le t-type) du type des domaines. Le nom des variables est (le même que celui des fonctions) celui des attributs. Il n'y a pas d'ambiguïté quant à TYP - R.

Pour le détail des capsules voir en annexe B.

Exemple de module correspondant à une r-machine et utilisant le t-type rb.

```
Module  rbs  for  rbs ;
    ressource
        f r b s :  f s e q ;
    end
proc init ;
    # initialisation :  assignation d'un fichier
```

```
proc ajouter (vrb : rb)
    var
     :
    end
    begin
     :
    end
proc  supprimer (va_1, ..., va_k : d_k) ;
       :
proc  modifier (va_1, ..., va_k : d_k, vrb : rb) ;
       :
proc  accès ... ;
      :
:
end
proc  existe (vrb : rb) returns  boolean
   var  t : text ;   b : boolean ;
        v : rb ;
   end
   begin
   # ouverture du fichier
        ∂ frbs open ;
   # accès au premier enregistrement
        ∂ frbs  get → t, b
   # parcours du fichier jusqu'à l'enregistrement recherché
        while b
            # code de conversion de la variable enregistrement
            # t en la variable v du t-type rb
             :
```



```
            if ∂ vrb  equal (v)
            then
                return  true ;
            endif ;
            ∂ frbs  get ⟶ t, b ;
        endwhile ;  return false ;
    end
Iter  extraire yields rb ;
   var  vrb ; rb ; ... end
   begin
     :
     yield  vrb ;
     :
end
```

Nous avons illustré un exemple d'opération (ici existe). Il faut noter l'utilisation de la fonction equal du t-type rb et la nécessité de convertir les objets de ce t-type en texte (car en ATM, on n'a que des fichiers de text) avant écriture sur fichier et inversement, après lecture.

Pour les détails voir annexe B.

V.III.4 Exécution de requêtes

V.III.4.1 Langage d'interrogation

Le langage d'interrogation utilise les opérateurs PROJECT, SELECT et JOIN qui nous ont semblé caractéristiques dans l'algèbre relationnelle.

Le langage offre la particularité d'exprimer des requêtes 'multibase',

Exemple : sur le schéma de la fig. V.1

a - PROJECT (JOIN (RESTAURANTS . SALLES, CINEMAS . SALLES,
RUE = RUE), RUE, NOMC, NOMR) ;

Cette requête s'adresse aux deux bases CINEMAS et RESTAURANTS pour chercher les salles de cinéma et de restaurant situées dans une même rue,

Ou des requêtes 'monobase'

Exemple :

b - PROJECT (JOIN (JOIN (SELECT (PLATS, NOMP = p),
MENUS, NUMP = NUMP),
RESTAURANT . SALLES, NUMR = NUMR), NOMR) ;

Cette requête s'adresse à la base RESTAURANTS et permet de sortir les noms de restaurants qui servent un plat p au menu. Notons que les relations MENUS et PLATS n'ont pas besoin d'être préfixées par le nom de la base.

A) Exécution d'une requête monobase :

Examinons la requête ci-dessus à l'exécution. Une première solution consiste à exécuter successivement chaque opérateur, du plus interne vers le plus externe, et créer une relation intermédiaire à chaque niveau.

Ainsi, on crée d'abord une relation, résultat de SELECT, qui contient les plats de nom p. A partir de cette relation, et de la relation MENUS, on effectue une jointure dont le résultat servira pour une autre jointure avec la relation SALLES. Ensuite, on projette le résultat

obtenu sur la colonne NOMR.

Cette solution qui a le mérite d'être simple, est assez longue à mettre en œuvre, et nécessite une mémoire secondaire de travail pour le stockage des relations intermédiaires, donc de l'espace et beaucoup d'entrées/sorties. Le temps de réponse peut être alors assez long.

On peut éviter de stocker les relations intermédiaires en les représentant par des pointeurs vers les valeurs contenues dans les relations de base, nécessaires à l'exécution d'un opérateur [NAS 81]. Par exemple le premier opérateur JOIN opère sur la relation de base MENUS, et sur des pointeurs vers les plats de la relation PLATS qui s'appellent p.

Une autre solution consiste à commencer l'exécution de cet opérateur JOIN dès qu'il a un tuple produit par SELECT, c'est-à-dire dès qu'on trouve un plat ayant pour nom p.

Cette solution a le mérite de permettre une exécution parallèle car pendant que JOIN traite un tuple produit par SELECT, ce dernier peut chercher le suivant. Mais cette façon de procéder, si elle est rapide, ne permet pas l'élimination des duplications.

Smith [SMI 75] a proposé une solution intermédiaire dite en 'pipeline' qui consiste à lancer en parallèle l'exécution de certains opérateurs dès qu'ils ont un nombre suffisant de tuples en entrée. Certains même s'exécutent dès qu'un tuple est disponible, et d'autres peuvent nécessiter la disponibilité de toute une relation intermédiaire. Dans la plupart des cas, les duplications sont ainsi évitées et le volume de mémoire secondaire nécessaire est réduit.

Solution TYP - R :

La solution choisie dans notre cas est une solution basée sur l'idée d'itérateurs abstraits. En effet, la possibilité étant offerte de parcourir une relation tuple par tuple et d'accèder par des fonctions d'accès aux différents composants d'un tuple, il est naturel d'interpréter les opérations de manipulation de relation par des instructions d'itérations.

Les algorithmes suivants illustrent cela :

1) PROJECT (SALLES, NOMR, RUE)

```
foreach s in $ Salles extraire
   then
      write (∂ s nomr, ∂ s rue)
end
```

2) SELECT (SALLES, NOMR = chine) ;

```
foreach s in $ salles extraire
    where ( ∂ S nomr = 'chine')
    then
    write ( ∂ s nomr, ∂ s numr, ∂ s rue, ∂ s type,
            ∂ s tel) ;
endforec ;
```

3) JOIN (SALLES, MENUS, NUMR = NUMR) ;

```
foreach s in $ Salles extraire
    then
        foreach m in $ menus extraire
        where ( ∂ s numr = ∂ m numr)
        then
         write ( ∂ s numr, ..., ∂ m numc, ...) ;
        endforec ;
endforec ;
```


Du point de vue général, et pour permettre la manipulation tuple par tuple des relations intermédiaires (résultat d'un opérateur), il faut considérer ces dernières comme des séquences de tuples. Pour cela, nous avons défini des itérateurs (un pour chaque opérateur), qui rendent par une instruction yield, un tuple à la fois le résultat de l'opérateur considéré. Ce résultat est traité dans une instruction foreach, par l'opérateur suivant, ou pour être délivré à l'utilisateur sur le terminal.

Pour décrire les algorithmes de ces itérateurs, nous utiliserons les conventions d'écriture suivantes :

- R et R' sont des relations (de base ou intermédiaires) définies comme ceci : R $(A_1, \ldots, A_n)$ et R' $(A'_1, \ldots, A'_m)$;
- t représente une variable du type prédéfini tuple.
- cs, cj , cp , représentent des variables des types prédéfinis : critère-de-sélection, critère-de-jointure, critère-de-projection et qui correspondent repsectivement aux formulations $\{A_i \ \theta \ a_i\}$, $\{A_i \ \theta \ A'_j\}$, $\{A_{i_1}, A_{i_2}, \ldots, A_{i_p}\}$ où $\theta \in \{ =, \neq, \leq, \ldots \}$.
- read (cs), read (cj), read (cp) sont les opérations de lecture de ces critères dans la zone contenant le texte de la requête.
- rbs et rbs' sont les r-machines (cf. V.III.2) des relations R et R'.
- rb et rb' sont les t-types (cf. V.II.2) de ces relations.
- vrb et vrb' sont des variables de ces types.
- tup (vrb) représente l'opération de conversion d'une variable, du type rb en type tuple.

  tup : rb ⟶ tuple.

- les fonctions suivantes (prédéfinies dans TYP - R) :

  select ? : tuple X critère-de-sélection ⟶ boolean

  project : tuple X critère-de-projection ⟶ tuple

join ? : tuple X critère-de-jointure X tuple ⟶ boolean

Concat : tuple X tuple ⟶ tuple

représentent respectivement :

la sélection ou non d'un tuple selon un critère,

la projection d'un tuple sur un autre, selon un critère

la possibilité de jointure de deux tuples, et

la concaténation de deux tuples.

Ainsi donc :

- PROJECT (R, $\{A_{i_1}, \ldots, A_{i_p}\}$)

s'interprête par :

```
foreach vrb in < R >
  then
      t := project (tup (vrb), read (cp)) ;
      yield t
end
```

- SELECT (R, $\{A_i \ \theta \ a_i\}$)

```
foreach vrb in < R >
   where
        Select ? (tup (vrb), read (cs))
   then
        yield tup (vrb) ;
   end
```

- JOIN (R, R', $\{A_i \ \theta A'_j\}$)

```
foreach vrb in < R >
  then foreach vrb' in < R' >
         where JOIN ? (tup(vrb), tup (vrb'), read (cj))
         then
```



```
              t := concat (tup (vrb), tup (vrb')) ;
         yield t ;
         end
end
```

< R > représente

```
SI RELATION DE BASE (R)
ALORS $ rbs extraire
SINON
     Si JOIN (R)
     ALORS $ oprel join
     SINON Si PROJECT (R)
           ALORS $ oprel project
           SINON Si SELECT (R)
                 ALORS $ oprel Select
                 FSI
           FSI
     FSI
FSI
```

En effet, les algorithmes ci-dessus qui réalisent les itérateurs interprétant les opérateurs relationnels sont groupés dans un module interfacé par une machine oprel :

```
Machine oprel ;
   Iter join yields tuple ;
   Iter Project yields tuple ;
   Iter Select yields tuple ;
```

L'application de la fonction tup (vrb) n'est alors nécessaire que si vrb est une variable du t-type d'une relation (cas où R est une relation de base).

Performance et optimisation :

La méthode d'interprétation d'une requête proposée ici, appelle quelques commentaires :

- On ne peut pas éliminer les duplications par le fait qu'on travaille sur des séquences de tuples, et non sur l'ensemble complet. En contrepartie, on n'a pas de relations intermédiaires à mémoriser, sauf pour des raisons d'optimisation (cf. ci-dessous). Il en résulte une grande économie d'espace mémoire et une relative rapidité dans le temps de réponse.

- JOIN est un opérateur binaire qui nécessite le parcours du 2ème opérande pour chaque tuple du 1er. Si ce deuxième opérande est un opérateur cela implique son exécution autant de fois qu'il y a des tuples dans le premier. Pour cela, il a été nécessaire de sauvegarder le résultat du deuxième opérande après une 1ère exécution.

Exemple :

JOIN (R, SELECT (R', ...)..).

Le SELECT est exécuté pour le premier tuple de R, son résultat est sauvegardé pour les tuples suivants de R.

Conclusion :

L'idée générale d'implantation d'un langage relationnel utilisé dans TYP - R est de ramener la manipulation de relation à un ensemble d'opérations définies sur des tuples et à des itérations.

Ces opérations sont suffisantes pour implanter n'importe quel langage de puissance égale à celle de l'algèbre relationnelle et s'avèrent nécessaires.

Une démonstration complète de ceci se trouve dans [BEC 80].

B) Exécution d'une requête multibase

Dans une base toutes les relations ont des noms différents. Deux relations peuvent avoir le même nom dans deux bases différentes.

En TYP - R, les relations ont un nom interne différent sur toute la multibase. Ce nom est celui de la r-machine de la relation.

Pourquoi ces noms sont différents ?

Le nom d'un t-type est celui de la relation.
Le nom d'une r-machine est celui du t-type auquel est rajouté (concaténé) la lettre 'S'.

Si on préfixe ces noms par celui de la base à laquelle appartient la relation, on a des noms tous différents. On peut considérer (alors du point de vue interne), que toutes les relations sont dans une même base, et tout se passe comme si une requête est adressée à cette base.

Dans ce cas l'algorithme d'exécution d'une requête multibase est le même que celui d'une requête monobase.

V.III.4.2 Langage de mise à jour

Ce langage concerne les insertions, modifications ou suppressions de tuples dans une relation. Ces opérations se font un tuple à la fois, et ne concernent qu'une seule relation.

1) Insertion

SALLES

NUMR	NOMF	RUE	TYPE	TEL
R1	CUNY	A. France	LORRAIN	123 45 67
R2	GOELAND	des Ponts	MARIN	234 56 78
R3	PEPLUM	Gembloux	ITALIEN	345 67 89
R4	CHINE	Hérè	CHINOIS	456 78 90

PLATS

NUMP	NOMP	NCAL
P1	CHOUCROUTE	3500
P2	ENTRECOTE	4000
P3	RIZ	4500
P4	PIZZA	3900
P5	PAELLA	4000

MENUS

NUMR	NUMP	PRIX
R1	P1	50.
R1	P2	40.
R2	P5	55.
R3	P4	45.

Fig. V.4 - Figure de référence pour les exemples qui suivent.

I1 - Ajouter la salle de restaurant (R5, OS-A-MOELLE, MARECHAUX, ABATTOIRS, ...).

INSERT (SALLES : NUMR := R5, NOMP := OS-A-MOELLE, RUE := MARECHAUX, TYPE := ABATTOIRS, TEL := 135 67 89) ;

tous les attributs doivent apparaître avec une valeur. En TYP - R la valeur nil ('null value') n'existe pas[1].

Si la salle R5 existait déjà dans la base, la requête serait rejetée.

I2 - Ajouter le plat (P6, BROCHETTES, 4600)

INSERT (PLATS : NUMP := P6, NOMP := BROCHETTES, NCAL := 4600) ;

I3 - Ajouter le menu (R3, P2, 50)

INSERT (MENUS : NUMR := R3, NUMP := P2, PRIX := 50) ;

La relation MENUS est liée aux autres relations SALLES et PLATS par une contrainte référentielle.

Les requêtes suivantes seront rejetées :

I4 - INSERT (MENUS : NUMR := R8, NUMP := P5, PRIX 40) ;

En effet, la valeur R8 ne réfère aucune ligne dans la relation SALLES. La base serait dans un état incohérent (un menu qui n'est servi par aucune salle de restaurant).

Il faut au préalable insérer une ligne dans SALLES avec R8.

Il en est de même pour la requête :

I5 - INSERT (MENUS : NUMR := R4, NUMP := P7, PRIX := ...)

Le plat P7 n'est pas répertorié dans la base.

Ces contraintes d'intégrité référentielle sont celles déduites des déclarations de clés secondaires (NUMR et NUMP dans la relation MENUS), et de clés primaires (NUMR dans SALLES, NUMP dans PLATS).

(1) On a la possibilité de l'introduire en définissant un type abstrait valeur avec une opération nil.

2) Suppression

S1 - Supprimer le restaurant R4.

DELETE (SALLES : NUMR = R4) ;

La suppression se fait à partir d'une clé.
Si cette dernière n'est pas dans la base, la requête est sans effet.

S2 - Supprimer le menu (R3, P4)

DELETE (MENUS : NUMR = R3, NUMP = P4) ;

La contrainte d'intégrité référentielle peut être violée par des suppressions aussi.

La requête :

S3 - DELETE (SALLES : NUMR = R3)

met la base dans un état incohérent, vu que la valeur R3 figure dans la relation MENUS. Cette requête doit être rejetée.

Il existe une solution qui, dans un cas pareil, fait supprimer "implicitement" la (ou les) ligne (s) de la relation MENUS qui ont la valeur R3. Cette solution à notre avis est à éviter, car une erreur de la part de l'utilisateur (écriture de R3 au lieu de R4) peut avoir des conséquences néfastes. Nous lui préférons la solution qui consiste à écrire explicitement les requêtes qu'il faut :

DELETE (MENUS : NUMR = R3 , NUMP = P4) ;
suivi de DELETE (SALLES : NUMR = R3) ;

3) Modification

M1 - modifier la rue et le téléphone du restaurant R4.
en (NHERE et 328 93 93)

MODIFY (SALLES : NUMR = R4 ; RUE := NHERE, TEL := 328 93 93) ;

La modification se fait en donnant la clé de la ligne à modifier, suivie des nouvelles valeurs des attributs à corriger.

On peut modifier un ou plusieurs de ces attributs.

La modification directe d'une clé n'est pas permise en TYP - R. Modifier R3 en R5 dans la relation SALLES pose un problème car R3 est référencé dans la relation MENUS, et il ne faut donc pas le supprimer ou le changer.

A l'exception de SYSTEM R, la plupart des autres systèmes interdisent la modification d'une clé.

Remarque : L'exemple de la base RESTAURANTS est particulier du fait que les seules clés secondaires qui existent : NUMR # et NUMP # dans la relation MENUS, sont composantes de la clé primaire de cette relation. Si on considère l'exemple suivant :

REL1 (CP #, CS, AUTRE1)
REL2 (CS #, AUTRE2) CS # défini sur le même domaine que CP # ;

quant on modifie dans REL1 une valeur de CS, il faut vérifier que la nouvelle valeur donnée existe dans la relation REL2 pour CS #.

Donc, dans notre exemple de la base RESTAURANTS ce problème ne se pose pas puisqu'on ne modifie pas une clé.

Mais par contre, on peut remplacer une modification de clé par une suppression suivie d'une adjonction. Si toutefois aucune contrainte n'est violée :

a) pour modifier R3 en R5 dans la relation SALLES (fig. V.4)
on écrirait DELETE (SALLES, NUMR = R3)
suivi de INSERT (SALLES : NUMF := R5, NOMF = PEPLUM...)
ce qui sera rejeté puisque R3 ne doit pas être supprimé.

b) Pour modifier P3 dans la relation PLATS en P6, on écrirait :

DELETE (PLATS, NUMP = P3)

INSERT (PLATS, NUMP := P6, NOMP := PIZZA ...)

ce qui sera accepté puisque P3 n'existe pas dans MENUS.

RECAPITULATION

opération	signification	restrictions
INSERT (R, t)	- Ajouter le tuple t à R	- Tous les composants de t ont une valeur, et la clé de t n'existe pas déjà dans R. - Tous les tuples référencés par t (par leur clé) existent.
DELETE (R, k)	- Supprimer de R le tuple de clé k	- Le tuple de clé k n'est pas référencé par un autre tuple dans une relation.
MODIFY (R, k, t)	- Modifier le tuple de clé k dans R, selon le tuple t	- des composants constituants la clé ne sont pas modifiés. - Tous les tuples référencés par t existent.

Fig. V - 5 Tableau récapitulatif du langage de mise à jour.

4) Interprétation du langage de mise à jour :

R est la relation $(A_1, A_2, \ldots, A_n)$ de la base B.
Les k premiers attributs forment la clé.

- INSERT $(R : A_1 := v_1, A_2 := v_2, \ldots, A_n := v_n)$;

s'interprête par :

$@$ vrb créer $(v_1, v_2, \ldots, v_n)$;
if not ($ rbs existe (vrb)) then $ rbs ajouter (vrb) : endif ;

en désignant par vrb une variable du type rb, t-type de la relation R, et par rbs la r-machine de cette relation.

- DELETE $(R : A_1 = v_1, A_2 = v_2, \ldots, A_k = v_k)$;

s'interprête par :

$ rbs supprimer $(v_1, v_2, \ldots, v_k)$

- MODIFY $(R : A_1 = v_1, A_2 = v_2, \ldots, A_k = v_k, A_{i_1} = v_{i_1}, \ldots, A_{i_p} = v_{i_p})$

$a \leq p \leq n - k \qquad k < i_j \leq n$

s'interprête par

$ rbs accès $(v_1, v_2, \ldots, v_k) \longrightarrow$ vrb, b
if b
then
$@$ vrbl créer $(v_1, v_2, \ldots, v_k, < v_j >_{j = k+1, n})$;
$@$ rbs modifier $(v_1, v_2, \ldots, v_k,$ vrbl) ;
endif

où

$< v_j >$ = (SI $j = i_\ell$ ALORS v_{i_ℓ} SINON $@$ vrb a_j)

$\ell \in [1, p]$.

V.III.5 Contrôles d'intégrité

Nous ne nous intéressons pas aux contraintes mettant en cause plusieurs bases. Nous supposons qu'il n'y a pas de lien sémantiques, ou de dépendances fonctionnelles entre attributs appartenant à 2 relations dans deux bases différentes.

Nous considérons donc le cas d'une seule base. On peut envisager trois types de contraintes :

V.III.5.1 Différents types de contraintes :

1) Contrainte d'entité : une clé doit toujours avoir une valeur. C'est le cas dans TYP R.

2) Contrainte référentielle : celle du type discuté ci-dessus, c'est-à-dire, si A et A' sont respectivement des attributs de R et R', deux relations d'une même base tels que :

   - A et A' sont définies sur un même domaine.
   - A est clé primaire pour R, A' clé secondaire pour R'

alors :

$\{\forall x' \in R', \exists n \in R : x' . A' = x . A\}$ R peut être égale à R'.

3) Autres contraintes :

Exemple : un menu a toujours un prix inférieur à 100 F.

Les salles de restaurant de la rue TELLE sont toutes de type CHINOIS.

Nous allons montrer sur un exemple comment vérifier les contraintes référentielles dans la base restaurants lors de l'insertion, ou suppression de tuples.

Reprenons les exemples de V.III.4.2.

I4 - INSERT (MENUS : NUMR := R8, NUMP := P5, PRIX := 40) ; il faut vérifier que R8 existe dans la relation SALLES, et que P5 existe dans la relation PLATS.

La séquence suivante permet cela en rendant un boolean disant si oui ou non l'insertion est conforme :

```
first s in $ salles extraire
    where ∂ numr = R8
    then
        yes := true ;
    else yes := false ;
endfirst ;
if yes
then first p in $ plats extraire
    where ∂ p nump = P5
      then
          return true ;
      else
          return false ;
    endfirst
else
    return false ;
endif ;
```

$(\exists s \in SALLES, \exists p \in PLATS : s . NUMR = R8 \wedge p . NUMP = P2)$

S3 - DELETE (SALLES : NUMR = R3).

Il faut vérifier que R3 ne figure pas dans la relation MENUS ($\nexists m \in MENUS : m . NUMR = R3$) :

```
first m in $ menus extraire
    where ∂ m numr = R3
        then
            yes := false ;
        else
            yes := true ;
endfirst ;
return yes ;
```

V.III.5.2 Modules de contrôle

En TYP R, nous avons la possibilité de définir un module de contrôle pour chaque relation, où seront décrits les algorithmes comme ceux ci-dessus qui vérifient des propriétés à tester avant chaque mise à jour.

L'interface de ce module sera appelé la C-machine de la relation.

Exemple : pour la relation SALLES nous aurons la C-machine suivante :

```
machine ctrlsalles ;
    proc ctrlajout (salle)⟶ boolean ;
    proc ctrlsupp (salle)⟶ boolean ;
    proc ctrlmod (integer, salle)⟶ boolean ;
end
```

ces opérations contrôlent respectivement si l'insertion, la suppression ou la modification dans la relation SALLES est possible ou non.

Le mécanisme de connexion de TYP, offre la possibilité de faire cela :

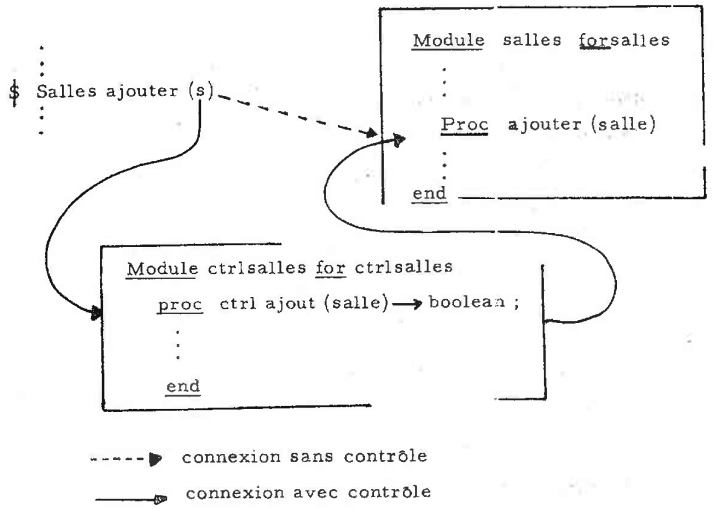


Fig. V.6

Les techniques de cette connexion sont largement détaillées dans [HEN 80] .

Les modules de contrôle sont écrits par l'administrateur de la base ou par la personne responsable de sa cohérence. Il est alors, le seul habilité à le modifier.

Il peut en écrire plusieurs pour une même relation. Chacun sera associé à une catégorie d'utilisateurs selon leurs droits.

Pour des contraintes du type, un menu doit toujours avoir un prix inférieur à 100 F, ou, tous les restaurants de la rue TELLE sont de type CHINOIS, il est possible de les vérifier dès la création d'objet. (opération créer d'un t-type) ; Cependant, il est préférable de les faire vérifier par une C-machine pour ne pas lier le t-type à cette contrainte.

Il est donc très facile de modifier à tout moment les contraintes d'intégrité de la base, puisque cela ne remet en cause que les modules de contrôle.

V.III.6 Dictionnaire de la multibase

Ce dictionnaire a pour rôle de gérer la description de la multibase. Il contient les noms des bases, et des relations de chaque base. Pour chaque relation d'une base donnée, il permet d'en connaître la liste d'attributs, en particulier ceux constituants la clé.

Dans l'analyse d'une requête, il fournit les informations nécessaires à la vérification de la correction de celle-ci.

V.III.7 Conclusion récapitulative

Nous sommes maintenant en mesure d'expliciter le contenu et le principe de fonctionnement du SMB, (Système Multibase annoncé au chapitre II) suivant :

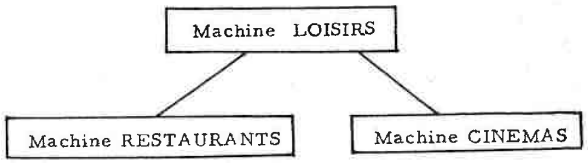


Fig. V.7 SMB correspondant à la maquette TYP-R.

Machine LOISIRS :

Elle est constituée de l'ensemble des modules qui concourent à la gestion de la multibase. On distinguera principalement les modules suivants :

- l'interface de dialogue avec l'utilisateur, et qui permet notamment de gérer les différentes vues ;
- l'analyseur de requêtes(1)
- l'interprêteur de requêtes(1)
- le gestionnaire du dictionnaire de la multibase ;
- le contrôleur des liens d'intégrité inter-base (n'est pas défini dans la maquette).

Appelons SGMB (Système de Gestion MultiBase) l'ensemble de ces modules.

Machine RESTAURANTS :

Elle est constituée de l'ensemble des modules qui définissent les relations de la base RESTAURANTS :

- module décrit en ATM, qui correspond à la r-machine d'une relation et qui réalise les opérations d'accès aux tuples d'une relation (Ajouter, supprimer, etc...), et de parcours d'une relation (l'itérateur).

- module qui correspond à la c-machine d'une relation et qui contrôle les règles d'intégrité sur cette relation (n'est pas implanté).

Nous négligeons ce module dans la suite de l'exposé par souci de simplification.

La machine CINEMAS est analogue à RESTAURANTS.

---

(1) Décomposé en plusieurs modules décrits en ATM.

Dans chacune des machines nous avons sous-entendu l'existence des types de données utilisés, par exemple le t-type d'une relation dans la machine RESTAURANTS, et aussi les types de bases ou prédéfinis (text, fichier, etc...).

On peut établir la figure suivante qui reprend la (fig. V.7)

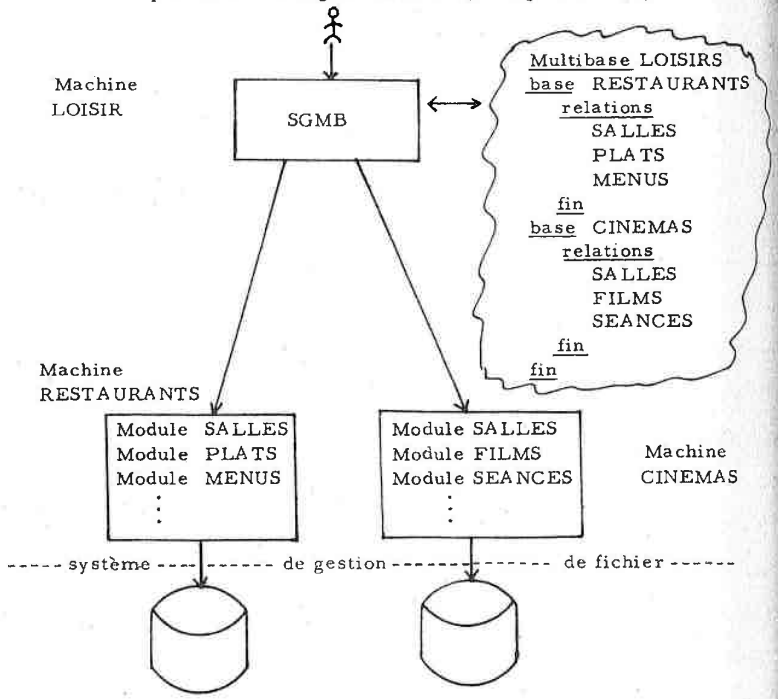
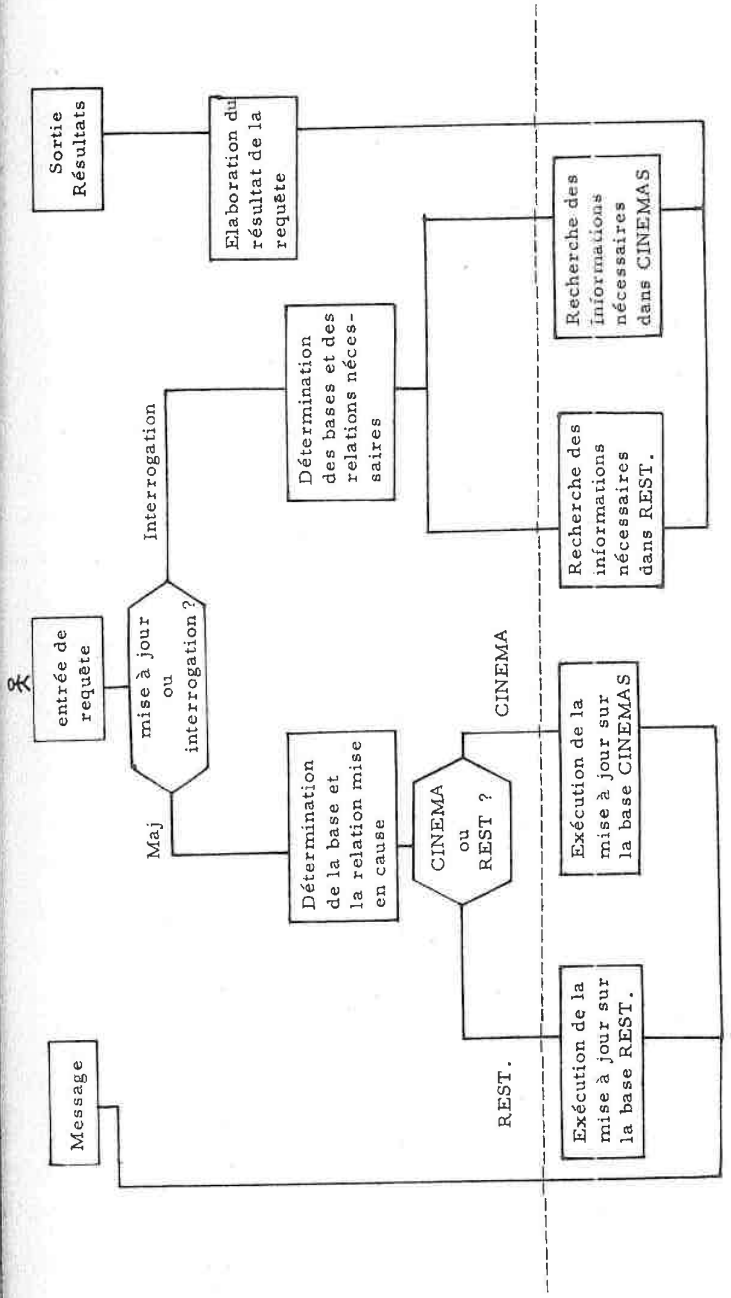


Fig. V.8 - Architecture du SMB

Le schéma de la page suivante illustre le principe de fonctionnement du SMB.



Au dessus du trait pointillé il y a la machine LOISIRS
En dessous, il y a les machines RESTAURANTS et CINEMAS.

Fig. V.9 - Schéma de principe de fonctionnement du SMB.

CONCLUSION

TYP - R est un système de bases de données relationnelles et réparties. L'approche suivie considère une base de données répartie comme une multibase, et permet ainsi de la construire progressivement.

L'analogie entre l'idée d'indépendance données - programmes utilisée dans le domaine des bases de données, l'idée de type abstrait de données utilisée dans les langages de programmation, et l'idée de modularité utilisée dans le domaine de conception de systèmes logiciels structurés, nous a amené à réaliser TYP - R sous le système TYP. Ce qui a contribué à la simplification de son développement.

Les concepts du système TYP mis en œuvre dans la réalisation de TYP-R, nous incitent à croire qu'il est possible de développer TYP-R aussi bien avec des langages comme CLU ou ADA qui seront très utilisés dans l'avenir.

En ce qui concerne les applications pratiques de TYP-R, on peut envisager des perspectives intéressantes, et qui méritent un approfondissement :

1 - TYP-R pourrait servir à réaliser une "coopération" entre plusieurs bases de données relationnelles existantes et gérées par leur propre SGBD.

L'idée consiste à considérer un SGBD dans ce cas, comme un serveur de relations, et à l'assimiler à un "système de gestion de fichier" très évolué qui, au lieu de fournir des opérateurs de lecture/écriture d'enregistrements dans un fichier, fournit des opérations de lecture/écriture de tuples dans une relation. Ce que tout SGBD relationnel pourra permettre.

D'un autre côté, même si toutes les bases ne sont pas relationnelles, il suffirait de définir en amont de chaque base décrite dans un autre modèle, un interface qui donne une vue relationnelle sur celle-ci. La réalisation de cet interface peut ne pas être difficile, suivant les fonctions qu'un SMB demande à un SGBD (serveur de relations). Il suffit, une fois définies les correspondances dans la structure relationnelle d'un objet décrit selon une autre structure (IMS, SOCRATE, ...), d'interprêter des opérations de lectures/écritures en fonction de ces correspondances[1].

2 - Il nous semble aussi que le principe même de multibase et de SMB pourra s'appliquer aux bases de données classiques.

On pourra considérer, comme leurs propres utilisateurs du reste, que les différentes vues sur une base classique, sont autant de bases différentes, mais ayant des liens entre-elles, et qui constituent une multibase.

Ainsi, au lieu de modéliser un univers, qui peut être complexe, par une seule base et, une partie de cet univers (ou sous-univers) par une vue sur cette base, on pourra constituer une base, conçue spécialement pour chaque sous-univers, et qui pourra être simple et donc facile à implanter. L'ensemble de ces bases constituant une multibase, correspondra alors à l'univers considéré.

Cela est particulièrement intéressant, car les problèmes de vues, qui se posent avec acuité aux concepteurs de bases de données, peuvent ainsi être évités. De même, une telle multibase pourra aussi contenir toutes les données concernant une grande organisation, et peut facilement s'étendre pour en contenir d'autres en fonction du développement de cette organisation.

---

(1) Les problèmes de traduction entre langages de manipulation dans les différents modèles connus, sont abordés de façon générale dans [WON 79], [KAT 80 - 82].

Ce sont là des idées qui, à notre avis, méritent d'être étudiées.

Pour conclure, nous considérons que le travail présenté dans cette thèse est d'abord original, et ensuite très riche d'enseignements. D'une part, les concepts de multibase et de systèmes multibases sont très récents, et d'autre part, l'application aux bases de données des idées développées dans le domaine du génie logiciel est maintenant couramment admise.

Nous restons néanmoins fermement convaincus que beaucoup de points abordés dans cette thèse sont très importants, et nécessitent une étude plus générale, et plus approfondie.

ANNEXE A

EXEMPLE D'EXECUTION

de T Y P - R

```
 VOUS ETES CONNECTE A LA MULTIBASE:

MULTIBASE LOISIR

   DATABASE RESTAURANT
        SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
        PLATS ( NUMP#, NOMP, NCAL )
        MENUS ( NUMR#,NUMP#, PRIX)
   END DATABASE

   DATABASE CINEMA
        SALLES ( NUMC#, NOMC, RUE, TEL)
        FILMS ( NUMF#, NOMF, GENRE )
        SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
   END DATABASE

END MULTIBASE


VOUS POUVEZ TRAVAILLER SUR LES BASES DE VOTRE CHOIX.
INDIQUER LE NOM DE LA BASE
 OU FAIRE RETOUR CHARIOT SI FINI.
?restaurant


INDIQUER LE NOM DE LA BASE
 OU FAIRE RETOUR CHARIOT SI FINI.
?
```

(CR)

Manipulation de la base : RESTAURANT

```
      UTILISER LE SCHEMA SUIVANT :

     MULTIBASE

        DATABASE RESTAURANT
            SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
            PLATS ( NUMP#, NOMP, NCAL )
            MENUS ( NUMR#,NUMP#, PRIX)
        END DATABASE

     END MULTIBASE


     INDIQUER LE TRAITEMENT PAR
     QUERY  : POUR L INTERROGATION,
     UPDATE  : POUR  LA M-A-J:
?query

     SOUMETTRE UNE REQUETE OU
     FAIRE RETOUR CHARIOT.
       VOULEZ-VOUS LA SYNTAXE?(O/N)
?o
      LES OPERATEURS SONT:
                PROJECT(<REL>, AT1,...,ATN);
                JOIN    (<REL>,<REL>, AT <?> AT);
                SELECT (<REL>, AT <?> ARG);
       <REL> PEUT ETRE UN OPERATEUR, OU NOM DE RELATION
       EVENTUELLEMENT PREFIXE PAR NOM DE BASE .
       <?> A PRENDRE DANS { = , < , <= , > , >= }

?project(salles, numr,nomr,type,rue,tel);
```



```
PROJECT(SALLES, NUMR,NOMR,TYPE,RUE,TEL);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NUMR            NOMR            TYPE            RUE             TEL

      1               GOELAND         MARIN           DES-PONTS       3351725
      2               CORDELIERS      ROTISSERIE      BENIT           3354732
      3               MANDARIN        CHINOIS         PL-CROIX-BOURG  3402785
      4               MONEDA          PIZZERIA        COMMANDERIE     3404242
      5               CAMARGUE        PIZZERIA        ST-DIZIER       3353117
      6               DES-AMIS        SPECIALITE      4-EGLISES       3355011
      7               ALADIN          MAROCAIN        4-EGLISES       3322132


      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?p

?project(plats,nump,nomp,ncal);


PROJECT(PLATS,NUMP,NOMP,NCAL);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n
```

NUMP	NOMP	NCAL
1	CHOUCROUTE	4000
2	COUSCOUS	4005
4	PAELA	4500
6	PIZZA	3400
8	HAMBURGER	2000
9	BROCHETTES	3000

```
      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?project(menus,numr,nump,prix);

PROJECT(MENUS,NUMR,NUMP,PRIX);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n
```

NUMR	NUMP	PRIX
2	9	55
4	6	28
5	6	30
6	2	50
6	4	62


```
     SOUMETTRE UNE REQUETE OU
     FAIRE RETOUR CHARIOT.
       VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?(CR)

      UTILISER LE SCHEMA SUIVANT :

     MULTIBASE

        DATABASE RESTAURANT
             SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
             PLATS ( NUMP#, NOMP, NCAL )
             MENUS ( NUMR#,NUMP#, PRIX)
        END DATABASE

     END MULTIBASE


      INDIQUER LE TRAITEMENT PAR
      QUERY  : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?update


       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
         VOULEZ-VOUS LA SYNTAXE?(O/N)
?o
```

```
LES COMMANDES SONT:
  INSERT(NOM DE RELATION,<CRITERE SOURCE>)
  DELETE(NOM DE RELATION,<CRITERE CIBLE>)
  UPDATE(NOM DE RELATION,<CRITERE CIBLE>:
                         <CRITERE SOURCE>)

   <CRITERE SOURCE> EST UNE LISTE DE
              "ATTRIBUT:=ARGUMENT".
   <CRITERE CIBLE> EST UNE LISTE DE
              "ATTRIBUT-CLE=ARGUMENT"


?insert(plats, nump:=5, nomp:=canard-laque, ncal:= 4300);

INSERT(PLATS, NUMP:=5, NOMP:=CANARD-LAQUE, NCAL:= 4300);

VOULEZ VOUS CHANGER ? (O/N)

?n


      NUMP            NOMP            NCAL
      5               CANARD-LAQUE    4300
       TUPLE AJOUTE:
      NUMP            NOMP            NCAL
      5               CANARD-LAQUE    4300
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
```



```
 n

?insert(menus, numr:=3, nump:=15, prix:= 55);

INSERT(MENUS, NUMR:=3, NUMP:=15, PRIX:= 55);

VOULEZ VOUS CHANGER ? (O/N)

?n


      NUMR            NUMP            PRIX
      3               15              55
VIOLATION INTEGRITE REFERENTIELLE
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?insert(menus, numr:=3, nump:=5, prix:=55);

INSERT(MENUS, NUMR:=3, NUMP:=5, PRIX:=55);

VOULEZ VOUS CHANGER ? (O/N)

?n
```

$\nexists\ p \in PLATS : p.NUMP =$

```
       NUMR              NUMP              PRIX
       3                 5                 55

        TUPLE AJOUTE:
       NUMR              NUMP              PRIX
       3                 5                 55
        SOUMETTRE UNE MISE A JOUR OU
        FAIRE RETOUR CHARIOT.
         VOULEZ-VOUS LA SYNTAXE?(O/N)
?
 n
?insert(plats, nump:=10, nomp:=pastilla, pcal:=4500);

INSERT(PLATS, NUMP:=10, NOMP:=PASTILLA, NCAL:=4500);

VOULEZ VOUS CHANGER ? (O/N)

?n


       NUMP              NOMP              NCAL
       10                PASTILLA          4500
        TUPLE AJOUTE:
       NUMP              NOMP              NCAL
       10                PASTILLA          4500
        SOUMETTRE UNE MISE A JOUR OU
        FAIRE RETOUR CHARIOT.
         VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
```



```
?insert(menus, numr:=7, nump:=10, prix:=110);

INSERT(MENUS, NUMR:=7, NUMP:=10, PRIX:=110);

VOULEZ VOUS CHANGER ? (O/N)

?n


      NUMR              NUMP              PRIX
      7                 10                110

       TUPLE AJOUTE:
      NUMR              NUMP              PRIX
      7                 10                110
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?insert(menus, numr:=7, nump:=2, prix:= 55);

INSERT(MENUS, NUMR:=7, NUMP:=2, PRIX:= 55);

VOULEZ VOUS CHANGER ? (O/N)

?n
```

```
      NUMR              NUMP              PRIX
      7                 2                 55

       TUPLE AJOUTE:
      NUMR              NUMP              PRIX
      7                 2                 55
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?delete(plats, nump=1);


DELETE(PLATS, NUMP=1);

VOULEZ VOUS CHANGER ? (O/N)

?n


      NUMP
      1

     TUPLE EFFACE:
      NUMP              NOMP              NCAL
      1                 CHOUCROUTE        4000
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
```



```
?delete(plats,nump=2);


DELETE(PLATS,NUMP=2);

VOULEZ VOUS CHANGER ? (O/N)

?n


      NUMP
      2
VIOLATION INTEGRITE REFERENTIELLE
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?CR
?CR

       UTILISER LE SCHEMA SUIVANT :

      MULTIBASE

         DATABASE RESTAURANT
             SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
             PLATS ( NUMP#, NOMP, NCAL )
             MENUS ( NUMR#,NUMP#, PRIX)
         END DATABASE

      END MULTIBASE
```

$\exists m \in MENUS : m.NUMP = 2$

```
      INDIQUER LE TRAITEMENT PAR
      QUERY   : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?query

      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?project(join(join(salles,menus,numr=numr),plats,nump=nump),nomr,nomp,rue,tel);

PROJECT(JOIN(JOIN(SALLES,MENUS,NUMR=NUMR),PLATS,NUMP=NUMP),NOMR,NOMP,RUE,TEL);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NOMR              NOMP              RUE               TEL

      CORDELIERS        BROCHETTES        BENIT             3354732
      MANDARIN          CANARD-LAQUE      PL-CROIX-BOURG    3402785
      MONEDA            PIZZA             COMMANDERIE       3404242
      CAMARGUE          PIZZA             ST-DIZIER         3353117
      DES-AMIS          COUSCOUS          4-EGLISES         3355011
      DES-AMIS          PAELA             4-EGLISES         3355011
      ALADIN            PASTILLA          4-EGLISES         3322132
      ALADIN            COUSCOUS          4-EGLISES         3322132
```



```
      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
?(CR)
       UTILISER LE SCHEMA SUIVANT :

      MULTIBASE

         DATABASE RESTAURANT
              SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
              PLATS ( NUMP#, NOMP, NCAL )
              MENUS ( NUMR#,NUMP#, PRIX)
         END DATABASE

      END MULTIBASE


      INDIQUER LE TRAITEMENT PAR
      QUERY   : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?(CR)
```

T Y P R A V O T R E S E R V I C E

```
       VOUS ETES CONNECTE A LA MULTIBASE:

      MULTIBASE LOISIR

         DATABASE RESTAURANT
             SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
             PLATS ( NUMP#, NOMP, NCAL )
             MENUS ( NUMR#,NUMP#, PRIX)
         END DATABASE

         DATABASE CINEMA
             SALLES ( NUMC#, NOMC, RUE, TEL)
             FILMS ( NUMF#, NOMF, GENRE )
             SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
         END DATABASE

      END MULTIBASE


      VOUS POUVEZ TRAVAILLER SUR LES BASES DE VOTRE CHOIX.
      INDIQUER LE NOM DE LA BASE
       OU FAIRE RETOUR CHARIOT SI FINI.
?cinema

      INDIQUER LE NOM DE LA BASE
       OU FAIRE RETOUR CHARIOT SI FINI.
?
```

(CR)

Manipulation de la base : CINEM

```
      UTILISER LE SCHEMA SUIVANT :

      MULTIBASE

         DATABASE CINEMA
             SALLES ( NUMC#, NOMC, RUE, TEL)
             FILMS ( NUMF#, NOMF, GENRE )
             SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
         END DATABASE

      END MULTIBASE


      INDIQUER LE TRAITEMENT PAR
      QUERY   : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?query


      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
  n      VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?project(salles,numc,nomc,rue,tel);


PROJECT(SALLES,NUMC,NOMC,RUE,TEL);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n
```

```
      NUMC              NOMC              RUE               TEL

      1                 CAMEO             COMMANDERIE       3403568
      2                 PARAMOUNT         BENIT             3354557
      3                 RIO               ST-DIZIER         3322487
      4                 PATHE             LALLEMENT         3354776
      5                 PARC              MAL-JUIN          3275788


      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?project(films,numf,nomf,genre);


PROJECT(FILMS,NUMF,NOMF,GENRE);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NUMF              NOMF              GENRE

      2                 MESSAGER          ROMANESQUE
      22                REDS              HISTORIQUE
      4                 RAGTIME           DRAMATIQUE
      6                 PROFESSIONNEL     POLAR
      9                 ROX-ET-ROUKY      DESSIN-ANIME
      99                M-A-T             VIOLENT
```



```
      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
?(CR)
       UTILISER LE SCHEMA SUIVANT :

      MULTIBASE

         DATABASE CINEMA
             SALLES ( NUMC#, NOMC, RUE, TEL)
             FILMS ( NUMF#, NOMF, GENRE )
             SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
         END DATABASE

      END MULTIBASE


      INDIQUER LE TRAITEMENT PAR
      QUERY  : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?update


       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?update(films,numf=22:nomf:=ghandi);
```

```
UPDATE(FILMS,NUMF=22:NOMF:=GHANDI);

VOULEZ VOUS CHANGER ? (O/N)

?n

    ANCIEN TUPLE:
      NUMF              NOMF              GENRE
      22                REDS              HISTORIQUE
    NOUVEAU TUPLE:
      NUMF              NOMF              GENRE
      22                GHANDI            HISTORMQUE
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?delete(films,numf=6);


DELETE(FILMS,NUMF=6);

VOULEZ VOUS CHANGER ? (O/N)

?n
```



```
      NUMF
      6
VIOLATION INTEGRITE REFERENTIELLE
       SOUMETTRE UNE MISE A JOUR OU
       FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
?(CR)
       UTILISER LE SCHEMA SUIVANT :

      MULTIBASE

         DATABASE CINEMA
             SALLES ( NUMC#, NOMC, RUE, TEL)
             FILMS ( NUMF#, NOMF, GENRE )
             SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
         END DATABASE

      END MULTIBASE


      INDIQUER LE TRAITEMENT PAR
      QUERY  : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?query
```

$\exists\ s \in SEANCES : s.NUMF = 6$

```
      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?select(seances,numf=6);

SELECT(SEANCES,NUMF=6);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NUMC            NUMF            HEURE           PRIX

      2               6               20              30

      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?(CK)
       UTILISER LE SCHEMA SUIVANT :
```

S. NumF = 6 s ∈ SEA


```
     MULTIBASE

        DATABASE CINEMA
            SALLES ( NUMC#, NOMC, RUE, TEL)
            FILMS ( NUMF#, NOMF, GENRE )
            SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
        END DATABASE

     END MULTIBASE


     INDIQUER LE TRAITEMENT PAR
     QUERY  : POUR L INTERROGATION,
     UPDATE  : POUR  LA M-A-J:
?update


      SOUMETTRE UNE MISE A JOUR OU
      FAIRE RETOUR CHARIOT.
       VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?delete(seances,numc=2,numf=6);


DELETE(SEANCES,NUMC=2,NUMF=6);

VOULEZ VOUS CHANGER ? (O/N)

?n
```

```
     NUMC              NUMF
     2                 6


    TUPLE EFFACE;
     NUMC              NUMF              HEURE             PRIX
     2                 6                 20                30
      SOUMETTRE UNE MISE A JOUR OU
      FAIRE RETOUR CHARIOT.
       VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?delete(films,numf=6);


DELETE(FILMS,NUMF=6);

VOULEZ VOUS CHANGER ? (O/N)

?n
```



```
     NUMF
     6

    TUPLE EFFACE:
     NUMF              NOMF              GENRE
     6                 PROFESSIONNEL     POLAR
      SOUMETTRE UNE MISE A JOUR OU
      FAIRE RETOUR CHARIOT.
       VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
?(CR)
      UTILISER LE SCHEMA SUIVANT :

     MULTIBASE

        DATABASE CINEMA
            SALLES ( NUMC#, NOMC, RUE, TEL)
            FILMS ( NUMF#, NOMF, GENRE )
            SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
        END DATABASE

     END MULTIBASE


     INDIQUER LE TRAITEMENT PAR
     QUERY  : POUR L INTERROGATION,
     UPDATE  : POUR  LA M-A-J:
?(CR)
```

```
T Y P R  A  V O T R E  [ E R V I C E


 VOUS ETE[ CONNECTE A LA MULTIBASE:

MULTIBASE LOISIR

   DATABASE RESTAURANT
       SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
       PLATS ( NUMP#, NOMP, NCAL )
       MENUS ( NUMR#,NUMP#, PRIX)
   END DATABASE

   DATABASE CINEMA
       SALLES ( NUMC#, NOMC, RUE, TEL)
       FILMS ( NUMF#, NOMF, GENRE )
       SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
   END DATABASE

END MULTIBASE


VOUS POUVEZ TRAVAILLER SUR 1ES BASES DE VOTRE CHOIX.
MNDIQUER LE NOM DE LA BASE
 OU FAIRE RETOUR CHARIOT SI FINI.
?restaurant


INDIQUER LE NOM DE LA BASE
 OU FAIRE RETOUR CHARIOT SI FINI.
?cinema
```



```
INDIQUER LE NOM DE LA BASE
 OU FAIRE RETOUR CHARIOT SI FINI.
?(CR)


 UTILISER LE SCHEMA SUIVANT :

MULTIBASE

   DATABASE RESTAURANT
       SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
       PLATS ( NUMP#, NOMP, NCAL )
       MENUS ( NUMR#,NUMP#, PRIX)
   END DATABASE

   DATABASE CINEMA
       SALLES ( NUMC#, NOMC, RUE, TEL)
       FILMS ( NUMF#, NOMF, GENRE )
       SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
   END DATABASE

END MULTIBASE
```

Manipulation de la base :
RESTAURANT et CINEMA

```
      INDIQUER LE TRAITEMENT PAR
      QUERY  : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
?query


      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?project(restaurant.salles,nomr,rue);


PROJECT(RESTAURANT.SALLES,NOMR,RUE);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NOMR              RUE

      GOELAND           DES-PONTS
      CORDELIERS        BENIT
      MANDARIN          PL-CROIX-BOURG
      MONEDA            COMMANDERIE
      CAMARGUE          ST-DIZIER
      DES-AMIS          4-EGLISES
      ALADIN            4-EGLISES
```



```
      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n

?project(cinema.salles,nomc,rue);


PROJECT(CINEMA.SALLES,NOMC,RUE);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NOMC              RUE

      CAMEO             COMMANDERIE
      PARAMOUNT         BENIT
      RIO               ST-DIZIER
      PATHE             LALLEMENT
      PARC              MAL-JUIN


      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA [YNTAXE?(O/N)
?n

?project(join(cinema.salles,restaurant.salles,rue=rue),nomr,nomc,rue);
```

```
PROJECT(JOIN(CINEMA.SALLES,RESTAURANT.SALLES,RUE=RUE),NOMR,NOMC,RUE);

      VOULEZ VOUS CHANGER LA REQUETE? (O/N)

?n

      NOMR              NOMC              RUE

      MONEDA            CAMEO             COMMANDERIE
      CORDELIERS        PARAMOUNT         BENIT
      CAMARGUE          RIO               ST-DIZIER


      SOUMETTRE UNE REQUETE OU
      FAIRE RETOUR CHARIOT.
        VOULEZ-VOUS LA SYNTAXE?(O/N)
?n
? (CR)
       UTILISER LE SCHEMA SUIVANT :

      MULTIBASE

         DATABASE RESTAURANT
             SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
             PLATS ( NUMP#, NOMP, NCAL )
             MENUS ( NUMR#,NUMP#, PRIX)
         END DATABASE

         DATABASE CINEMA
             SALLES ( NUMC#, NOMC, RUE, TEL)
             FILMS ( NUMF#, NOMF, GENRE )
             SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
         END DATABASE

      END MULTIBASE
```



```
      INDIQUER LE TRAITEMENT PAR
      QUERY  : POUR L INTERROGATION,
      UPDATE  : POUR  LA M-A-J:
? (CR)

           T Y P  R  A  V O T R E  S E R V I C E


       VOUS ETES CONNECTE A LA MULTIBASE:

      MULTIBASE LOISIR

         DATABASE RESTAURANT
             SALLES ( NUMR#, NOMR, RUE, TYPE, TEL )
             PLATS ( NUMP#, NOMP, NCAL )
             MENUS ( NUMR#,NUMP#, PRIX)
         END DATABASE

         DATABASE CINEMA
             SALLES ( NUMC#, NOMC, RUE, TEL)
             FILMS ( NUMF#, NOMF, GENRE )
             SEANCES ( NUMC#, NUMF#, HEURE, PRIX )
         END DATABASE

      END MULTIBASE
```

```
      VOUS POUVEZ TRAVAILLER SUR LES BASES DE VOTRE CHOIX.
      INDIQUER LE NOM DE LA BASE
       OU FAIRE RETOUR CHARIOT SI FINI.
? [illegible]

AU REVOIR
 00336: TIME := #00003A30
T-N: OK
?
```

ANNEXE B

EXEMPLES d'UNITES A T M :

TYPES, CAPSULES, MACHINES et MODULES

CORRESPONDANT à la DESCRIPTION de RELATIONS

en T Y P.

```
      VOUS POUVEZ TRAVAILLER SUR LES BASES DE VOTRE CHOIX.
      INDIQUER LE NOM DE LA BASE
       OU FAIRE RETOUR CHARIOT SI FINI.
? [illegible]

AU REVOIR
 00336: TIME := #00003A30
T-N: OK
?
```

ANNEXE B

EXEMPLES d'UNITES A T M :

TYPES, CAPSULES, MACHINES et MODULES

CORRESPONDANT à la DESCRIPTION de RELATIONS

en T Y P.

```
TYPE RESTAU;
     # TYPE RESTAURANT

BUILD CONST(INTEGER,TEXT,TEXT,TEXT,INTEGER);
     # CONSTRUCTION D'UN OBJET DE TYPE RESTAURANT.
     # AVEC POUR CHAQUE RESTAURANT UN NUMERO, UN NOM,
     # UNE RUE, LE TYPE DE LA CUISINE ET UN NUMERO DE
     # TELEPHONE.
FUNCTION NUMR RETURNS INTEGER;
     # FONCTION D'ACCES AU NUMERO.
FUNCTION NOMR RETURNS TEXT;
     # FONCTION D'ACCES AU NOM.
FUNCTION RUE RETURNS TEXT;
     # FONCTION D'ACCES AU NOM DE LA RUE.
FUNCTION TYPE RETURNS TEXT;
     #FONCTION D'ACCES AU TYPE DE CUISINE.
FUNCTION TEL RETURNS INTEGER;
     #FONCTION D'ACCES AU NUMERO TELEPHONE.

     # EXEMPLE:
     # SI ON DECLARE  R:RESTAU;I,J:INTEGER;P,S,T:TEXT;
     # APRES  @ R CONST(I,P,S,T,J) ON A:
     # @ R NUMR = I, @ R NOMR = P, @ R RUE = S, @ R TYPE =T
     # ET @ R TEL = J.
FUNCTION EQUAL(RESTAU)RETURNS BOOLEAN;
     # @ R1 EQUAL(R2) = VRAI SSI @ R1 NUMR = @ R2 NUMR.
FUNCTION CONVTUP RETURNS TUPLE;
     # CONVERSION D'UN OBJET DE TYPE RESTAU EN UN
     # OBJET DE TYPE TUPLE.
END
```

```
TYPE PLAT;

BUILD CONST(INTEGER,TEXT,INTEGER);
     # CONSTRUCTION D'UN OBJET DE TYPE PLAT,
     # AVEC POUR CHAQUE PLAT UN NUMERO(>0), UN NOM,
     # ET LA QTE DE CALORIES (>0).
FUNCTION NUMP RETURNS INTEGER;
     # FONCTION D'ACCES AU NUMERO.
FUNCTION NOMP RETURNS TEXT;
     # FONCTION D'ACCES AU NOM.
FUNCTION NCAL RETURNS INTEGER;
     # FONCTION D'ACCES AU NOMBRE DE CALORIES
     # D'UN PLAT.
FUNCTION EQUAL(PLAT)RETURNS BOOLEAN;
     # @ P1 EQUAL(P2) =VRAI SSI @ P1 NUMP = @ P2 NUMP.
FUNCTION CONVTUP RETURNS TUPLE;
     # OPERATION DE CONVERSION D'UN OBJET DE TYPE
     # PLAT EN UN OBJET DE TYPE TUPLE.
END
```



```
TYPE SEANCE;
BUILD CONST(INTEGER,INTEGER,INTEGER,INTEGER);
    #CONSTRUIT UN OBJET DE TYPE SEANCE
    # AVEC UN NUMERO DE CINEMA , UN NUMERO DE FILM,
    # L'HEURE DE PROJECTION ET LE PRIX DE LA SEANCE.
FUNCTION NUMC RETURNS INTEGER;
    # FONCTION D'ACCES AU NUMERO DU CINEMA.
FUNCTION NUMF RETURNS INTEGER;
    # FONCTION D'ACCES AU NUMERO DU FILM.
FUNCTION HEURE RETURNS INTEGER;
    # FONCTION D'ACCES A L'HEURE DE PROJECTION.
FUNCTION PRIX RETURNS INTEGER;
    # FONCTION D'ACCES AU PRIX DE LA SEANCE.
FUNCTION EQUAL(SEANCE) RETURNS BOOLEAN;
    #@S1 EQ(S2) = VRAI SSI (@S1 NUMF=@S2 NUMF)ET
    #(@S1 NUMC = @S2 NUMC)
FUNCTION CONVTUP RETURNS TUPLE;
    #CONVERTIT UN OBJET DE TYPE SEANCE EN UN OBJET DE TYPE TUPLE
END
```

```
CAPSULE RESTAU FOR RESTAU;
REP
   NUMR,TEL:INTEGER;
   NOMR,RUE,TYPE:TEXT;
END

BUILD CONST(I:INTEGER,P:TEXT,S:TEXT,T:TEXT,J:INTEGER);
  BEGIN
   IF I<=0
   THEN $ WRITER WRITE(´ERREUR:NUMERO RESTAURANT NEGATIF POUR:´);
        $ WRITER WRITE(P);
   ELSIF J<0
      THEN $ WRITER WRITE(´ERREUR:TELEPHONE RESTAURANT NEGATIF POUR:´);
        $ WRITER WRITE(P);
      ELSE NUMR:=I;NOMR:=P;RUE:=S;TYPE:=T;TEL:=J;
    ENDIF;
END
FUNCTION NUMR RETURNS INTEGER;BEGIN RETURN NUMR; END
FUNCTION NOMR RETURNS TEXT; BEGIN RETURN NOMR; END
FUNCTION RUE RETURNS TEXT; BEGIN RETURN RUE; END
FUNCTION TYPE RETURNS TEXT; BEGIN RETURN TYPE; END
FUNCTION TEL RETURNS INTEGER; BEGIN RETURN TEL; END
FUNCTION EQUAL(R:RESTAU)RETURNS BOOLEAN;
   BEGIN RETURN @ R NUMR = NUMR; END
FUNCTION CONVTUP RETURNS TUPLE;
   VAR T:TUPLE END
   BEGIN @ T INIT(5);
     @ T PUTCONST(1,´NUMR´);@ T PUTDOM(1,´INTEGER´);@ T PUTVALI(1,NUMR);
     @ T PUTCONST(5,´TEL´);@ T PUTDOM(5,´INTEGER´);@ T PUTVALI(5,TEL);
     @ T PUTCONST(2,´NOMR´);@ T PUTDOM(2,´TEXT´);@ T PUTVALT(2,NOMR);
     @ T PUTCONST(3,´RUE´);@ T PUTDOM(3,´TEXT´);@ T PUTVALT(3,RUE);
     @ T PUTCONST(4,´TYPE´);@ T PUTDOM(4,´TEXT´);@ T PUTVALT(4,TYPE);
     RETURN T;
END
```



```
CAPSULE PLAT FOR PLAT;
REP
   NUMP,NCAL:INTEGER;
   NOMP:TEXT;
END

BUILD CONST(N:INTEGER,T:TEXT,C:INTEGER);
    BEGIN
     IF (N<=0)OR(C<=0)
     THEN $ WRITER WRITE(´ERREUR:PARAMETRE NEGATIF DANS´);
          $ WRITER WRITE(´BUILD CONST DU TYPE PLAT´);
      ELSE NUMP:=N;NOMP:=T;NCAL:=C;
     ENDIF;
    END
FUNCTION NUMP RETURNS INTEGER; BEGIN RETURN NUMP; END
FUNCTION NOMP RETURNS TEXT; BEGIN RETURN NOMP; END
FUNCTION NCAL RETURNS INTEGER; BEGIN RETURN NCAL; END
FUNCTION EQUAL(P:PLAT)RETURNS BOOLEAN;
    BEGIN RETURN @ P NUMP = NUMP; END
FUNCTION CONVTUP RETURNS TUPLE;
    VAR T:TUPLE; END
    BEGIN
     @ T INIT(3);
     @ T PUTCONST(1,´NUMP´); @ T PUTDOM(1,´INTEGER´); @ T PUTVALI(1,NUMP);
     @ T PUTCONST(2,´NOMP´); @ T PUTDOM(2,´TEXT´); @ T PUTVALT(2,NOMP);
     @ T PUTCONST(3,´NCAL´); @ T PUTDOM(3,´INTEGER´); @ T PUTVALI(3,NCAL);
     RETURN T;
   END
END
```

```
CAPSULE SEANCE FOR SEANCE;
REP
  NUMF:INTEGER;
  NUMC :INTEGER;
  HEURE,PRIX :INTEGER;
END
BUILD CONST(NC:INTEGER,NF:INTEGER,H:INTEGER,P:INTEGER);
  BEGIN
    IF(NF<=0) OR(NC<=0)OR(H<0)OR(P<0) THEN
      $WRITER WRITE('ERREUR:PARAMETRE NEGATIF DANS');
      $WRITER WRITE('BUILD CONST DU TYPE SEANCE');
    ELSE
      NUMF :=NF;NUMC:=NC;HEURE:=H;PRIX:=P;
    ENDIF;
  END
FUNCTION NUMC RETURNS INTEGER;
  BEGIN  RETURN NUMC; END
FUNCTION NUMF RETURNS INTEGER;
  BEGIN RETURN NUMF; END
FUNCTION HEURE RETURNS INTEGER;
  BEGIN  RETURN HEURE ; END
FUNCTION PRIX RETURNS INTEGER;
  BEGIN  RETURN PRIX ; END
FUNCTION EQUAL(S:SEANCE) RETURNS BOOLEAN;
  BEGIN
    RETURN(@S NUMF=NUMF) AND(@S NUMC=NUMC);
  END
FUNCTION CONVTUP RETURNS TUPLE;
  VAR T : TUPLE; END
  BEGIN
    @T INIT(4);
    @T PUTCONST(1,'NUMC');@T PUTDOM(1,'INTEGER');@T PUTVALI(1,NUMC);
    @T PUTCONST(2,'NUMF');
    @T PUTDOM(2,'INTEGER');@T PUTVALI(2,NUMF);
    @T PUTCONST(3,'HEURE');@T PUTDOM(3,'INTEGER');@T PUTVALI(3,HEURE);
    @T PUTCONST(4,'PRIX');@T PUTDOM(4,'INTEGER');@T PUTVALI(4,PRIX);
    RETURN T;
  END
END
```



```
TYPE TUPLE;
    BUILD INIT(INTEGER);
          # INITIALISATION A FAIRE AVANT TOUTE OPERATION.
          # I=NOMBRE MAX DE CONSTITUANTS. 0<I<16.
    FUNCTION NB RETURNS INTEGER;
          # NOMBRE DE CHAMPS CONSTITUANT LE TUPLE.
    FUNCTION CONSTIT(INTEGER) RETURNS TEXT;
          # DONNE LE NOM DU IEME CONSTITUANT.
    FUNCTION DOM(INTEGER) RETURNS TEXT;
          # DONNE LE NOM DU DOMAINE DU IEME CONSTITUANT.
          # CE DERNIER EST SOIT TEXT SOIT INTEGER.
    FUNCTION VALTEXT(INTEGER)RETURNS TEXT;
    FUNCTION VALINT(INTEGER)RETURNS INTEGER;
          # CES FONCTIONS DONNENT LA VALEUR DU IEME
          # CONSTITUANT DANS LE CAS TEXT ET LE CAS INTEGER.
    MODIF PUTCONST(INTEGER,TEXT);#TEXT NE DOIT PAS CONTENIR LES CARACTERE $ OU &
    MODIF PUTDOM(INTEGER,TEXT);# MEME REMARQUE QUE PUTCONST
    MODIF PUTVALT(INTEGER,TEXT);
    MODIF PUTVALI(INTEGER,INTEGER);
          # "RECIPROQUES" DES 4 FONCTIONS PRECEDENTES.
    CONSULT TESTCONS(TEXT)->BOOLEAN,INTEGER;
          # TESTE SI LE CONSTITUANT EN PARAMETRE FAIT
          # PARTI DU TUPLE. SI OUI BOOLEAN VRAI EST INTEGER
          # VAUT LE RANG DE CE CONSTITUANT.
END
```

```
MACHINE OPTUPLE;
      # OPERATIONS SUR LES TUPLES
    PROC PRINT(TUPLE);
         # IMPRIME LES VALEURS CONSTITUANTS LE TUPLE.
    PROC PRINTC(TUPLE);
         # IMPRIME LES NOMS DES CONSTITUANTS.
    FUNCTION PROJECT(TUPLE,CRITEREP)RETURNS TUPLE;
         # PROJETTE LE TUPLE DONNE DANS LE TUPLE RESULTAT
          # SUIVANT LE CRITERE PASSE EN PARAMETRE
    FUNCTION CRITSEL(TUPLE,CRITERES)RETURNS BOOLEAN;
         # TEST SI LE TUPLE DONNE VERIFIE LE CRITERE
         # PASSE EN PARAMETRE
    FUNCTION CRITJOIN(TUPLE,TUPLE,CRITEREJ)RETURNS BOOLEAN;
          # TESTE SI LES 2 TUPLES DONNES VERIFIENT
         # LE CRITERE DU JOIN
    PROC CONCAT(TUPLE ,TUPLE,CRITEREJ)->TUPLE;
         # CONCATENE LE PREMIER ET LE DEUXIEME TUPLE
         # POUR FORMER LE TROISIEME. LE CONSTITUANT COMMUN N'EST PAS DUPPLIQUE.
         # SI LA LONGUEUR DU RESULTAT > 15 L'OPERATION EST SANS EFFET.
END
```



```
MACHINE RESTAUS ;
     #MACHINE QUI GERE L'ENSEMBLE DES RESTAURANTS DE LA BASE.

PROC VIDE;
    # INITIALISATION A VIDE.
PROC ADD(RESTAU)->BOOLEAN;
    # ADJONCTION D'UN RESTAURANT S'IL N'Y EST PAS.
    # BOOLEAN VRAI DANS CE CAS, FAUX AUTREMENT.
PROC DEL(RESTAU)->BOOLEAN;
    # SUPPRESSION DE RESTAURANT.
PROC ACCES(INTEGER)->RESTAU,BOOLEAN;
    # ACCES A UN RESTAURANT DE NUMERO DONNE.
    # BOOLEAN VRAI SI ACCES REALISE, FAUX SINON( RESTAU INEXISTANT).
FUNCTION EXISTE(RESTAU)RETURNS BOOLEAN;
    # TEST D'EXISTANCE.
ITER SELECT YIELDS RESTAU;
    # ITERATEUR QUI PARCOURT L'ENSEMBLE DES RESTAURANTS
    # ET LES FOURNIT UN PAR UN.
PROC INIT;
    # INITIALISATION DE L'ENSEMBLE DES RESTAURANTS
    # PAR LECTURE DANS L'ORDRE CROISSANT DES CLES.
END
```

```
MACHINE PLATS ;
     #MACHINE QUI GERE L'ENSEMBLE DES PLATS DE LA BASE.

PROC VIDE;
    # INITIALISATION A VIDE.
PROC ADD(PLAT)->BOOLEAN;
    # ADJONCTION D'UN PLAT S'IL N'Y EST PAS.
    # BOOLEAN VRAI DANS CE CAS, FAUX AUTREMENT.
PROC DEL(PLAT)->BOOLEAN;
    # SUPPRESSION DE PLAT.
PROC ACCES(INTEGER)->PLAT,BOOLEAN;
    # ACCES A UN PLAT DE NUMERO DONNE.
    # BOOLEAN VRAI SI ACCES REALISE, FAUX SINON( PLAT INEXISTANT).
FUNCTION EXISTE(PLAT)RETURNS BOOLEAN;
    # TEST D'EXISTANCE.
ITER SELECT YIELDS PLAT;
    # ITERATEUR QUI PARCOURT L'ENSEMBLE DES PLATS
    # ET LES FOURNIT UN PAR UN.
PROC INIT;
    # INITIALISATION DE L'ENSEMBLE DES PLATS
    # PAR LECTURE DANS L'ORDRE CROISSANT DES NUMEROS.
END
```



```
MACHINE SEANCES ;
     #MACHINE QUI GERE L'ENSEMBLE DES SEANCES DE LA BASE.

PROC VIDE;
    # INITIALISATION A VIDE.
PROC ADD(SEANCE)->BOOLEAN;
    # ADJONCTION D'UN SEANCE S'IL N'Y EST PAS.
    # BOOLEAN VRAI DANS CE CAS, FAUX AUTREMENT.
PROC DEL(SEANCE)->BOOLEAN;
    # SUPPRESSION DE SEANCE.
PROC ACCES(INTEGER,INTEGER)->SEANCE,BOOLEAN;
    # ACCES A UN SEANCE DE NUMERO CINEMA ET FILME DONNE.
    # BOOLEAN VRAI SI ACCES REALISE, FAUX SINON( SEANCE INEXISTANT).
FUNCTION EXISTE(SEANCE)RETURNS BOOLEAN;
    # TEST D'EXISTANCE.
ITER SELECT YIELDS SEANCE;
    # ITERATEUR QUI PARCOURT L'ENSEMBLE DES SEANCES
    # ET LES FOURNIT UN PAR UN.
PROC INIT;
    # INITIALISATION DE L'ENSEMBLE DES SEANCES
    # PAR LECTURE DANS L'ORDRE CROISSANT DES NUMEROS.
END
```

```
MODULE RESTAUS FOR RESTAUS;
   RESOURCE
      F:FIND;
   END

PROC VIDE;
   BEGIN
 @ F CRE('REST',136,1024,'RESTAURANT',8,1);
     END
PROC ADD(R:RESTAU)->B:BOOLEAN;
   VAR NUMR,NOMR,RUE,TYPE,TEL,T:TEXT;B1:BOOLEAN;END
   BEGIN NUMR:=' ';
     @ NUMR CONTEXT( $ CONV TEXTINT(@ R NUMR));@ NUMR AJUST(9);
     NOMR:=@ R NOMR; @ NOMR AJUST(19);@ NOMR CONCHAR(" ");
     RUE := @ R RUE; @ RUE AJUST(19);@ RUE CONCHAR(" ");
     TYPE:= @ R TYPE; @ TYPE AJUST(19);@ TYPE CONCHAR(" ");
     TEL := $ CONV TEXTINT(@ R TEL);@ TEL AJUST(7);
     @ F OPENU;
         @ NUMR CONTEXT(NOMR);
         @ NUMR CONTEXT(RUE);
         @ NUMR CONTEXT(TYPE);
         @ NUMR CONTEXT(TEL);
         @ NUMR AJUST(136);
         @ F PUTN(NUMR)->B;
     @ F CLBEGIN;END
PROC DEL(R:RESTAU)->B:BOOLEAN;
   VAR CLE,T:TEXT;B1:BOOLEAN; END
   BEGIN
     @ F OPENU;CLE:= $ CONV TEXTINT(@ R NUMR);@ CLE AJUST(8);
     @ F GETN(CLE)->T,B1;B:=B1;
     IF B1
     THEN @ F DELETE;
     ENDIF;
     @ F CLBEGIN;
   END
PROC ACCES(I:INTEGER)->R:RESTAU,B1:BOOLEAN;
    VAR B:BOOLEAN;
    T,N,NOMR,RUE,TYPE:TEXT;NUMR,TEL:INTEGER;END
    BEGIN
       @ F OPENU;
        @ F GETN('0000')->T,B;
        @ F GET->T,B;
      WHILE B;
          N:=@ T SUBSTR(2,9);
         NUMR:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(' ')-1));
       IF NUMR=I THEN
       B1:=TRUE;
          NOMR:=@ T SUBSTR(10,29);NOMR:=@ NOMR SUBSTR(1,@ NOMR INDEX(' ')-1);
          RUE := @ T SUBSTR(30,49);RUE:=@ RUE SUBSTR(1,@ RUE INDEX(' ')-1);
          TYPE :=@ T SUBSTR(50,69);TYPE:=@ TYPE SUBSTR(1,@ TYPE INDEX(' ')-1);
          TEL:=$ CONV INTTEXT(@T SUBSTR(70,76));
          @ R CONST(NUMR,NOMR,RUE,TYPE,TEL);
          @ F CLBEGIN;RETURN;
        ELSE
        @ F GET->T,B;ENDIF;
     ENDWHILE;
      @ F CLBEGIN;B1:=FALSE;
  END
```



```
FUNCTION EXISTE(R:RESTAU)RETURNS BOOLEAN;
    VAR B:BOOLEAN;R1:RESTAU;END
    BEGIN
     $ RESTAUS ACCES( @ R NUMR)->R1,B;
        RETURN B;
     END
ITER SELECT YIELDS RESTAU;
    VAR R:RESTAU;B:BOOLEAN;
    T,N,NOMR,RUE,TYPE:TEXT;NUMR,TEL:INTEGER;END
    BEGIN
       @ F OPENU;
        @ F GETN('0000')->T,B;
        @ F GET->T,B;
      WHILE B;
         N:=@ T SUBSTR(2,9);NUMR:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(' ')-1));
          NOMR:=@ T SUBSTR(10,29);NOMR:=@ NOMR SUBSTR(1,@ NOMR INDEX(' ')-1);
          RUE := @ T SUBSTR(30,49);RUE:=@ RUE SUBSTR(1,@ RUE INDEX(' ')-1);
          TYPE :=@ T SUBSTR(50,69);TYPE:=@ TYPE SUBSTR(1,@ TYPE INDEX(' ')-1);
          TEL:=$ CONV INTTEXT(@T SUBSTR(70,76));
          @ R CONST(NUMR,NOMR,RUE,TYPE,TEL);
          YIELD R;
          @ F GET ->T,B;
     ENDWHILE;
      @ F CLBEGIN;END
PROC INIT;
    VAR NUMR,T:TEXT;B:BOOLEAN;END
    BEGIN
      $ WRITER WRITE('INITIALISATION DES RESTAURANTS');
      @ F OPENP; $ WRITER WRITE('NUMR#');
      $ READER READ->T,B;
      WHILE @ T LENGTH<>0
       @ T AJUST(8);NUMR:=' ';@ NUMR CONTEXT(T);
      $ WRITER WRITE('NOMR');$ READER READ->T,B;@ T AJUST(20);@ NUMR CONTEXT(T);
       $ WRITER WRITE('RUE');$ READER READ->T,B;@ T AJUST(20);@ NUMR CONTEXT(T);
       $ WRITER WRITE('TYPE');$ READER READ->T,B;@ T AJUST(20);@NUMR CONTEXT(T);
       $ WRITER WRITE('TEL');$ READER READ->T,B;@ T AJUST(7);@ NUMR CONTEXT(T);
       @ NUMR AJUST(136); @ F PUT(NUMR)->B;$ WRITER WRITE('NUMR#');
       $ READER READ->T,B;
     ENDWHILE;
     @ F CLBEGIN;
END
```

```
MODULE PLATS FOR PLATS;
   RESOURCE
      F:FIND;
   END

PROC VIDE;
   BEGIN
 @ F CRE(´PLAT´,136,1024,´PLATS´,8,1);
     END
PROC ADD(R:PLAT)->B:BOOLEAN;
   VAR NUMP,NOMP,NCAL,T:TEXT;B1:BOOLEAN;END
   BEGIN NUMP:=´ ´;
     @ NUMP CONTEXT( $ CONV TEXTINT(@ R NUMP));@ NUMP AJUST(9);
     NOMP:=@ R NOMP; @ NOMP AJUST(20);
     NCAL := $ CONV TEXTINT(@ R NCAL);@ NCAL AJUST(8);
     @ F OPENU;
      @ NUMP CONTEXT(NOMP);
           @ NUMP CONTEXT(NCAL);
           @ NUMP AJUST(136);
           @ F PUTN(NUMP)->B;# B EST VRAI
     @ F CLBEGIN;END
PROC DEL(R:PLAT)->B:BOOLEAN;
   VAR CLE,T:TEXT;B1:BOOLEAN; END
   BEGIN
     @ F OPENU;CLE:= $ CONV TEXTINT(@ R NUMP);@ CLE AJUST(8);
     @ F GETN(CLE)->T,B1;B:=B1;
     IF B1
     THEN @ F DELETE;
     ENDIF;
     @ F CLBEGIN;
   END
PROC ACCES(I:INTEGER)->P:PLAT,B1:BOOLEAN;
    VAR B:BOOLEAN;
    T,N,NOMP:TEXT;NUMP,NCAL:INTEGER;END
    BEGIN
       @ F OPENU;
        @ F GETN(´0000´)->T,B;
        @ F GET->T,B;
      WHILE B;
         N:=@ T SUBSTR(2,9);NUMP:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(´ ´)-1));
         IF NUMP=I THEN B1:=TRUE;
          NOMP:=@ T SUBSTR(10,29); @ NOMP CONCHAR(" ");
          NOMP:= @ NOMP SUBSTR(1, @ NOMP INDEX(´ ´)-1);
       N:=@ T SUBSTR(30,37);NCAL:=$ CONV INTTEXT(@N SUBSTR(1,@N INDEX(´ ´)-1));
         @ P CONST(NUMP,NOMP,NCAL);
          @ F CLBEGIN;RETURN;
         ELSE
         @ F GET ->T,B;ENDIF;
     ENDWHILE;
      @ F CLBEGIN;B1:=FALSE;END
```



```
FUNCTION EXISTE(R:PLAT)RETURNS BOOLEAN;
    VAR B:BOOLEAN;R1:PLAT;END
    BEGIN
     $ PLATS ACCES( @ R NUMP)->R1,B;
        RETURN B;
     END
ITER SELECT YIELDS PLAT;
    VAR R:PLAT;B:BOOLEAN;
    T,N,NOMP:TEXT;NUMP,NCAL:INTEGER;END
    BEGIN
       @ F OPENU;
        @ F GETN(´0000´)->T,B;
        @ F GET->T,B;
      WHILE B;
         N:=@ T SUBSTR(2,9);NUMP:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(´ ´)-1));
          NOMP:=@ T SUBSTR(10,29); @ NOMP CONCHAR(" ");
          NOMP:= @ NOMP SUBSTR(1, @ NOMP INDEX(´ ´)-1);
       N:=@ T SUBSTR(30,37);NCAL:=$ CONV INTTEXT(@N SUBSTR(1,@N INDEX(´ ´)-1));
          @ R CONST(NUMP,NOMP,NCAL);
         YIELD R;
         @ F GET ->T,B;
     ENDWHILE;
      @ F CLBEGIN;END
PROC INIT;
    VAR NUMP,T:TEXT;B:BOOLEAN;END
    BEGIN
      $ WRITER WRITE(´INITIALISATION DES PLATS´);
      @ F OPENP; $ WRITER WRITE(´NUMP#´);
      $ READER READ->T,B;
      WHILE @ T LENGTH<>0
       @ T AJUST(8);NUMP:=´ ´;@ NUMP CONTEXT(T);
      $ WRITER WRITE(´NOMP´);$ READER READ->T,B;@ T AJUST(20);@ NUMP CONTEXT(T);
       $ WRITER WRITE(´NCAL´);$ READER READ->T,B;@ T AJUST(8);@ NUMP CONTEXT(T);
       @ NUMP AJUST(136); @ F PUT(NUMP)->B;$ WRITER WRITE(´NUMP#´);
       $ READER READ->T,B;
     ENDWHILE;
     @ F CLBEGIN;
END
```

```
MODULE SEANCES FOR SEANCES;
   RESOURCE
      F:FIND;
   END

PROC VIDE;
   BEGIN
 @ F CRE(´SEAN´,136,1024,´SEANCES´,16,1);
      END
PROC ADD(R:SEANCE)->B:BOOLEAN;
   VAR NUMC,NUMF,HEURE,PRIX,T:TEXT;B1:BOOLEAN;END
   BEGIN NUMC:=´ ´;
     @ NUMC CONTEXT( $ CONV TEXTINT(@ R NUMC));@ NUMC AJUST(9);
     NUMF:=$ CONV TEXTINT(@ R NUMF); @ NUMF AJUST(8);
     HEURE:=$ CONV TEXTINT(@ R HEURE); @ HEURE AJUST(8);
     PRIX := $ CONV TEXTINT(@ R PRIX);@ PRIX AJUST(8);
     @ F OPENU;@ NUMC CONTEXT(NUMF); @ F GETN(NUMC)->T,B1;B:=B1;
     IF NOT B1
     THEN @ NUMC CONTEXT(HEURE); @ NUMC CONTEXT(PRIX);
          @ NUMC AJUST(136);
          @ F PUTN(NUMC)->B;# B EST VRAI
     ENDIF;
     @ F CLBEGIN;END
PROC DEL(R:SEANCE)->B:BOOLEAN;
   VAR CLE,T:TEXT;B1:BOOLEAN; END
   BEGIN
     @ F OPENU;CLE:= $ CONV TEXTINT(@ R NUMC);@ CLE AJUST(8);
 @ CLE CONTEXT($CONV TEXTINT(@R NUMF));@CLE AJUST(16);@ F GETN(CLE)->T,B1;B:=B1;
 $ WRITER WRITE(CLE);$ WRITER WRITE(@T SUBSTR(1,130));      IF B1
     THEN @ F DELETE;
     ENDIF;
     @ F CLBEGIN;
   END
PROC ACCES(I:INTEGER,J:INTEGER)->S:SEANCE,B:BOOLEAN;
  VAR CLE,N,T:TEXT;HEURE,PRIX:INTEGER;B1:BOOLEAN;END
  BEGIN
    CLE:=$ CONV TEXTINT(I); @ CLE AJUST(8);
    @ CLE CONTEXT($ CONV TEXTINT(J)); @ CLE AJUST(16);
       @ T AJUST(136);
    @ F OPENU; @ F GETN(CLE)->T,B1; B:=B1;
    IF B1
    THEN
        N:=@ T SUBSTR(18,25);HEURE:=$CONV INTTEXT(@N SUBSTR(1,@N INDEX(´ ´)-1));
       N:=@ T SUBSTR(26,33);PRIX:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(´ ´)-1));
          @ S CONST(I,J,HEURE,PRIX);
     ENDIF;
    @ F CLBEGIN;
 END
```



```
FUNCTION EXISTE(R:SEANCE)RETURNS BOOLEAN;
    VAR B:BOOLEAN;R1:SEANCE;END
    BEGIN
     $ SEANCES ACCES( @ R NUMC, @ R NUMF)->R1,B;
         RETURN B;
     END
ITER SELECT YIELDS SEANCE;
    VAR R:SEANCE;B:BOOLEAN;
    T,N:TEXT;NUMF,NUMC,HEURE,PRIX:INTEGER;END
    BEGIN
       @ F OPENU;
        @ F GETN(´0000´)->T,B;
        @ F GET->T,B;
      WHILE B;
         N:=@ T SUBSTR(2,9);NUMC:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(´ ´)-1));
        N:=@ T SUBSTR(10,17);NUMF:=$CONV INTTEXT(@N SUBSTR(1,@N INDEX(´ ´)-1));
        N:=@ T SUBSTR(18,25);HEURE:=$CONV INTTEXT(@N SUBSTR(1,@N INDEX(´ ´)-1));
       N:=@ T SUBSTR(26,33);PRIX:=$ CONV INTTEXT(@N SUBSTR(1,@ N INDEX(´ ´)-1));
          @ R CONST(NUMC,NUMF,HEURE,PRIX);
         YIELD R;
         @ F GET ->T,B;
     ENDWHILE;
      @ F CLBEGIN;END
PROC INIT;
    VAR NUM,T:TEXT;B:BOOLEAN;END
    BEGIN
      $ WRITER WRITE(´INITIALISATION DES SEANCES´);
      @ F OPENP; $ WRITER WRITE(´NUMC#´);
      $ READER READ->T,B;
   WHILE @ T LENGTH<>0
    @ T AJUST(8);NUM:=´ ´;@ NUM CONTEXT(T);
    $ WRITER WRITE(´NUMF#´);$ READER READ->T,B;@ T AJUST(8);@ NUM CONTEXT(T);
    $ WRITER WRITE(´HEURE´);$ READER READ->T,B;@ T AJUST(8);@ NUM CONTEXT(T);
    $ WRITER WRITE(´PRIX´);$ READER READ->T,B;@ T AJUST(8);@ NUM CONTEXT(T);
    @ NUM AJUST(136); @ F PUT(NUM)->B;$ WRITER WRITE(´NUMC#´);
    $ READER READ->T,B;
     ENDWHILE;
     @ F CLBEGIN;
END
```

Bibliographie

[ADI 80] ADIBA M. et LINDSEY B.G.
" Database snapshots" dans[VLD 80].

[ANS 75] ANSI/X/SPARC
Study Group on Data Base Management Systems, Interim Report, FDT (ACM Sigmod Bul.) 7, No 2, 1975,

[AST 75] ASTRAHAN M.M. et CHAMBERLIN D.D.
"Implementation of a structured english query language." CACM, Vol 18, No 10, 1975, pp 580-588.

[AST 76] ASTRAHAN M.M. et al.
"SYSTEM R: A relational approach to database management" ACM TODS, Vol 1, No 2, 1976 , pp 97-136.

[AST 79] ASTRAHAN M.M. et al
"SYSTEM R : A relational database management system" IEEE computer society: computer 12, No 5, 1979, pp 43-48

[AST 81] ASTRAHAN M.M. et al.
"SYSTEM R : An architectural overview" IBM SJ, Vol 29, No 1, 1981, pp 41-46.

[BEC 80] BECK L.
"A generalized implementation method for relational data sublanguage" IEEE-TSE, Vol S.E.6 , No 2- Mars 1980, pp 152,162.

[BOY 73] BOYCE R.F. CHAMBERLIN D.D. KING W.F. et HAMMER M.M.
"SQUARE: Specifying QUeries As Relational Expressions" Proc. ACM SIGPLAN/SIGIR, Nov 1973. Aussi dans CACM, Vol 18 , No 11, 1975, pp 621-628 .

[CHA 74] CHAMBERLIN D.D. BOYCE R.F.
"SEQUEL : A Structured Engilsh Query Language" Proc. ACM SIGMOD, workshop on data description access and control, ACM N.Y. 1974, pp 249-264.

[CHA 76] CHAMBERLIN D.D.
"Relational Database Management Systems: a survey" Computing Surveys, Vol 8, No 1, 1976, pp 43-66.

[CHA 76a] CHAMBERLIN D.D. et al.
" SEQUEL 2 : A unified approach to data definition, manipulation and control" IBM JRD, Vol 20, No 6, 1976, pp 560-576.

[CHA 80] CHAMBERLIN D.D.
"A summary of user experience with the SQL data sublanguage" RJ 2737 IBM research report, San Jose, 1980.

[CHA 82] CHABRIER J.J. DERNIAME J.C.
"Programmation a l'aide de types abstraits : le système TYP" TSI, Vol 1, No 4, 1982

[COD 70] CODD E.F.
"A Relational Model of data for large shared data banks" CACM, Vol 13, No 6, 1970, pp 377-387.

[COD 71] CODD E.F.
" Further normalization of the database relational model" Database systems, Prentice Hall, Englwood Cliffs, N.J. 1971, pp 56-98.

[COD 71a] CODD E.F.
"A database sublanguage founded on the relational calculus". Proc. ACM SIGFIDET, Workshop on data description access and control. San Diego, 1971.

[COD 72] CODD E.F.
"Relational completness of database sublanguages" Database systems, Curent computer science symposia, Vol 6, Prentice Hall, 1972, pp 65-93.

[COD 74] CODD E.F.
"Recent investigations in relational database systems" Information processing, North Holland Publishing company,1974, pp 1017-1021.

[COD 79] CODD E.F.
"Extending the database relational model to capture more meaning" ACM TODS, Vol 4, No 4, Dec 1979, pp 397-434.

[COD 81] CODD E.F.
"Data model in database management" ACM SIGMOD RECORD, Vol 11, No 2, Fev 1981, pp 112-114.

[COD 82] CODD E.F.
"Relational database: a practical foundation of productivity" CACM, Vol 25, No 2, Fev 1982, pp 109-117

[CRE 74] CREHANGE M.
"Description, représentation, interrogation, traitement des informationsstructurées. Langage PIVOINES" RAIRO, Sept 1974, pp 5-43.

[DAT 81] DATE C.J.
"Referential integrity" Proc. of the 7th intern. conf. on VLDB, Sept 81, Cannes FRANCE.

[DAT 81a] DATE C.J.
"An Introduction to database systems" 3rd edition , 1981, ADDISON-WESLEY Pub. Comp.

[DAR 79] DARGENT L.
"Mise a jour de données réparties: controle de la concurrence d'accés" Thèse de troisième cycle, Univ. Nancy I , Dec 1979.

[DDB 80] DELOBEL C. et LITWIN W. Edition
"Distributed database systems" North Holland Publ. Comp., 1980.

[DEL 80] DELOBEL C.
"An overview of the relational data theory" Proc. IFIP Conf. 80, TOKYO, Oct 1980.

[DIJ 72] DIJKSTRA E.W. HOARE C.A.R. et DAHL O.J.
"Structured programming" New-York, Academic Press, 1972.

[ENG 71] ENGELS R.W.
"An analysis of the April 1971 DBTG Report" Proc. ACM SIGFIDET, Workshop on data description, access and control, 1971.

[ENG 72] ENGELS R.W.
"A tutorial on database organisation" Annual review in Automatic Programming , Vol 7, Pergamon Press, 1972, pp 1,64.

[ESC 81] ESCULIER C.
"Le SGBDR SIRIUS-DELTA et le système siloe de gestion de données réparties" Dans [SIR 81] pp 65-91.

[FAG 77] FAGIN R.
"Multivalued dependencies and a new normal form for relational databases" ACM TODS, Vol 2, No 3, Sept 1977, pp 262-278.

[GOD 81] GODARD C.
"SGMB: un système de gestion multibase", thèse de troisième cycle, Univ. de Nancy I , Juil 1981.

[GRI 82] GRISON T.
"Génération des textes ATM d'implantation de relations en TYP-R" Rapport DEA , Univ. Nancy I, Sept 1982.

[GUE 81] GUEMARA S.
"MRDSM: Système de gestion d'une multibase relationnelle." Publication INRIA-SIRIUS Numéro MOD-I-064, Oct 1981.

[HEL 75] HELD G.D. STONEBRAKER M.R. et WONG E.
"INGRES: A relational database system." Proc. NCC, AFIPS 44, Mai 1975, pp 447-452

[HEN 80] HENRY P.
"La connexion dans le projet TYP" Thèse de troisième cycle, Univ. Nancy I, Dec 1980.

[KAB 81] KABBAJ K.
"MUQUAPOL: Un système multibase" Publication INRIA-SIRIUS numéro MOD-I-044, Nov 1981.

[KAB 82] KABBAJ K.
"Un système de gestion multibase de données" Segundo symposia espanol de infomatica repartida, Santiago de Compostella, Sep. 1982.

[KAT 80] KATZ R.H.
"Database design and translation for multiple data models" Ph.D. Dissertation Univ. California, BERKELEY, Juin 1980.

[KAT 82] KATZ R.H. et WONG E.
"Decompiling CODASYL DML into Relational Queries" ACM TODS, Vol 7, No 1, Mars 1982, pp 1-23.

[KIM 79] KIM W.
"Relatinal database systems" Comp. Surveys, Vol 11, No 3, Sept 1979, pp 185-211.

[LEB 80] LE BIHAN J. et al.
"SIRIUS-DELTA: un prototype de système de gestion de bases de données réparties." Symposium international sur les bases de données réparties , Paris, Mars 1980. Aussi dans [DDB 80].

[LEB 80a] LE BIHAN J. et al.
"SIRIUS: A Frensh nationwide project on distributed databases" Publication INRIA numéro GAL-I-036. Aussi dans [VLD 80].

[LEB 81] LE BIHAN J.
"Une introduction aux systèmes de gestion de bases de données réparties." Dans [SIR 81] pp 7-22.

[LIS 74] LISKOV B.H. et ZILLES S.N.
"Programming with abstract data types" SIGPLAN notices, Vol 9, No 4, 1974.

[LIT 79] LITWIN W.
"Distributed databases: A way of thinking about." International seminar on distributed data sharing systems, Aix-en-Provence, Mai 1979.

[LIT 81] LITWIN W.
"A logical design of distributed databases" Publication INRIA-SIRIUS numero MOD-I-043, 1981.

[LIT 81a] LITWIN W.
"Présentation du projet B A BA"Publication INRIA-SIRIUS numéro GAL-I-042, 1981

[LIT 81b] LITWIN W.
"Projet B A BA: Gestion de multibases de données." Dans[SIR81] pp 321-325.

[MAI 81] MAIBAUN T.S.E.
"Database instances, abstract data types and database specifications" Univ. of Waterloo, Computer science dept. CANADA, N2L 3G1, 1981.

[McG 81] Mc GEE W.C.
"Database technology" IBM JRD, Vol 25, No 5, Sept 1981, pp 505-519.

[MEY 78] MEYER D. et DARGENT L.
"Une base de données répartie pour une organisation etoilée" congrès AFCET, Gif sur Yvette, 1978.

[MIN 79] MINOT R.
"ATM: un système de fabrication de programmes basé sur les concepts de modularité et de type abstrait" Thèse de troisième cycle, Univ. Nancy I, Mars 1979.

[NAS 81] NASSIF R.
"Réalisation des opérateeurs relationnels" Rapport de DEA, Univ. Nancy I, Juin 1981.

[NGU 80] NGUYEN G.T.
"Decentralized dynamic query decomposition for DDB systems" ACM pacific conference, San Francisco, Nov 1980.

[PAR 72] PARNAS D.L.
"A Technique for the Specification of Software Modules with Examples" CACM, Vol 15, No 5, Mai 1972, pp 330-336.

[PAO 80] PAOLINI P.
"Abstract data types and databases" Dans [SIG 80] pp 171-173.

[POL 79]
"POLYPHEME: un système de gestion de bases de données réparties" Publication INRIA-SIRIUS numéro GAL-I-034 et aussi dans [VLD 80].

[REY 75] REISNER P. BOYCE R.F. et CHAMBERLIN D.D.
"Human factors evaluation of two database query languages-SQUARE and SEQUEL." Proc. NCC, AFIPS 44, Mai 1975, pp 447-452.

[SHA 75] SHARMAN G.C.H.
" A new model of relational database and high level languages" T.R. 12.136, IBM Hersley Park Laboratory, England, Fev 1975.

[SIG 80]
SIGPLAN notices, Workshop in data abstraction, databases and conceptual modelling, Special Issue, Dec 1980 pp 46-48.

[SMI 75] SMITH J.M. CHANG G.P.
"Optimizing the performance of a relational algebra database interface" CACM, Vol 18, No 10, Oct 1975, pp 568-579

[SIR 81]
Projet SIRIUS, actes des journées de présentation des resultats, Paris Nov 1981. Publication INRIA-ADI 1981.

[THO 80] THOMPA F.W.
"A practical example of the specification of abstract data types" Acta informatica 13, 1980, pp 205-224

[TOU 79] TOUNSI N.
"Réalisation d'un mécanisme d'itérations de haut niveau en ATM" Rapport de DEA, Univ. de Nancy I, Sept. 1979.

[TOU 82] TOUNSI N. DERNIAME J.C. CHABRIER J.J. et LITWIN W.
"TYP-R: Un système de bases de données relationnelles realisé a l'aide de types abstraits de données" Segundo Symposia Espanol de Informatica Repartida, Santiago de Compostella, Sept. 1982.

[VLD 80]
Proc. of the 6th Intern. Conf. on VLDB, Monreal-CANADA, 1980.

[WIR 71] WIRTH N.
"Program development by stepwise refinement" CACM Vol 14, No 4, Avr 1971, pp 221-227.

[WON 79] WONG E. et KATZ R.H.
"Logical design and schema conversion for relational and DBTG databases" Proc. Int. conf. E-R approach to system analysis and design, UCLA, Dec 1979.

[ZLO 75] ZLOOF M.M.
"Query by example" Proc. NNC, AFIPS 44, Mai 1975, pp 431-438

NOM DE L'ETUDIANT : TOUNSI Najib

NATURE DE LA THESE : Doctorat 3ème cycle en Informatique

VU, APPROUVE

ET PERMIS D'IMPRIMER

NANCY, le 21 MARS 1983 465

LE PRESIDENT DE L'UNIVERSITE DE NANCY I,

RESUME :

T Y P - R est un système de bases de données relationnelles et réparties ecrit en T Y P : un système integré de programmation modulaire, avec utilisation de types abstraits de données.

Nous avons suivi une approche qui considère de telles bases comme des ensembles de bases, appelés MULTIBASES. Cette approche permet de concevoir des systèmes multibases sans se préoccuper de la répartition des données sur plus d'un site, et de construire de telles bases de manière progressive à partir de plusieurs bases élémentaires.

T Y P - R se caractérise par le fait d'étudier et d'appliquer les nouvelles méthodes et techniques en matière de génie logiciel pour développer des systèmes multibases. Un tel système doit être modulaire de façon à s'adapter progressivement à la construction d'une multibase, et l'utilisation de types abstraits de données permet de ne pas se préoccuper des contraintes physiques d'implantation.

MOTS CLES :

Bases de données relationnelles et réparties, Bases ensemble de bases, Multibases, Applications bases de données et Systèmes de programmation modulaire, Types abstraits et bases de données.