Réalisation d'un noyau de système de gestion de base de données relationnelle sous APL

Michel Nakache

ECOLE NATIONALE
SUPERIEURE DES MINES
DE SAINT-ETIENNE

INSTITUT NATIONAL
POLYTECHNIQUE
DE GRENOBLE

N° d'ordre: 10 ii

THESE

présentée par

Michel NAKACHE

pour obtenir le grade de

DOCTEUR-INGENIEUR
"SYSTEMES ET RESEAUX INFORMATIQUES"

REALISATION D'UN NOYAU DE SYSTEME DE GESTION DE BASE DE DONNEES RELATIONNELLE SOUS APL

soutenue à Saint-Etienne le 12 juin 1978 devant la commission d'examen:

Président M. VEILLON

Examinateurs MM. GUIBOUD-RIBAUD
KOULOUMIDJIAN
MAHL
ROHMER

ECOLE NATIONALE
SUPERIEURE DES MINES
DE SAINT-ETIENNE

INSTITUT NATIONAL
POLYTECHNIQUE
DE GRENOBLE

N° d'ordre: 10 ii

THESE

présentée par

Michel NAKACHE

pour obtenir le grade de

DOCTEUR-INGENIEUR
"SYSTEMES ET RESEAUX INFORMATIQUES"

REALISATION D'UN NOYAU DE SYSTEME DE GESTION DE BASE DE DONNEES RELATIONNELLE SOUS APL

soutenue à Saint-Etienne le 12 juin 1978 devant la commission d'examen:

Président	**M. VEILLON**
Examinateurs	**MM. GUIBOUD-RIBAUD**
	KOULOUMIDJIAN
	MAHL
	ROHMER

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur: M. G. ARNOUIL
Direction des Recherches: M. P. COUEIGNOUX
Direction des Etudes: M. A. COINDE

PROFESSEURS DE 1ère CATEGORIE

MM. COINDE Alexandre	Gestion
GOUX Claude	Métallurgie
LEVY Jacques	Métallurgie
RIEU Jean	Mécanique - Résistance des Matériaux
SOUSTELLE Michel	Chimie
FORMERY Philippe	Mathématiques Appliquées

PROFESSEURS DE 2ème CATEGORIE

MM. GUIBOUD-RIBAUD Serge	Informatique
LOWYS Jean-pierre	Physique
TOUCHARD Bernard	Physique Industrielle

PROFESSEURS ASSOCIES DE 2ème CATEGORIE

MM. DAVOINE Philippe	Géologie
PERRIN Michel	Géologie

DIRECTEUR DE RECHERCHE

M. LESBATS Pierre	Métallurgie

MAITRES DE RECHERCHE

MM. BISCONDI Michel	Métallurgie
BOOS Jean-Yves	Métallurgie
COUEIGNOUX Philippe	Informatique
Mlle FOURDEUX Angéline	Métallurgie
MM. LALAUZE René	Chimie
LANCELOT Francis	Chimie
LE COZE Jean	Métallurgie
THEVENOT François	Chimie
TRAN MINH Canh	Chimie

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président: M. P. TRAYNARD
Vice-Présidents: M. R. PAUTHENET
M. G. LESPINARD

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Electronique - Automatique
	BESSON Jean	Chimie Minérale
	BLOCH Daniel	Physique du Solide - Cristallographie
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUDOURIS Georges	Electronique - Automatique
	BRISSONNEAU Pierre	Physique du Solide - Cristallographie
	BUYLE-BODIN Maurice	Electronique - Automatique
	COUMES André	Electronique - Automatique
	DURAND Francis	Metallurgie
	FELICI Noël	Electronique - Automatique
	FOULARD Claude	Electronique - Automatique
	LANCIA Roland	Electronique - Automatique
	LONGEQUEUE Jean Pierre	Physique Nucléaire Corpusculaire
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean Charles	Chimie - Physique
	PAUTHENET René	Electronique - Automatique
	PERRET René	Electronique - Automatique
	POLOUJADOFF Michel	Electronique - Automatique
	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique Fondamentale et Appliquée

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuël	Electronique - Automatique
	BOUVARD Maurice	Génie Mécanique
	COHEN Joseph	Electronique - Automatique
	GUYOT Pierre	Mettallurgie Physique
	LACOUME Jean Louis	Electronique - Automatique
	JOUBERT Jean Claude	Physique du Solide - Cristallographie
	ROBERT André	Chimie Appliquée - Chimie des Matériaux
	ROBERT François	Analyse Numérique
	ZADWORNY François	Electronique - Automatique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Informatique Fondamentale et Appliquée
	CHARTIER Germain	Electronique - Automatique
	CHIAVERINA Jean	Biologie - Biochimie - Agronomie
	IVANES Marcel	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PIAU Jean Michel	Mécanique
	PIERRARD Jean Marie	Mécanique
	SABONNADIERE Jean Claude	Informatique Fondamentale et Appliquée
MME	SAUCIER Gabrielle	Informatique Fondamentale et Appliquée
M.	SOHM Jean Claude	Chimie Physique

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

M. FRUCHART Robert	Directeur de Recherche
MM. ANSARA Ibrahim	Maître de Recherche
BRONOEL Guy	Maître de Recherche
CARRE René	Maître de Recherche
DRIOLE Jean	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ionan Doré	Maître de Recherche
MATHIEU Jean Claude	Maître de Recherche
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche

Je tiens à remercier:

Monsieur Gérard VEILLON, Professeur à l'ENSIMAG et l'Institut National Polytechnique de Grenoble, qui m'a fait l'honneur de présider ce jury,

Monsieur Serge GUIBOUD-RIBAUD, Directeur du département Informatique de l'Ecole des Mines de Saint-Etienne, qui a été l'instigateur de ce projet, l'a suivi et ne m'a jamais ménagé ni son aide, ni ses conseils,

Monsieur Robert MAHL, Chef du Service Economique et Financier à la DIELI, qui, lorsqu'il était professeur à l'Ecole des Mines, a su m'initier à l'informatique et m'encourager à faire de la recherche,

Monsieur Jacques KOULOUMIDJIAN, Maître de Conférence à l'Université de Lyon I, qui a bien voulu participer à ce jury,

Monsieur Jean ROHMER, Ingénieur IRIA/LABORIA, qui a marqué son intérêt pour mon travail et accepté de le juger.

Je tiens également à exprimer ma reconnaissance à:

Messieurs J.J. GIRARDOT et F. MIREAUX, Chercheurs à l'Ecole des Mines, pour la contribution active qu'ils m'ont apportée tout au long de ma recherche,

Madame G. LALLICH qui m'a aidé à établir la bibliographie de cette étude,

Messieurs LOUBET et DARLES qui ont assuré avec beaucoup de soin et de gentillesse le tirage de cet ouvrage.

ECOLE NATIONALE SUPERIEURE
des MINES de SAINT-ETIENNE

R E S U M E

Thèse présentée par Michel NAKACHE

pour obtenir le titre de

Docteur-ingénieur

Date de soutenance : 12 juin 1978

Titre : Réalisation d'un noyau de système de gestion de bases de données relationnelles sous APL

Nous présentons dans la première partie de ce document les principaux modèles de gestion de base de données, les objectifs des langages de manipulation de données, puis les précautions que doit prendre le concepteur d'un tel système dans un contexte d'accès concurrentiel,en vue de maintenir la qualité des informations.

Après une brève description du langage APL orienté vers l'interrogation d'une base de données, nous présentons dans la deuxième partie du document différentes extensions à APL proposées ou réalisées pour une meilleure adaptation à un système de gestion de bases de données.

La troisième partie décrit notre réalisation d'un point de vue externe puis interne. Cette réalisation a consisté à adjoindre au système APL un noyau de système indispensable à l'utilisation d'APL comme langage hôte d'un système de gestion de base de données. Nous présentons enfin un langage d'interrogation de base de donnéesdéfini suivant les concepts du modèle relationnel.

RECAPITULATIF

1

QU'EST-CE QU'UN SYSTEME DE GESTION DE BASES DE DONNEES (SGBD)

Ce chapitre définit dans ses grandes lignes les services que doit fournir un SGBD.

Il décrit les principaux modèles de SGBD existants.

Il présente quelques objectifs que doivent remplir les langages de manipulation de Base de Données.

Il présente enfin quelques solutions à adopter en vue de maintenir la qualité des informations enregistrées:

- synchroniser les accès à la Base,
- contrôler le droit d'accès à la Base,
- vérifier la cohérence des informations enregistrées lors d'une mise-à-jour,
- veiller à la fiabilité du système et du matériel.

C H A P I T R E I

1.1 LE ROLE D'UNE BASE DE DONNEES:

Une base de données est une collection d'informations qui a été constituée en vue d'un ou plusieurs buts précis [SGBD1].

Dans une entreprise, par exemple, les informations concernant l'ensemble de son fonctionnement doivent être rassemblées et mises à la disposition de nombreux utilisateurs: cadres, gestionnaires,...

Le fait de partager ces données est le reflet d'une évolution dans la gestion d'une entreprise. Historiquement, chaque nouvelle application engendrait ses propres fichiers et ses propres programmes. La création d'une base de données va à l'encontre de cette façon de faire: elle rend possible la centralisation, la coordination, l'intégration et la diffusion de l'information archivée.

Ceci a pour corollaire:

a. d'améliorer la cohérence des informations,
b. de réduire les redondances,
c. de réduire les efforts de saisie et de mise à jour des informations.

Les systèmes de gestion de bases de données, tâchent de s'approcher de ce point de vue idéal et théorique. Ils offrent à des utilisateurs très différents la possibilité de créer, mettre à jour et interroger une grosse masse structurée de données : une base de données.

1.2 LES FONCTIONS D'UN SGBD:

Lorsque l'on définit une base de données, on modélise une certaine partie du monde réel en essayant de caractériser les entités que l'on manipule: des personnes, des comptes bancaires, des clients,...

De plus, on essaye de caractériser les attributs de ces entités en fonction des problèmes que l'on désire résoudre.

On définira, par exemple, les attributs:

- numéro de compte,
- nom du propriétaire,
- adresse du propriétaire,
- solde du compte,

de l'entité "compte" si l'on désire réaliser la gestion des comptes bancaires d'une banque.

Une fois la structure établie, il faut pouvoir archiver les informations correspondantes, vérifier leur cohérence...

Enfin, la base "remplie", il faut pouvoir l'interroger...
A ce niveau, les utilisateurs de la base désirent rechercher parmi l'ensemble des différentes réalisations ou occurences des entités stockées, certaines d'entre elles, qui répondent à des critères de choix très divers.

Par exemple: *Quels sont les employés qui ont moins de 30 ans et un salaire supérieur à 3400 F.*

Pour répondre à cette question, il serait bon de disposer de l'entité "EMPLOYE" dont deux des attributs seraient "AGE" et "SALAIRE".

Pour ce type de systèmes, on s'accorde à reconnaître trois fonctions principales: [SGBD2]

- fonction de description de la structure de la base:

  définition des entités et de leurs attributs ...

- fonction de manipulation de la base:

  chargement et mise à jour des informations...

- fonction d'utilisation de la base:

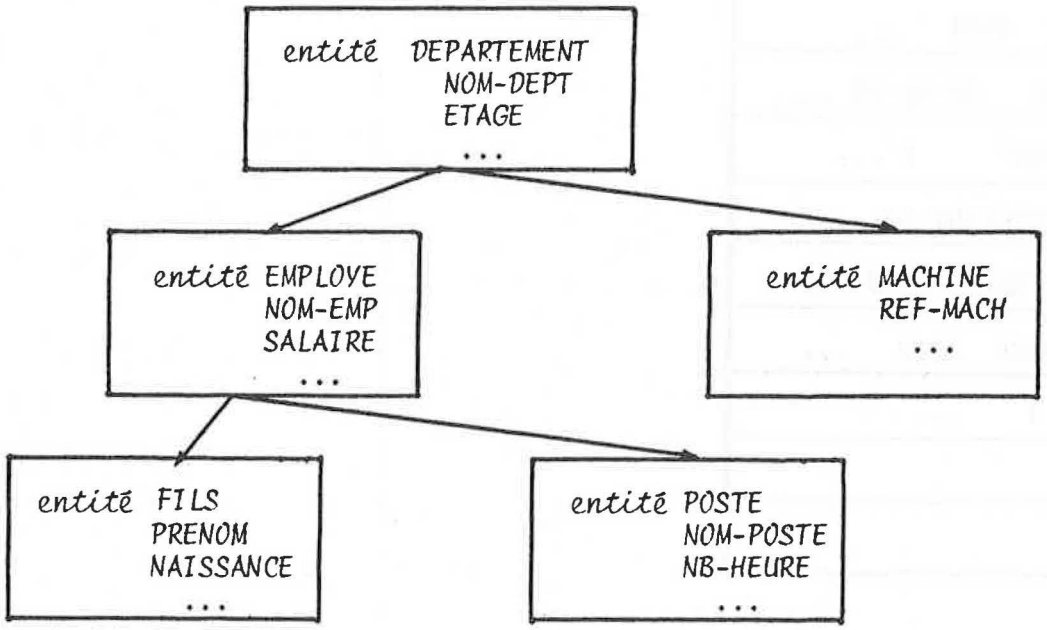
  description des applications à réaliser, interrogation ...

1.3 LES PRINCIPAUX MODELES DE DESCRIPTION DE DONNEES:

1.3.1 STRUCTURES ARBORESCENTES:

Pour des raisons historiques, cette approche est très répandue. Elle correspond aux structures de mémoires secondaires où les informations sont organisées de manière séquentielle [SGBD1].

Une structure arborescente (hiérarchisée) correspond à un arbre où chaque nœud est une entité. Chaque entité est décrite à l'aide d'attributs; ces attributs peuvent eux-mêmes être des entités...
En général, il existe une correspondance: (1 nœud) → (N nœuds) lorsque l'on passe d'un niveau au niveau inférieur.

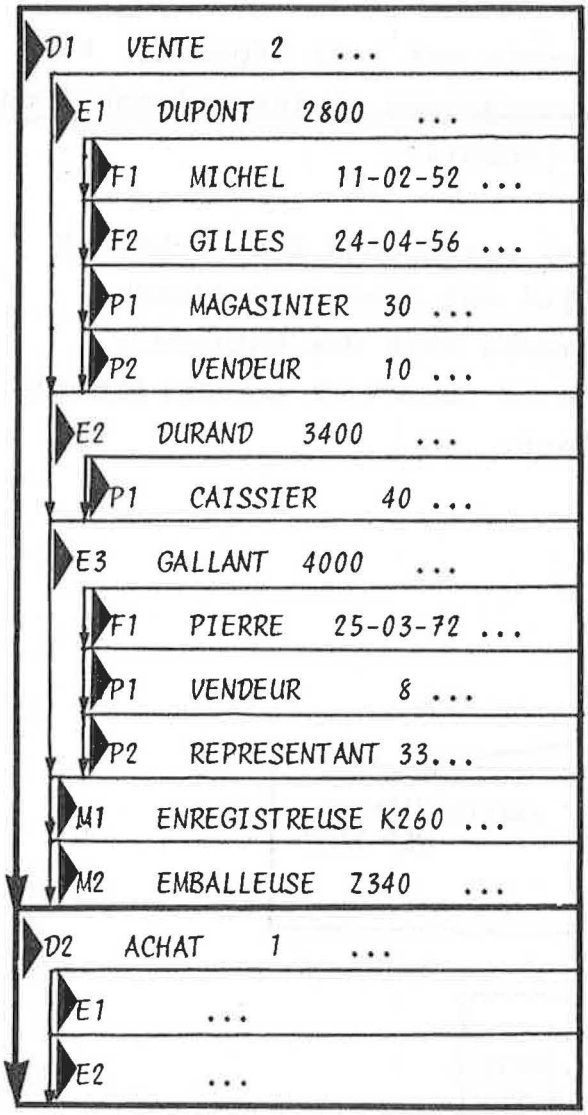


Dans l'exemple ci-dessus, nous faisons correspondre à un DEPARTEMENT plusieurs EMPLOYE(S) et plusieurs MATERIEL(S) tandis qu'à un EMPLOYE, nous faisons correspondre plusieurs FILS et plusieurs POSTE(S) occupés.

Un des principaux systèmes de ce type est I.M.S.[SGBD3]. Dans IMS, une entité est appelée segment.

Les occurences des segments sont organisées de manière hiérarchique. Ainsi, à une occurrence du segment DEPARTEMENT sont attachées toutes les occurrences des segments EMPLOYE du département, à une occurrence

du segment EMPLOYE sont rattachées toutes les occurrences des segments FILS de l'employé...



Occurrences des segments
DEPARTEMENT, EMPLOYE, FILS, POSTE, MACHINE
dans un système type IMS
Di représente une occurrence du segment DEPARTEMENT
Ei représente une occurrence du segment EMPLOYE
Fi représente une occurrence du segment FILS
Pi représente une occurrence du segment POSTE
Mi représente une occurrence du segment MACHINE

Une occurrence d'un segment est la plus petite unité d'information qui peut être impliquée dans un transfert de données.

La caractéristique principale d'un système comme IMS est que chaque occurrence d'un segment n'a de signification que dans le contexte où

elle a été définie. Ainsi, l'occurrence du segment FILS:

"GILLES 24-04-56 ..."

n'a de sens que dans le contexte: fils de DUPONT ... employé au département VENTE ...

Une occurrence ne peut donc exister sans les occurrences qui lui sont hiérarchiquement supérieures.

Les opérations de recherche peuvent s'avérer coûteuses en temps machine. Dans l'exemple précédent, l'accès à un employé ne peut se faire que si l'on connaît son département.

Le système SOCRATE [SGBD4] offre aussi cette possibilité de structure hiérarchisée, connue sous le nom d'"organisation dispersée". Le créateur de la base définit alors ses entités sous forme de blocs imbriqués:

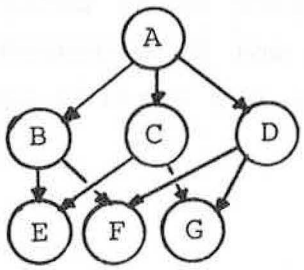
```
entité DEPARTEMENT
début
|       NOM-DEPT
|       ETAGE
|       ...
|       entité EMPLOYE
|       début
|       |       NOM-EMP
|       |       SALAIRE
|       |       ...
|       |       entité FILS
|       |       début
|       |       |       PRENOM
|       |       |       NAISSANCE
|       |       |       ...
|       |       fin
|       |       entité POSTE
|       |       début
|       |       |       NOM-POSTE
|       |       |       NB-HEURE
|       |       |       ...
|       |       fin
|       fin
|       entité MACHINE
|       début
|       |       REF-MACH
|       |       ...
|       fin
fin
```

exemple de structure hiérarchisée définie en langage SOCRATE

Notons enfin, qu'une structure une fois définie, il devient très difficile de la modifier. Si l'on s'aperçoit au bout d'un moment que l'on désire rajouter un attribut à une entité, le SEXE d'un employé par exemple, et si le système le permet, ce ne sera fait qu'au prix d'une gestion d'indirections vers des zones 'overflow' ralentissant de façon notable l'accès aux informations.

1.3.2 RESEAUX:

C'est la forme la plus générale de structure; là aussi, les nœuds du graphe orienté sont des entités, mais contrairement au cas précédent, il peut exister une correspondance (N nœuds) → (1 nœud) lorsque l'on passe d'un niveau au niveau inférieur.



Structure en réseaux
On peut accéder à l'entité F
à partir des entités B ou D

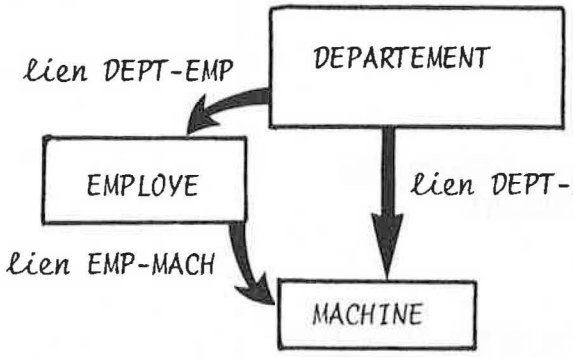
Dans la pratique, cette possibilité permet d'avoir accès aux occurrences d'une entité à partir de différents contextes.

Cette approche correspond à celle du système proposé par le Data Base Task Group (DBTG) de CODASYL [SGBD5].

Dans cette proposition, une entité à laquelle on désire accéder à partir de plusieurs contextes est appelée enregistrement (RECORD). Une liaison entre deux enregistrements peut être exprimée à l'aide d'un lien (SET) entre un enregistrement père (OWNER) et un enregistrement fils (MEMBER). Cette liaison est orientée (hiérarchisée), elle signifie qu'à une occurrence de l'enregistrement père on peut relier plusieurs occurrences de l'enregistrement fils, mais qu'à une occurrence du fils ne correspond qu'une occurrence du père.

```
RECORD NAME IS DEPARTEMENT
  02  NOM-DEPT
  02  ETAGE
      ...

RECORD NAME IS EMPLOYE
  02  NOM-DEPT VIRTUAL SOURCE IS DEPARTEMENT
  02  NOM-EMP
  02  SALAIRE
      ...
  02  FILS OCCURS 10 TIMES
      03  PRENOM
      03  NAISSANCE
          ...
  02  POSTE OCCURS 5 TIMES
      03  NOM-POSTE
      03  NB-HEURE
          ...

RECORD NAME IS MACHINE
  02  NOM-DEPT VIRTUAL SOURCE IS DEPARTEMENT
  02  NOM-EMP  VIRTUAL SOURCE IS EMPLOYE
  02  REF-MACHINE
      ...

SET NAME IS DEPT-EMP OWNER IS DEPARTEMENT MEMBER IS EMPLOYE

SET NAME IS DEPT-MACH OWNER IS DEPARTEMENT MEMBER IS MACHINE

SET NAME IS EMP-MACH OWNER IS EMPLOYE MEMBER IS MACHINE
```

Exemple de description de trois enregistements dans le langage de description de structure proposé par le DBTG



réseau engendré par la description ci-dessus

Le système SOCRATE offre également cette possibilité de structures en réseaux (organisation mixte) au moyen des liaisons inverse et référence. On exprimera ce lien dans l'entité père à l'aide du mot 'inverse tous' et dans l'entité fils à l'aide de 'référence un'.

```
entité DEPARTEMENT
début
|       NOM-DEPT
|       ETAGE
|       ...
|       EMPLOYES inverse tous EMPLOYE
|       MACHINES inverse tous MACHINE
fin

entité EMPLOYE
début
|       NOM-EMP
|       ...
|       entité FILS
|       début
|       |       PRENOM
|       |       NAISSANCE
|       |       ...
|       fin
|       entité POSTE
|       début
|       |       NOM-POSTE
|       |       NB-HEURE
|       |       ...
|       fin
|       DEPARTEMENT référence un DEPARTEMENT
|       MACHINES inverse tous MACHINE
fin

entité MACHINE
début
|       REF-MACHINE
|       ...
|       DEPARTEMENT référence un DEPARTEMENT
|       EMPLOYE référence un EMPLOYE
fin
```

Exemple d'organisation mixte engendrée à l'aide du langage SOCRATE

Ce genre d'organisation permet de résoudre des interrogations complexes, mettant en jeu plusieurs entités comme:

Liste des machines dont le responsable est 'DUPONT' .

1.3.3 LE MODELE RELATIONNEL:

Proposé par Codd [SGBD6] vers 1970, ce modèle est basé sur la théorie mathématique des relations. En mathématiques, une relation peut être définie de la manière suivante:

> *Etant donnés n ensembles D1,...,Dn et un sous-ensemble G du produit cartésien D1×...×Dn, on dira qu'un n-uplet <d1,d2,...,dn> est en relation s'il appartient à G.*
>
> *En fait, une relation est définie par ce sous-ensemble G du produit cartésien D1×...×Dn. Les ensembles D1,...,Dn sont appelés les domaines de la relation; n est le degré de la relation.*

On représente généralement une relation par ce sous-ensemble G, sous forme d'un tableau à n colonnes où chaque ligne représente un n-uplet d'informations.

NOM	N°MATRICULE	PRENOM	AGE
DUPONT	1250	PAUL	30
DURAND	1900	PIERRE	38
.....	.....	.....	..

Exemple de la relation EMPLOYE construite sur les quatre domaines NOM, N°MATRICULE, PRENOM, AGE

A partir de cet exemple, nous pouvons faire les remarques suivantes suivantes [SGBD1]:

- il n'y a pas deux n-uplets identiques,
- l'ordre des n-uplets (des lignes) n'a pas de signification particulière, sauf si l'on désire organiser un domaine suivant un critère particulier (par ordre alphabétique par exemple),
- l'ordre des colonnes n'a pas de signification particulière.

Dans la littérature, il existe bien souvent une ambiguïté sous le terme de 'domaine': ce terme désigne soit un ensemble Di soit sa restriction à G.

Cette ambiguïté devient d'autant plus navrante, si l'on construit une relation sur deux domaines confondus.

Prenons par exemple, la relation COMPOSITION bâtie sur les 'domaines' PIECE, PIECE, QUANTITE, permettant de décrire la façon dont est construite une bicyclette:

PIECE	PIECE	QUANTITE
bicyclette	roue	2
roue	jante	1
roue	pneu	1
jante	rayon	56
...	...	...

Dans ce cas, il faut donner au terme PIECE des noms différents selon le rôle qu'il joue dans la relation.

De façon générale, on dira qu'une relation est bâtie sur des domaines (ensembles Di du produit cartésien) et qu'elle s'appuie sur des constituants ou attributs (termes servant à désigner les colonnes de la relation). Les domaines ne sont pas forcément distincts, les attributs sont toujours distincts (leur nom en particuliers).

Le modèle relationnel permet de représenter aussi bien des structures hiérarchisées que des réseaux. Un point fondamental est que, dans le modèle relationnel, on ne préjuge pas du type d'accès que l'on fera aux constituants.

Ainsi, la structure hiérarchisée vue précédemment peut se définir comme suit:

DEPARTEMENT(NOM-DEPT,ETAGE,EMPLOYE(NOM-EMP,FILS(PRENOM),POSTE(..)),MACHINE(...))

Une relation sous cette forme est dite non-normalisée car elle comporte des domaines qui sont eux-mêmes des relations. Il existe une abondante littérature traitant de la théorie de la décomposition des relations sous forme normalisée [SGBD6][SGBD7]. Elle peut paraître un peu hermétique au lecteur peu averti.

Les premiers systèmes que l'on peut qualifier de relationnels,ont été ceux développés vers 1968 autour de la théorie A.O.V. (Attribute, Object, Value), citons le système TRAMP [SGBD8].

Puis, vers 1970, naissent les propositions de CODD au sujet d'un modèle fondé sur la notion de relation n-aire.

Vers 1973, les propositions de ABRIAL avec 'Data Semantics' [SGBD9] basées sur la notion d'associations binaires donnent naissance à de nombreux systèmes.

DATE	SYSTEME	MODELE	LANGAGE
1967	Relational Data File	binaire	algébrique
1968	TRAMP	binaire	fonction d'accès
1969	STDS	n-aire	algébrique (incomplet)
1969	LEAP	binaire	fonction d'accès
1970	CODD	n-aire	algébrique et prédicatif
1970	CAMBRIDGE	binaire	algébrique
1970	Mac AIMS	n-aire	algébrique
1972	IS/1	n-aire	algébrique
1972	RDMS	n-aire	algébrique
1972	MORRIS	n-aire	prédicatif
1973	GAMMA-0	n-aire	
1974	Data Semantics	binaire	fonction d'accès
1974	INGRES	n-aire	prédicatif
1974	SYNTEX	binaire	prédicatif
1974	System R	n-aire	prédicatif et procédural

tableau extrait de [SGBD1]

1.4 LES LANGAGES DE MANIPULATION, LEURS OBJECTIFS:

Les modèles relationnels proposent des structures très simples pour décrire le contenu d'une base de données. Cette simplicité est d'un grand avantage pour l'utilisateur qui veut inter-agir avec la base de données. Cette possibilité d'inter-action est offerte par le langage de manipulation.

De nombreux langages ont été proposés, en association avec les modèles relationnels. Ces langages sont destinés à être employés par des utilisateurs non spécialistes de la programmation. Pour cette raison, ils doivent être simples.

On notera néammoins, qu'ils nécessitent une certaine connaissance des mathématiques élémentaires ou du moins de la logique.

1.4.1 LA NOTION DE PREDICAT:

Une première caractéristique souhaitable d'un langage de manipulation est d'être non-procédural (le moins procédural possible); c'est-à-dire, qu'il ne doit pas imposer à l'utilisateur la formulation détaillée de l'algorithme d'obtention de l'information désirée. Cette propriété de procéduralité ne peut se mesurer de façon absolue, elle ne peut être obtenue que qualitativement [SGBD1].

En outre, la notion de simplicité n'est pas facile à cerner; se mesure-t-elle à la longueur des programmes ou à la facilité d'apprentissage du langage? Tel utilisateur préférera un langage plus pauvre mais simple à appréhender, tel autre, un langage plus dense, mais nécessitant dans sa maîtrise, un investissement plus important.

Prenons par exemple le produit de deux matrices A et B de dimensions 10×10 exprimé en FORTRAN et en APL:

```
      DO 1 I=1,10
      DO 1 J=1,10
      R(I,J)=0                                  R←A+.×B
      DO 1 K=1,10
1     R(I,J)=A(I,K)*B(K,J)
```

On dira dans ce cas, que FORTRAN est plus procédural qu'APL ou qu'APL est un langage plus dense que FORTRAN.

La poursuite de cet objectif de non-procéduralité a amené de nombreux concepteurs de langages à introduire la notion de prédicat:

> *Soient un ensemble X et une variable x qui parcourt X, on appelle prédicat, une forme d'énoncé P(x) qui devient une proposition vraie ou fausse lorsque x prend une valeur déterminée de l'ensemble X.*

Nous allons illustrer cette définition à l'aide d'un exemple:

- soit X l'ensemble des noms propres d'un dictionnaire,
- soit P(x) :=: "x est un homme",
  - si x :=: 'SOCRATE' alors P('SOCRATE') est vrai,
  - si x :=: 'CERBERE' alors P('CERBERE') est faux.

Nous venons de définir un prédicat à une variable, cette notion peut se généraliser à n variables où les x_i parcourent des ensembles Xi.

Dans l'interrogation d'une base de données, l'utilisateur définit un prédicat ou une suite de prédicats unis par des opérations logiques (non, et, ou, implique,...), les ensembles Xi étant les différents constituants des relations.

Suivant que l'utilisateur s'attache à obtenir une occurrence ou toutes les occurrences vérifiant le prédicat, il fera précéder les variables x_i utilisées des quantificateurs $\exists$ ou $\forall$.

1.4.2 LA FACON DE VOIR LA BASE:

Une seconde caractéristique des langages de manipulation a trait à leur façon de 'voir' la base de données. Si tous voient la base de données comme un ensemble de tableaux où les lignes représentent les caractéristiques des entités décrites, ils n'accèdent pas à ces tableaux de la même façon.

L'idée de base du langage SQUARE [SGBD11],par exemple est de se passer autant que possible des variables et de présenter la sélection des informations dans une relation, plus comme la construction

de sous-ensembles de valeurs que comme résultat de l'examen, ligne par ligne, des relations ainsi que semble le concevoir le langage ALPHA [SGBD12].

Prenons la relation:

EMPLOYE(NOM,PRENOM,SALAIRE,DEPARTEMENT)

et nous allons formuler, à titre d'exemple la question suivante dans les langages ALPHA et SQUARE:

Donner la liste des employés du département JOUET qui gagnent plus de 2000F.

ALPHA:

GET W [NOM] x : EMPLOYE x ∧ ([DEPARTEMENT] x = 'JOUET') ∧ ([SALAIRE] x > 2000)

SQUARE:

get (NOM EMPLOYE DEPARTEMENT,SALAIRE('JOUET', >'2000'))

Cette façon différente de percevoir la base a des répercussions sur la formulation des questions que va poser l'utilisateur, sur la nature des variables qu'il va définir, etc...

1.4.3 CE SONT DES LANGAGES INCOMPLETS:

Enfin, une dernière caractéristique est que la plupart de ces langages ne sont pas complets, ils sont une extension d'un langage hôte, en ce sens que les expressions écrites dans ces langages de manipulation produisent des valeurs qui sont destinées à être traitées par le langage hôte: FORTRAN, COBOL, PL1,...,APL.

1.4.4 QUELQUES OPERATIONS ALGEBRIQUES FONDAMENTALES SUR LES RELATIONS:

Dans ce paragraphe , nous présenterons quelques opérations ayant pour opérandes une ou deux relations.

1.4.4.1 Restriction d'une relation:

Cette opération a pour but de sélectionner les occurrences d'une relation qui vérifient une expression conditionnelle . Une telle expression est obtenue par comparaison d'un constituant de la relation à une valeur appartenant au domaine dans lequel est défini le constituant. Elle peut être formulée de la façon suivante:

- <A θ a> où
- A est un constituant de la relation,
- θ est un opérateur de comparaison: < ≤ = ≥ > ≠,
- a est une valeur appartenant au domaine associé à A.

Une expression conditionnelle peut être formée d'une suite d'expressions conditionnelles séparées par les opérateurs 'et' ou 'ou'.

Nous pouvons définir la restriction d'une relation par le prédicat:

$[\omega]R(A,B,C)$ est défini par: $\forall (x,y,z) \in A\times B\times C$, $\omega(x,y,z)$ vrai

R(A,B,C)			alors $[(A=a1)\wedge(B\neq b2)]R$
a1	b1	c1	a1 b1 c1
a1	b2	c1	a1 b3 c2
a1	b3	c2	
a2	b2	c2	
a2	b1	c1	

1.4.4.2 Projection d'une relation:

La projection d'une relation sur un constituant permet de n'obtenir que les occurrences différentes de ce constituant; cette opération peut être étendue à plusieurs constituants.

Nous pouvons définir la projection d'une relation par le prédicat:

$[A]R(A,B,C)$ est défini par: $\exists x \in A$, $R(x,B,C)$

R(A,B,C)			*alors* [A]R	*et* [B,C]R	
a1	*b1*	*c1*	*a1*	*b1*	*c1*
a1	*b2*	*c1*	*a2*	*b2*	*c1*
a1	*b1*	*c2*		*b1*	*c2*
a2	*b1*	*c1*		*b3*	*c1*
a2	*b3*	*c1*			
a2	*b1*	*c2*			

1.4.4.3 θ-produit de deux relations:

Cette opération permet d'obtenir en vis-à-vis toutes les occurrences de l'une et de l'autre relation dont la θ-comparaison de constituants déterminés des deux relations est vraie. L'opérateur de comparaison peut-être: $<$ $\leq$ $=$ $\geq$ $>$ $\neq$.

Nous pouvons définir un θ-produit par le prédicat:

R(A,B,C)[AθX]S(X,Y) est défini par: $\forall(a,x)\in A\times X$, xθy vrai

R(A,B,C)			*et*	S(X,Y)		*alors*	R[A=X]S				
a1	*b1*	*c1*		*a1*	*y1*		*a1*	*b1*	*c1*	*a1*	*y1*
a1	*b2*	*c1*		*a2*	*y2*		*a1*	*b2*	*c1*	*a1*	*y1*
a2	*b2*	*c1*		*a2*	*y1*		*a1*	*b3*	*c2*	*a1*	*y1*
a2	*b1*	*c2*					*a2*	*b2*	*c1*	*a2*	*y2*
a3	*b2*	*c2*					*a2*	*b1*	*c2*	*a2*	*y2*
a1	*b3*	*c2*					*a2*	*b2*	*c1*	*a2*	*y1*
							a2	*b1*	*c2*	*a2*	*y1*

1.5 LES PROBLEMES D'INTEGRITE:

1.5.1 INTRODUCTION:

L'utilisateur qui accède à une Base de Données doit pouvoir avoir la garantie que les données qu'il va utiliser sont fiables. Le concept d'intégrité recouvre plusieurs notions [SGBD1]:

- partager l'information entre plusieurs utilisateurs de telle sorte que chaque utilisateur ne détruise pas les données d'un autre utilisateur;

- protéger l'information, c'est-à-dire, ne donner l'information qu'à des personnes autorisées. Dans ce cas, on parle parfois de sécurité des données plutôt que d'intégrité;

- maintenir la qualité des informations, autrement dit, veiller à ce que l'information enregistrée soit conforme aux déclarations de l'utilisateur et qu'aucun traitement parasite soit venu altérer ces informations;

- maintenir l'existence de la base quels que soient les incidents techniques qui puissent survenir: pannes du système, défaillance du matériel ou du logiciel...

1.5.2 PARTAGE DE L'INFORMATION ET INTEGRITE:

1.5.2.1 Définition du problème:

Un utilisateur peut utiliser une machine pour une application indépendante, mais il peut aussi vouloir communiquer ou partager de l'information avec d'autres utilisateurs.

Les raisons de ce partage peuvent être diverses:

- utiliser des travaux déjà effectués par d'autres utilisateurs et conservés par le système,

- mettre à la disposition de la communauté des utilisateurs tout ou partie de ses travaux personnels (fichiers ou programmes),

- consulter ou mettre à jour les informations contenues dans une Base de Données: un système de réservation de places ou de vente par correspondance...

Un partage peut être considéré de deux manières:

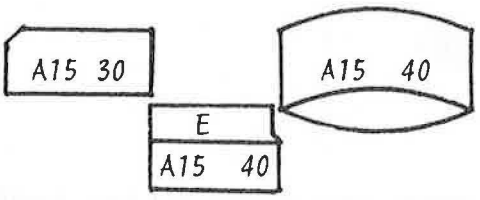
- partage_séquentiel, c'est-à-dire qu'un même objet peut être accédé par deux utilisateurs différents, en des instants distincts,

- partage_simultané, c'est-à-dire qu'un même objet peut être accédé par deux utilisateurs différents, au même instant.

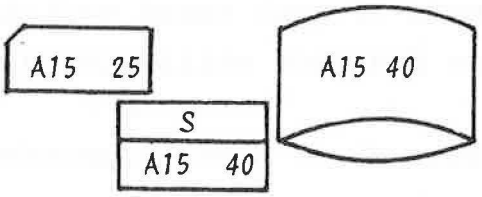
Dans le cas d'un partage simultané, montrons à l'aide d'un exemple comment l'intégrité d'un objet peut être mise en cause si aucune précaution n'est prise:

Soit un processus E traitant des entrées en stock, un processus S traitant des sorties de stock et un fichier A d'état du stock, composé d'un numéro d'article et d'une quantité en stock. Supposons que ces deux processus mettent à jour simultanément le fichier d'état des stocks de la façon suivante: (6étapes)

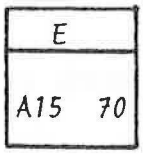
PROCESSUS ENTREE E *PROCESSUS SORTIE S*



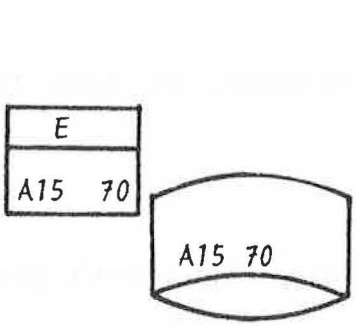
1 *Le processus lit une carte puis transfert dans l'un de ses buffers le contenu de l'enregistrement A15*



2 *Leprocessus lit une carte puis transfert dans l'un de ses buffers le contenu de l'enregistrement A15*

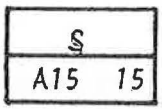


3 *Le processus fait la mise à jour dans son buffer de l' enregistrement A15*

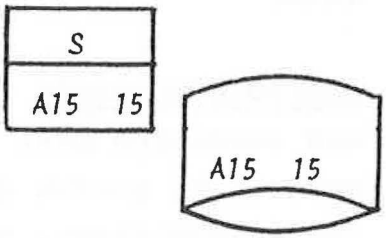


5 Le processus recopie dans le fichier A la valeur de l'enregistrement A15

LES RECTANGLES REPRESENTENT DES ZONES DE MEMOIRE AFFECTEES AUX PROCESSUS, LES CYLINDRES REPRESENTENT LE MEME FICHIER A



4 Le processus fait la mise à jour dans son buffer de l'enregistrement A15



6 Le processus recopie dans le fichier A la valeur de l'enregistrement A15

EXEMPLE DE NON SYNCHRONISATION DE DEUX PROCESSUS PARALLELES

Dans cet exemple, tout se passe comme si la mise à jour effectuée par le processus E avait été ignorée.

De façon générale, le problème se pose dans tout système d'exploitation où il est question de partage de ressources par des processus. Un processus (un programme, un utilisateur), pour s'exécuter a besoin de ressources (mémoire centrale, fichiers, processeurs).

Le problème fondamental dans un environnement d'opérations simultanées est de synchroniser la succession des processus de telle sorte que l'intégrité des données soit respectée.

Dans le cas d'un partage séquentiel, le problème ne se pose pas.

1.5.2.2 Le principe d'exclusion mutuelle:

Une solution qui est souvent adoptée consiste à ce qu'une information en cours de modification ne peut être utilisée par un autre processus que lorsque cette modification est terminée.

Ce principe d'exclusion mutuelle peut s'énoncer de deux façons, suivant le niveau de finesse désiré:

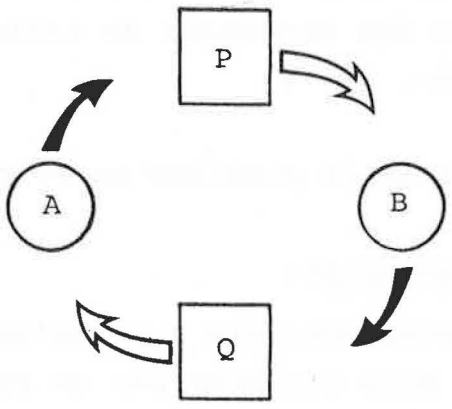
- contrôle exclusif:
  une ressource ne peut être allouée qu'à un seul processus à la fois.

- contrôle partagé:
  une ressource peut être partagée entre plusieurs processus consultant pourvu que ceux-ci soient protégés contre tout processus modifiant; une ressource ne peut être allouée qu'à un seul processus à la fois si celui-ci est modifiant.

1.5.2.3 La situation d'étreinte fatale:

L'utilisation de mécanismes permettant à une ressource de n'être détenue que par un seul processus à la fois peut conduire à une situation d'interblocage (appelée aussi étreinte fatale ou encore embrasse mortelle).

En effet, supposons que deux processus P et Q aient besoin, pour s'exécuter, des ressources A et B et que chacun des deux processus ait déjà acquis une ressource; l'un et l'autre resteront en attente de la ressource manquante.



La situation d'étreinte fatale

P a le contrôle exclusif de A (flèche de A vers P)

Q a le contrôle exclusif de B (flèche B vers Q)

P demande la ressource B mais doit attendre que Q l'ait libérée (flèche de P vers B)

Q demande la ressource A mais doit attendre que P l'ait libérée (flèche de Q vers A)

1.5.2.4 Les_remèdes_possibles,_face_à_une_telle_situation:

Le concepteur du système est confronté à deux faits:

- la nécessité de réaliser un mécanisme de synchronisation afin de préserver l'intégrité des informations,

- la possibilité d'assister à un phénomène d'interblocage s'il réalise le mécanisme de synchronisation.

Devant une telle situation, le concepteur peut adopter trois stratégies, dans la réalisation de son système:

- ignorer_le_phénomène_d'interblocage:
  cette solution n'est pas si absurde, la probabilité d'apparition d'un tel phénomène est très faible et le coût de réalisation d'un moyen de s'en prémunir est important.

  Cette situation sera détectée par quelques utilisateurs, assis à leur console et attendant en vain une réponse du système.

  La tendance actuelle des systèmes étant d'offrir de larges possibilités de dialogue en temps réel avec les Bases de Données, cette probabilité d'apparition d'une situation d'interblocage devient de moins en moins négligeable.

- détecter_le_phénomène_d'interblocage:
  les algorithmes de détection de ce phénomène ne sont pas simples. En supposant que,néammoins,un tel moyen soit réalisé, il faudra détruire un processus qui libèrera ainsi les ressources qu'il s'était attribuées.

  L'abandon d'un ou de plusieurs processus peut menacer l'intégrité de la Base de Données, car ces processus peuvent déjà avoir effectué certaines modifications.

- prévenir_le_phénomène_d'interblocage:
  c'est certainement la meilleure stratégie, elle consiste à concevoir un allocateur_de_ressources (scheduler) qui alloue

celles-ci aux processus dans un ordre tel que le phénomène d'interblocage ne peut se produire. Chaque processus demande alors, avant de commencer à s'exécuter l'ensemble des ressources dont il aura besoin. Ces ressources lui seront attribuées par l'allocateur en bloc ou pas dutout.

Cette méthode présente pour inconvénient d'immobiliser des ressources pour un long moment alors qu'elles ne seront utilisées peut-être qu'un court instant.

1.5.3 PROTECTION DE L'INFORMATION:

Dans un univers de partage d'informations, certaines données doivent pouvoir être protégées contre des utilisateurs indiscrets. C'est un problème très complexe qui outre son aspect technique, revêt un aspect politique et sociologique [SGBD1]: le salaire d'un PDG d'entreprise doit-il être une donnée confidentielle ou accessible à tous?

La plupart des systèmes actuels permettent d'exercer un contôle uniquement au niveau des fichiers:

- soit par identification de l'individu et consultation dans une table de ses droits d'accéder au fichier,

- soit par définition par le propriétaire d'un mot de passe rattaché au fichier; un utilisateur ne peut alors interroger le fichier qu'après avoir fourni le mot de passe.

Ces deux types de protection présentent l'inconvénient majeur dans un cadre de Base de Données, d'être extrêmement grossiers et peu sélectifs.

Bien souvent, le concepteur d'une Base de Données voudra exercer un contrôle plus fin des accès à la Base, par exemple, donner à un utilisateur le droit de consulter:

- tous les constituants d'une relation sauf les salaires si cet utilisateur ne fait pas partie du service comptabilité,

- tous les salaires inférieurs au sien,
- toutes les caractéristiques des employés de son département.

Certains systèmes récents offrent des possibilités allant dans ce sens: citons entre autres le système INGRES [SGBD13]. Dans ce système, le concepteur définit une table de contraintes de sécurité pour chacun des utilisateurs concernés; chaque ligne de cette table contient deux paramètres:

- le nom de l'opération utilisée (insertion, modification, suppression, consultation),
- la liste des informations sur lesquelles l'utilisateur peut interagir ainsi que les conditions qu'il doit remplir. Ce dernier paramètre peut être exprimé sous une forme de prédicat.

Prenons pour exemple, la contrainte de sécurité suivante, définie en langage ALPHA:

GET x : EMPLOYE x ∧ [DEPARTEMENT] x = 'JOUET'

qui exprime qu'un utilisateur ne peut consulter (opération GET) que les caractéristiques des employés du département des jouets.

Lorsqu'un utilisateur interroge la Base, à sa requête est juxtaposée l'expression définissant la contrainte de sécurité. Supposons que l'utilisateur pose la question suivante:

GET W [SALAIRE] x : EMPLOYE x ∧ [ETAGE] x = 2

qui permet d'obtenir les salaires des employés du deuxième étage, cette requête est alors modifiée par le système suivant:

GET W [SALAIRE] x : EMPLOYE x ∧ [DEPARTEMENT] x = 'JOUET' ∧ [ETAGE] x = 2

La réponse du système est alors un sous-ensemble des salaires des employés du deuxième étage: le sous-ensemble de ceux qui appartiennent aussi au département des jouets.

La réponse du système à une requête, si elle est incomplète, n'en est pas moins valide. Par contre, si l'utilisateur fait ensuite des opérations arithmétiques, le résultat obtenu pourra être erroné et l'utilisateur n'en sera pas prévenu: si l'utilisateur désire maintenant

connaître la somme des salaires versés par l'entreprise aux employés du deuxième étage, il obtiendra une réponse fausse puisque seuls les salaires des employés du département des jouets et travaillant au deuxième étage sont pris en compte.

Cette méthode de définition des contraintes de sécurité nous paraît tout de même très élégante.

1.5.4 LA COHERENCE DE L'INFORMATION:

Lorsqu'un utilisateur effectue l'une des trois opérations fondamentales de mise-à-jour: insertion, suppression ou modification, il peut détruire une contrainte d'intégrité que l'on souhaite toujours voir vérifiée, quelque soit la valeur des données.Il existe de nombreux exemples de contraintes d'intégrité:

- le salaire d'un employé est toujours positif,
- un employé ne peut travailler dans plusieurs départements,
- les employés d'un département ne peuvent gagner plus que leur responsable...

Ces contraintes peuvent être mises en œuvre de deux façons générales:

- sous forme de procédures programmées d'accès à la Base en insertion, modification ou suppression. Cette première solution, sans doute peu élégante, car elle demande au concepteur de détailler les actions élémentaires à effectuer,permet un contrôle très fin de la cohérence de l'information validée;

- sous forme d'assertions prédicatives juxtaposées à la définition des relations. Dans ce dernier cas, il n'est pas toujours possible de garantir que l'algorithme d'interprétation de la contrainte prendra en compte tous les cas possibles, surtout s'il y a une cascade de vérifications à faire.

1.5.5 LA FIABILITE DU SYSTEME:

Il faut pouvoir protéger l'information contre un fonctionnement défectueux soit de la machine, soit du système lui-même. Si dans le premier cas, on peut délimiter les informations qui ont été perturbées, dans le second, le problème est beaucoup plus complexe.

Une solution couramment adoptée consiste à prendre à intervalles réguliers, des copies de la Base et à mémoriser l'ensemble des transactions effectuées depuis la dernière copie. Ces différentes précautions permettent de définir des points de reprise du système.

2

POURQUOI APL ?

Après une brève introduction du langage APL, ce chapitre montre les avantages et les inconvénients d'un tel système utilisé pour gérer une Base de Données.

Ce chapitre décrit ensuite un certain nombre d'extensions au système ou au langage APL, qui ont été réalisées ou proposées dans les directions:

- partage simultané d'informations,
- structure complexe d'informations,

afin d'étendre le domaine d'application d'APL aux Bases de Données.

C H A P I T R E I I

2.1 APL LANGAGE DE MANIPULATION DE DONNEES:

APL, langage né dans les années 60, était primitivement destiné à servir d'outil de description d'algorithmes généraux dans les domaines les plus variés, plutôt que de langage de programmation. Sa philosophie tranche donc nettement avec celle des langages utilisés à l'époque [APL1].

APL possède un aspect conversationnel très poussé; un utilisateur, une fois reconnu du système, engage un véritable dialogue avec l'ordinateur; de larges possibilités d'interrogation, de modification interactives sont offertes [APL2].

2.1.1 ORGANISATION DU SYSTEME APL:

Un utilisateur, pour se faire reconnaître du système, doit frapper au terminal son numéro de compte suivi éventuellement de son mot de passe.

Une fois connecté, l'utilisateur dispose d'une aire temporaire de travail appelée bloc, espace ou zone de travail actif (Working Storage). Dans cette zone, il peut stocker des informations, c'est-à-dire, des données ou des programmes.

A l'aide de commandes système, il peut sauver son bloc de travail actif (commande *)SAVE*), copier dans son bloc actif tout ou partie des blocs de travail (commandes *)LOAD* ou *)COPY*) préalablement sauvés.

Un objet *APL* est soit une variable, soit une fonction utilisateur. Une variable est créée lors d'une instruction d'affectation, sans déclaration préalable de type ni de dimension. Sa valeur reste inchangée jusqu'à la suivante affectation à celle-ci. Des commandes système permettent d'obtenir la liste des variables définies ou de détruire certaines (commandes *)VARS*, *)ERASE*).

```
      A ← 1 2 3   (*)
      A
1  2  3
      A ← 'LE PETIT CHAT'
      A
LE PETIT CHAT
```

affectation à la variable A de la valeur 1 2 3 (tableau de trois nombres)

modification de la valeur de A par affectation à celle-ci de la chaine de 13 caractères: LE PETIT CHAT

A un objet *APL* correspond un descripteur; le descripteur d'une variable contient entre autre:

- son type: logique, caractère, entier, réel...
- sa structure:-sans dimension (scalaire),
  - une dimension (vecteur),
  - deux dimensions (matrices),
  - ou plus, suivant les réalisations.

Une variable sans dimension ne peut être indicée, une variable possédant n dimensions doit être indicée à l'aide de n indices, la structure du résultat étant liée à celle des indices.

Les opérations *APL* sont approximativement au nombre de 80, elles opèrent sur un opérande (fonction_de_base_monadique) ou sur deux opérandes (fonction__de_base_dyadique). L'ordre d'évaluation d'une expression *APL* se fait de droite à gauche, sans priorité .

```
      3 × 4
12
      3 × 4 + 2
18
      | ¯2 × 18 ÷ 6
6
```

produit de deux scalaires

la somme est exécutée avant le produit

la barre verticale (monadique) signifie valeur absolue

Lorsque l'on frappe une instruction, elle est aussitôt exécutée, nous sommes en mode_terminal. Si l'on veut créer une fonction utilisateur, il faut passer en mode_édition_de_fonction en frappant le symbole ∇.

Une fonction utilisateur peut être aussi monadique ou dyadique, elle est utilisable au même titre que les fonctions de base.

(*) Les réponses de l'ordinateur sont décalées vers la gauche de six caractères.

```
      ∇R←MOY V                          Création d'une fonction calculant la
[1]    R←(+/V)÷ρV                       moyenne d'un vecteur
[2]   ∇

      MOY 10 20 0                       utilisation de la fonction en mode terminal
10
      3 + MOY 10 20 30                  utilisation de la fonction dans une
13                                      expression APL
```

Les fonctions de bases sont regroupées en trois catégories:

- fonctions_scalaires: elles s'appliquent élément par élément à un (ou deux) opérande(s de même structure), la structure du résultat est identique à celle du (ou des) opérande(s); c'est le cas en particulier des opérations arithmétiques usuelles,
- fonctions_de_restructuration: les valeurs de l' (ou des) oprande(s) n'est pas modifiée, seule la structure est changée,
- fonctions_mixtes: les valeurs et la structure de (ou des) oprande(s) sont modifiées.

Cette description correspond au fond commun de tous les systèmes *APL*.

En outre, sur certains systèmes, il est possible d'intéger à *APL* des sous-programmes écrits en un autre langage. La syntaxe d'appel de ces sous-programmes appelés fonctions_système ne diffère pas de celle des fonctions utilisateurs; les noms de ces fonctions système sont repérables par le fait qu'ils commencent par le symbole □.

2.1.2 MANIPULATION D'UNE BASE DE DONNEES:

Citons pour mémoire quelques fonctions de bases [APL3] qui paraissent plus spécialement utiles dans une orientation Base De Données.

a) fonctions_à_résultat_logique:

non, et, ou, non-et, non-ou, appartient : ~ ∧ ∨ ⍲ ⍱ ∈

comparaisons: < ≤ = ≥ > ≠

```
      'ABCDEDD' ∈ 'AZERT'
                  QSDFF
1 0 0 1 1 1
```

Les éléments de l'opérande gauche, figurent-ils dans l'opérande droit? Une réponse logique est fournie par élément recherché

```
      AGE ← 25 34 26 35 19
      AGE < 28
1 0 1 0 1
```

Comparaison d'une liste d'âges à un nombre d'années

Ces fonctions permettent d'obtenir des filtres logiques.

b) la compression:

L'opérande gauche est un vecteur logique, l'opérande droit a une structure quelconque. L'élément de doite (respectivement la colonne ou la ligne) est conservé lorsqu'à la position correspondante à gauche figure un un:

```
      1 1 0 1 / 1 2 34 51
1   2   51
```

compression d'un vecteur

```
      1 0 1 1 / 'ACEG'
                BDFH
AEG
BFH
```

compression par les colonnes d'un tableau à deux dimensions

```
                'ARMAND '
      1  0  1 ⌿ CLEMENT
                NORBERT
ARMAND
NORBERT
```

compression par les lignes d'un tableau à deux dimensions (la barre horizontale sur le symbole slash signifie qu'il faut effectuer la compression sur le premier indice, à savoir les lignes)

Cette opération est utile lors d'une sélection sur un critère, le filtre étant l'opérande gauche de la compression.
A la question SOCRATE:

¿ NOM de tout EMPLOYE ayant AGE < 28

nous pourrions répondre:

(AGE<28)/NOM

c) la réduction:

C'est l'application d'une fonction scalaire f aux éléments (respectivement aux colonnes ou aux lignes) de l'opérande:

$f / a_1\ a_2\ a_3\ a_4$ effectue l'opération $a_1\ f\ a_2\ f\ a_3\ f\ a_4$

```
      + / 1 2 34 51
```

réduction du vecteur par l'addition

```
      × / 1 2 3
          4 5 6                    réduction d'un tableau par le produit sur
6    120                           les colonnes

      ∨ ≠ 1 1 0 1
          1 1 0 0                  réduction d'un tableau par l'opération 'ou'
1 1 0 1                            sur les lignes
```

Cette opération est utile lorsque l'on veut compter, calculer un nombre d'occurences répondant à un critère, et plus généralement si l'on désire effectuer un calcul algébrique tel que :

somme,produit, maximum, minimum, moyenne,...

d) la réduction d'un produit interne (recherche associative):

Un produit interne est noté f.g où f et g sont deux fonctions scalaires dyadiques. Si f est l'addition et g le produit, un produit de matrices est noté:

```
      1 2 3 +.× 7 8
      4 5 6     7 6                 produit de matrices classique
                5 4
36  32
93  86
```

La réduction d'un produit interne particulier ∧.= appliqué entre un tableau de caractères et un vecteur de caractères, permet de savoir si ce vecteur existe parmi les lignes du tableau:

```
      TABLEAU ← 5 4 ρ 'TOTOTUTUTITITATATETE'     création d'un tableau de
      TABLEAU                                    5 lignes et 4 colonnes
TOTO
TUTU
TITI
TATA
TETE
      TABLEAU ∧.= 'TUTU'                         application du produit interne
0 1 0 0 0
      ∨ ≠ TABLEAU ∧.= 'TUTU'                     le nom TUTU existe dans TABLEAU
1
      TABLEAU ∧.= 'BOBO'                         application du produit interne
0 0 0 0 0
      ∨ ≠ TABLEAU ∧.= 'BOBO'                     le nom BOBO n'existe pas dans TABLEAU
0
```

Le produit interne est utile lors d'une sélection obtenue par comparaison de chaînes de caractères.

A la question SOCRATE:

i ALTITUDE de toute STATION ayant NOM='tignes'

nous pourrions répondre:

(NOM∧.='TIGNES')/ALTITUDE

Si par contre nous désirons savoir si cette station est répertoriée:

∨/NOM∧.='TIGNES'

e) les_fonctions_prend_et_laisse:

Ces fonctions permettent de prendre ou de laisser une partie de l'opérande droit. Suivant le signe des éléments de l'opérande gauche, l'opération a lieu à partir du début ou de la fin:

```
      8 ↑ 'LE PETIT CHAT NOIR'      prendre les 8 premiers éléments du vecteur
LE PETIT
      ¯4↓ 'LE PETIT CHAT NOIR'      laisser les 4 derniers éléments du vecteur
LE PETIT CHAT
              'ABCD'
      ¯3 2 ↑  EFGH                  prendre les 3 dernières lignes et les 2
              IJKL                  premières colonnes du tableau
              MNOP
EF
IJ
MN
```

Ces opérations permettent de sélectionner des occurences particulières répondant à un critère.

A la question SOCRATE:

i ALTITUDE de une STATION ayant NOM='tignes'

qui permet d'obtenir l'altitude de la première station répertoriée sous le nom de TIGNES, nous pourrions répondre:

1↑(NOM∧.='TIGNES')/ALTITUDE

f) fonctions_de_tri_croissant_et_décroissant:

Ces fonctions monadiques rendent pour résultat la liste des indices d'un vecteur trié (par ordre croissant:⍋ ou par ordre décroissant:⍒):

```
      ⍋ 18 ¯17 10 1 10 11          tri numérique croissant
2  4  3  5  6  1
```

Si l'on désire obtenir une liste de noms d'employé par ordre de leur âge croissant, on écrira:

NOM[⍋AGE;] *tableau NOM indicé par:*
- *numéros des lignes par âge croissant,*
- *toutes les colonnes (indice élidé)*

On notera que certains systèmes *APL* permettent en outre de trier des tableaux de caractères, le résultat est alors un vecteur d'indices de lignes du tableau trié.

Pour ces raisons multiples, *APL* possède les opérations élémentaires permettant à un programmeur d'applications de réaliser les fonctions de manipulation et d'utilisation d'une base de données.

2.2 LES LIMITES D'APL DANS UN CONTEXTE DE BASE DE DONNEES:

Dans ce paragraphe, nous présenterons trois limites importantes rencontrées dans les systèmes *APL* classiques; le terme 'classique' signifie ici: fond commun standard des systèmes *APL*. Ces limites rendent l'utilisation de ceux-ci en tant que Systèmes de Gestion de Bases de Données peu efficiente.

2.2.1 LE PARTAGE DE L'INFORMATION:

Dans les systèmes *APL* classiques, seul le partage séquentiel est possible. Ce partage peut s'effectuer soit au niveau du bloc de travail entier (commande *)LOAD*), soit au niveau d'un objet *APL* (commande *)COPY*). Un utilisateur peut ainsi recopier dans son bloc actif, tout ou partie d'un bloc de travail appartenant à un autre utilisateur et préalablement sauvé. Une telle commande ne peut être exécutée qu'après fourniture du mot de passe éventuel attaché au bloc que l'on veut recopier.

2.2.2 L'HOMOGENEITE DES VARIABLES:

Un objet *APL* est un couple 'nom-valeur'; le nom permet de le désigner dans son contexte, la valeur contient soit des instructions (fonction utilisateur), soit des données (variable). C'est ce deuxième ensemble qui nous intéresse ici.

Une variable ne peut avoir des caractéristiques mixtes:

- -soit dans sa structure (tableau à éléments ayant un structure de tableaux),
- -soit dans son type (tableau à éléments numériques et alphabétiques).

De ce fait il est difficile de créer des structures arborescentes ou des relations bâties sur des domaines de types différents.

2.2.3 LA TAILLE DU BLOC DE TRAVAIL ACTIF:

Pour des raisons de simplicité, la plupart des systèmes *APL* actuels ne disposent que d'un processeur de stockage dans une zone réservée de la mémoire centrale: le bloc de travail actif. Or cette zone, qui constitue en quelque sorte l'espace en ligne disponible, ne dépasse que rarement 90 Koctets. Cette limite est vite atteinte dans une application industrielle.

Dans la majorité des systèmes, les commandes ne peuvent s'enchaîner automatiquement, ce qui réduit considérablement l'espace d'exécution d'un programme (limité au seul bloc de travail actif).

Certains systèmes récents ont adopté un mécanisme de mémoire virtuelle pour gérer le bloc de travail [APL8] repoussant à quelques millions d'octets la taille d'un bloc de travail.

Cette intégration peut être faite de deux façons:

- soit par adoption d'une mémoire segmentée; dans ce cas la totalité d'une variable doit être contenue en mémoire centrale,
- soit par adoption d'une mémoire paginée; dans ce cas, seule la partie directement utile de la variable doit être en mémoire centrale.

2.2.4 CONCLUSION:

Un système *APL* ne peut être considéré comme un Système de Gestion de Bases de Données, mais le langage *APL* présente un intérêt certain en tant que langage hôte d'un Système de Gestion de Bases de Données.

2.3 QUELQUES EXTENSIONS PROPOSEES A APL DANS LE SENS SGBD:

2.3.1 INTRODUCTION:

Nous avons vu dans le paragraphe 2.2 trois difficultés qui ne permettent pas aux systèmes *APL* classiques de pouvoir faire office de SGBD:

- accès concurrentiel à des informations impossible,
- pauvreté relative des structures d'information offertes,
- faible capacité de stockage en ligne.

Un certain nombre de constructeurs ou d'équipes de recherche , conscients de ces problèmes, se sont penchés sur ce sujet et ont proposé ou réalisé des solutions destinées à réduire cette inadéquation des systèmes *APL* classiques aux problèmes engendrés par la gestion d'une base de données.

Ces recherches, visant des objectifs différents, se sont orientées dans deux grandes directions:

a) Soit vers l'adjonction au système d'une verrue constituée d'un ensemble de fonctions système; ces fonctions écrites en langage machine mais intégrées à *APL* permettent d'appeler un nouveau processeur de stockage hors du bloc de travail. C'est les cas des systèmes de gestion de fichiers, d'*APL/SV* de l'IBM 360... Le but recherché est essentiellement d'offrir à l'utilisateur la possibilité de partage simultané d'information, le nouveau processeur ayant pour fonction de transférer d'une zone commune à l'ensemble des utilisateurs vers le bloc de travail,l'information mise en commun et vice-versa.
Ces extensions ne modifient en rien la sémantique des opérateurs ni la structure des objets; elles peuvent presque toutes être simulées sur d'autres systèmes, rendant ainsi les applications transportables sans trop de frais.

b) Soit par modification du système lui-même, en définissantde nouvelles structures de variables, de nouvelles fonctions permettant de les créer et en étendant les fonctions classiques

à ces nouvelles structures. Cette dernière approche a pour objectif d'offrir à l'utilisateur une extension dans la structuration des variables et la sémantiques des opérateurs; les applications utilisant ces nouvelles possibilités sont moins facilement transportables, car de telles extensions nécessitent une modification profonde des interprètes *APL*.

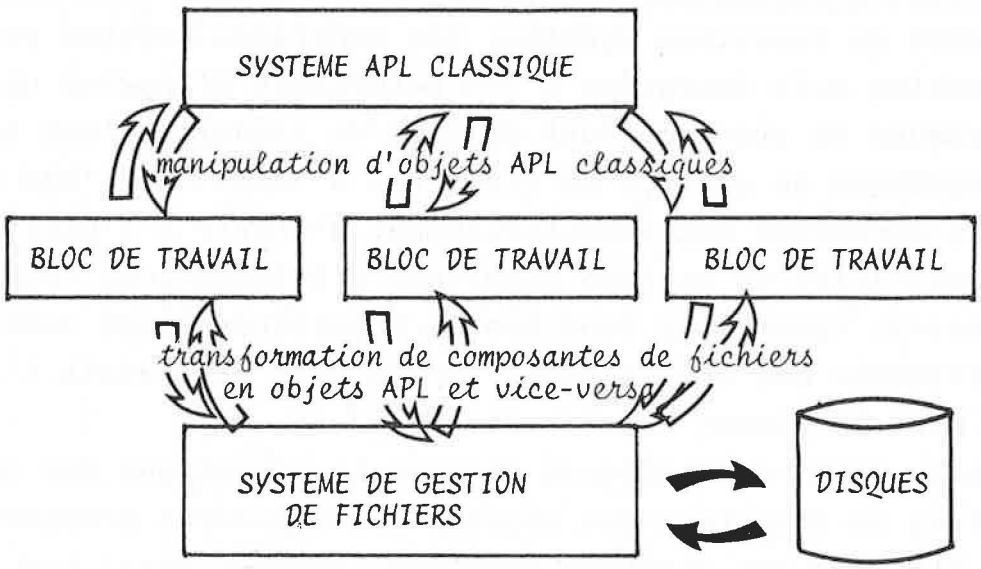
Nous allons présenter quelques unes des solutions proposées ou réalisées.

2.3.2 LE SYSTEME DE GESTION DE FICHIERS APL-PLUS:

Quelques constructeurs se sont accordés sur un standard de système de gestion de fichiers nommé *APL-PLUS* [APL5].

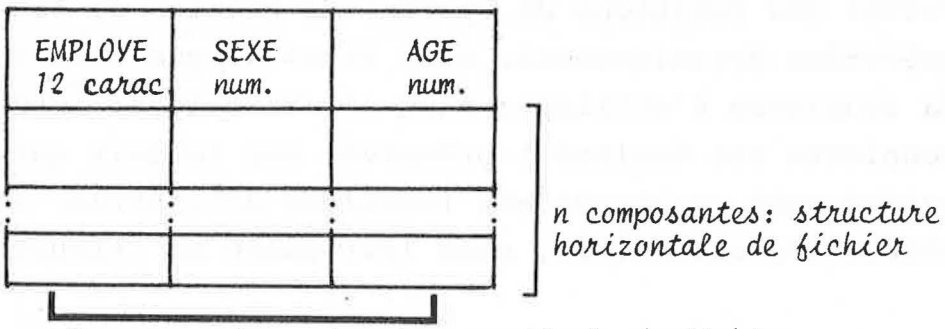
2.3.2.1 Présentation:

A partir d'un ensemble de fonctions système, un utilisateur a la possibilité de gérer un espace de fichiers; c'est un nouveau processeur de stockage, accessible sous *APL*, qui a été adjoint au système. Il n'est activé qu'à partir de programmes utilisateur.



Un fichier *APL* a une structure 'verticale' contrairement aux fichiers classiques COBOL. Une composante de fichier décrit toutes les occurrences d'une caractéristique d'une entité et non l'ensemble des

caractéristiques d'une occurrence d'entité; dans une vue relationnelle, c'est une colonne (un constituant) d'une relation plutôt qu'une ligne. En effet, une composante est l'image d'une variable *APL* et donc doit posséder une structure et un type définis.



L'accès à une composante d'un fichier se fait par l'intermédiaire d'une clé: le numéro de la composante.

Les fonctions d'accès autorisent les opérations habituelles sur les fichiers:

- création, ouverture, fermeture, destruction de fichiers,
- lecture, modification, adjonction, destruction de composantes.

Ces systèmes de gestion de fichiers permettent en outre le partage simultané d'informations; une fonction (*□FHOLD*) permet à un utilisateur d'interdire momentanément aux autres, l'accès à plusieurs fichiers assurant ainsi la synchronisation au niveau de ceux-ci.

< nom du fichier > □FTIE 25	*ouverture d'un fichier sous le numéro 25*
□FHOLD 25	*verrouillage de l'accès au fichier 25*
R ← □FREAD 25 10	*lecture de la composante 10 du fichier 25*
R ← ...	*modification de la variable R*
R □FREPLACE 25 10	*mise à jour de la composante 10 du fichier 25*
□FHOLD 10	*déverrouillage de l'accès au fichier 25*

Enfin, une autre fonction (*□FSTAC*) permet au propriétaire d'un fichier de protéger l'accès à celui-ci. Cette protection est assurée en four-

nissant au système la liste des numéros de compte ayant droit à accéder au fichier en consultation ou en consultation/modification.

2.3.2.2 Avantages:

Toutes les fonctions du système de gestion de fichiers peuvent être exécutées dynamiquement, cela signifie que l'on conserve au langage sa souplesse d'utilisation en n'introduisant aucune déclaration. Cette souplesse est également préservée par le fait que les fonctions manipulées sont analogues aux fonctions *APL*. Enfin, elles manipulent des objets *APL* classiques, sans introduire de structures nouvelles [APL6].

En outre, ces fonctions permettent non seulement le partage simultané d'informations, mais aussi le contrôle de l'accès à celles-ci, pour chacune des opérations autorisées.

Un fichier *APL* est une suite de composantes où chacune peut avoir un type et une structure propres; on peut donc considérer un fichier comme un ensemble d'informations à caractéristiques mixtes: éléments, à savoir les composantes du fichier, à structure et type indépendants.

2.3.2.3 Inconvénients ou limitations:

Le passage d'une application fonctionnant uniquemant dans le bloc de travail à la même, mais utilisant des fichiers (pour des raisons d'importance du volume des données, par exemple),nécessite une modification parfois profonde de celle-ci; il faut, en particulier , insérer des ordres d'accès aux fichiers.

Les problèmes de taille maximale d'une variable deviennent cruciaux si le système *APL* ne dispose pas d'un mécanisme d'adressage virtuel, en effet, la taille d'une composante de fichier peut devenir très importante à cause de l'organisation verticale imposée.

2.3.3 LES VARIABLES PARTAGEES [APL7]:

Le Centre Scientifique IBM de Philadelphie, à partir d'une étude générale sur la communication entre processeurs a appliqué à *APL* la notion de variables partagées.

2.3.3.1 Présentation:

Par l'intermédiaire d'un ensemble de six fonctions système, l'utilisateur peut partager des variables *APL* avec un autre processeur du système d'exploitation, qui peut être un autre utilisateur ou un processeur externe, un système de gestion de fichiers, par exemple.

Utilisée dans le bloc de travail, une variable partagée peut être soit globale, soit locale à une fonction; sur le plan syntaxique, une fois la déclaration de partage effectuée, rien ne permet de la distinger d'une variable ordinaire.

Un partage se fait entre deux processeurs (uniquement), le degré de couplage de la variable étant le nombre de processeurs qui ont offert de partager cette variable.

Une offre se fait à l'aide de la fonction dyadique □*SVO* où l'opérande gauche définit le processeur avec lequel on désire coopérer et l'opérande droit, le nom de la variable que l'on désire partager. Cette fonction a pour résultat le degré de couplage de la variable:

utilisateur n° 1111	*utilisateur n° 2222*	
`      2222 □SVO 'V'`		*offre de partage faite à l'utilisateur 2222 par 1111*
`1`		
	`      1111 □SVO 'V'`	*offre de partage faite à l'utilisateur 1111 par 2222*
	`2`	
`      V ← 15`		
`      V`		*affectation de la variable partagée par 1111*
`15`		
	`      V`	
	`15`	
	`      V ← 3 × V`	*affectation de la variable partagée par 2222*
	`      V`	
	`45`	
`      V`		*la valeur a été modifiée par l'utilisateur 2222*
`45`		

La notion de propriétaire n'est pas définie. Une variable n'est accessible par deux processeurs que si tous deux ont manifesté une offre de partage; aucune prérogative n'existe de l'un des processeurs sur l'autre.

La synchronisation et le contrôle du partage sont définis par chacun des 2 processeurs l'un vis-à-vis de l'autre à l'aide de la fonction □*SVC*, d'une façon originale: un processeur peut définir les droits d'accès de l'autre en fonction de ses propres accès.

Lors d'une commande de sauvegarde du bloc actif ou d'une demande de retrait (□*SVR*), la valeur de la variable partagée est recopiée dans le bloc actif.

SYNTAXE		ACTION	RESULTAT
< *id. processeur* >	□*SVO* < *nom var.* >	*offre de partage*	*degré de couplage*
	□*SVO* < *nom var.* >	*aucune*	*degré de couplage*
< *accès* >	□*SVC* < *nom var.* >	*déf. du contrôle*	*matrice d'accès*
	□*SVC* < *nom var.* >	*aucune*	*matrice d'accès*
	□*SVR* < *nom var.* >	*offre de retrait*	*degré de couplage*
	□*SVQ* < *id. proc.*>	*aucune*	*matrice des noms offerts en partage par le processeur*

2.3.3.2 Avantages:

Sans modification de l'aspect externe de ses programmes, l'utilisateur peut s'ouvrir vers le monde extérieur; ce mécanisme général et souple est par là même très puissant.

2.3.3.3 Inconvénients ou limitations:

La possibilité de créer des structures de données élaborées n'est pas envisagée (ce n'est d'ailleurs pas l'optique de cette extension).

D'autre part, la zone de stockage des variables partagées est de faible capacité (une dizaine de Koctets pour l'ensemble des utilisateurs qui peuvent être 80 en ligne sur un IBM 360/148).

2.3.4 APL MITRA [APL9], LES TABLEAUX DE TABLEAUX [APL10]:

Lors de la conception de ce système, ses auteurs se sont attachés essentiellement à l'aspect volume et structure des informations.

2.3.4.1 Présentation:

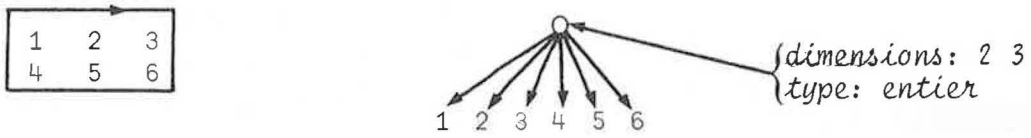
L'aspect gros volume d'informations a conduit les auteurs de ce système à gérer le bloc de travail actif d'un utilisateur à l'aide d'un mécanisme de mémoire virtuelle paginée: la taille du bloc de travail est portée à 4 millions d'octets de mémoire virtuelle, une zone en mémoire centrale de 13 K octets permet au système de stocker temporairement tout ou partie des informations (table des symboles, pile des contextes, variables...) nécessaires à l'exécution d'une opération.

L'aspect structure complexe d'informations a conduit les auteurs de ce système à réaliser une extension des structures de variables classiques dans le sens 'tableaux de tableaux'. Cette extension est réalisée par l'introduction d'un nouveau type de variables: le type pointeur. Une variable de type pointeur peut avoir des éléments de structure et de type différents.

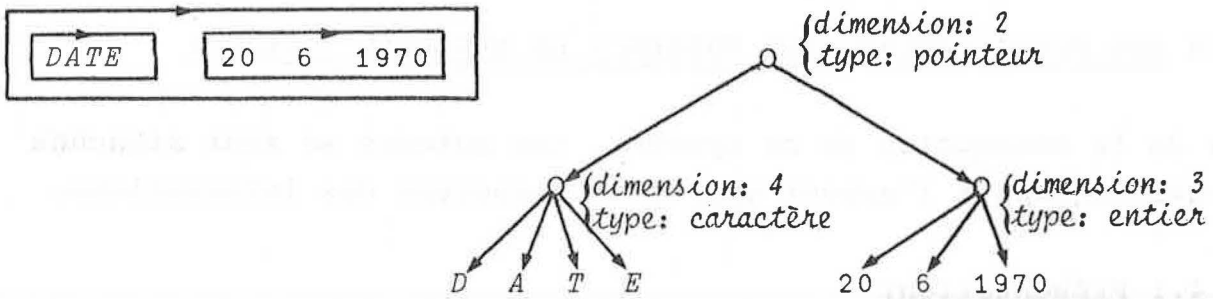
A chaque nœud de la structure hiérarchisée correspond un descripteur. Ce descripteur contient les informations nécessaires à décrire un nœud, c'est-à-dire:

- sa structure (ses dimensions),
- son type (pointeur, logique, entier, réel ou caractère),
- etc...

Seuls les nœuds terminaux correspondent à des variables classiques.



représentation d'une variable APL classique
tableau entier de 2 lignes et 3 colonnes

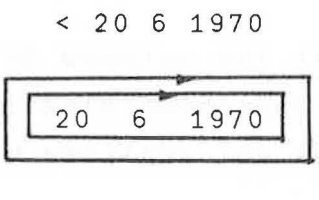


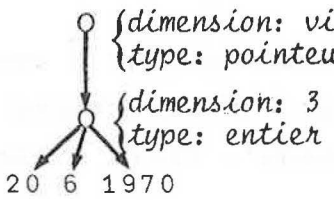
représentation d'un tableau de tableaux de 2 éléments

Afin de créer et manipuler ces nouvelles structures, la sémantique des fonctions *APL* primitives a été étendue et d'autres fonctions primitives ont été définies. Nous en présenterons quelques unes:

fonction 'enfermer' (enclose): `<A`

`< 20 6 1970`





Cette primitive crée un pointeur vers l'objet opérande

fonction 'découvrir' (declose): `>A`

Cette fonction exécute l'opération inverse; elle ne peut se faire que si l'opérande ne possède qu'un élément. Aucune action sur les objets classiques.

fonction 'concaténer' (catenate): `A,B`

`'DATE' , 20 6 1970`

La représentation figure en début de cette page.

Le résultat est un vecteur de 2 éléments de type pointeur. L'opération est équivalente à: `(<A),<B`

fonction 'chainer' (link): `A→B`

L'opérande droit est une variable de type pointeur à laquelle, pour obtenir le résultat, on ajoute un élément pointeur: le pointeur vers l'opérande gauche. L'opération est équivalente à: `(<A),B` .

2.3.4.2 Avantages:

De réelles structures nouvelles sont proposées; mais reste à mesurer leur efficacité dans la pratique.

Le système,grâce à son mécanisme de gestion de mémoire, repousse dans de larges limites les problèmes inhérents au volume important des données à traiter lors d'une application de type base de données.

2.3.4.3 Inconvénients ou limitations:

Le partage d'information n'est pas abordé, mais il faut remarquer que ce problème ne semble pas pas faire partie des objectifs de ces auteurs.

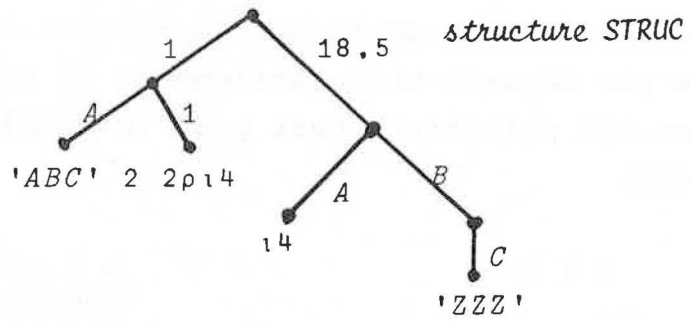
2.3.5 STRUCTURES ARBORESCENTES DE L'UNIVERSITE DE LAVAL:

Cette extension dans le domaine des structures de données propose de créer et manipuler des structures arborescentes à l'intérieur d'une variable *APL* classique: un vecteur de caractères.

2.3.5.1 Présentation:

Un ensemble de fonctions système permet d'utiliser cette extension. Les branches d'une structure sont repérées par des noms (nombre ou suite de caractères); lorsque l'on désire atteindre un nœud, on spécifie le chemin emprunté par l'intermédiaire de la liste des noms des branches traversées. Si un nœud est terminal, il supporte de l'information.

```
      SGET (STRUC;1 1)
1  2
3  4

      SGET (STRUC;18.5;B;C)
ZZZ
```



Une fonction (*SINIT*) permet d'initialiser une variable comme structure, une autre (*SPUT*) permet d'ajouter une sous-structure à une structure ou de supprimer une sous-structure.

SYNTAXE	ACTION
S←SINIT 0	*création d'une structure dans la variable S*
R←SGET (S;<chemin>)	*obtention de la sous-structure spécifiée de S*
V SPUT (S;<chemin>)	*modification ou création de la sous-structure spécifiée*
SPUT (S;<chemin>)	*effacement de la sous-structure spécifiée de S*
R←STEST(S;<chemin>)	*validation de l'existence du chemin spécifié dans S*
R←SBR S	*liste les branches de S*
SIMP S	*imprime sous forme schématique tous les niveaux de S*

2.3.5.2 Intérêts:

Cette extension s'apparente à la création d'un système de gestion de fichiers 'multi-clés' où les ordres de lecture et d'écriture sont appelés *SGET* et *SPUT*.

Ce mini-système a l'avantage d'être totalement indépendant de l'interprète *APL* sous-jacent et ne surcharge donc absolument pas le processus d'exécution de ce dernier; d'autre part, cette extension est réalisable sur toute autre machine, à peu de frais.

Enfin, cette extension présente un intérêt certain dans la réalisation de petites applications s'appuyant sur des données faibles en volume, mais à structures arborescentes (*).

2.3.4.3 Inconvénients ou limitations:

L'accès concurrentiel n'est pas abordé, mais il faut remarquer que ce problème de fait pas partie des objectifs des auteurs.
D'autre part, une variable contenant une structure n'étant pas reconnue en tant que telle par le système *APL*, peut être détruite ou altérée par mégarde très facilement; il suffit de lui appliquer une fonction *APL* primitive. Ceci pose le problème de la sécurité des informations:

```
3 ↑ S

S[1 2 3 4] ← 'TOTO'
```

Si S est une structure, les résultats de ces opérations fournissent n'importe quoi ou bien risquent de détériorer les informations contenues dans la structure; ces informations sont soit des pointeurs à l'intérieur de la variable APL soit des valeurs entrées dans la structure par l'utilisateur.

2.3.6 CENTRES SCIENTIFIQUES IBM DE MADRID ET DE PALO-ALTO:

Ces deux équipes de recherche proposent séparément des extensions des structures internes des objets *APL* (structures d'arbres) et un jeu de fonctions primitives nouvelles permettant de les créer; c'est une ex-

---

(*) Lors d'un séjour à Laval (Québec), l'auteur a eu l'occasion d'utiliser ces structures pour simuler l'extension proposée par GULL & JENKINS alors au centre de Palo-Alto [APL13].

tension du langage de base. Les deux propositions sont assez similaires dans leur finalité: permettre à l'utilisateur de créer des structures d'information mixtes et complexes.

Les deux communications tentent ensuite de généraliser la sémantique de quelques fonctions *APL* classiques à ces nouvelles structures (voir l'exemple du ρ monadique à la page suivante).

Nous allons présenter brièvement quelques aspects de ces propositions:

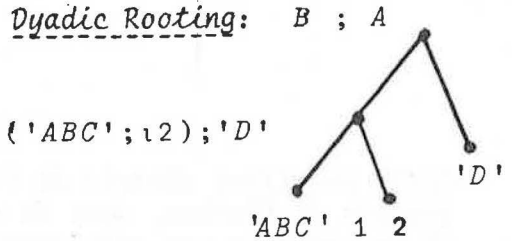
2.3.6.1 Centre de Madrid [APL12]:

Les structures arborescentes sont représentées par des arbres où chaque feuille contient de l'information classique. Aucune dimension n'est associée aux nœuds non-terminaux. Une structure classique est un arbre de profondeur zéro.

Monadic Rooting: ;*A*

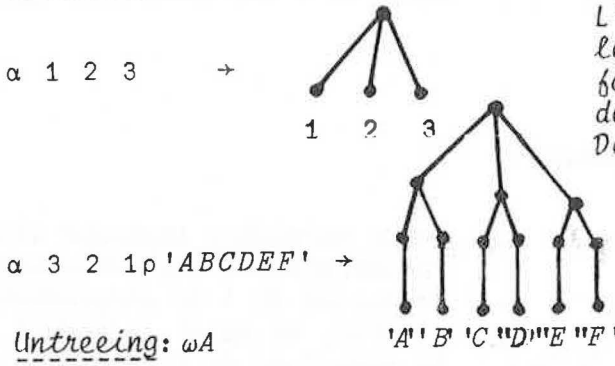
Cette primitive accroît de 1 la profondeur de l'arbre.

Dyadic Rooting: *B* ; *A*



Cette primitive accroît de 1 la profondeur des arbres opérandes puis les concatène par leur racine.

Treeing: α*A*

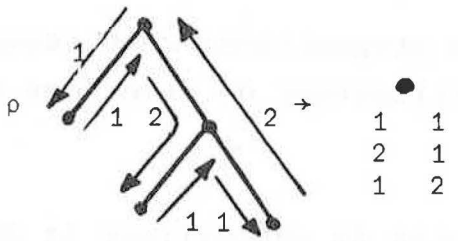


L'argument droit est de niveau 0; le résultat est un arbre de profondeur ρρA ayant tous les nœuds de niveau I-1 à un degré (ρA)[I]. Degré signifie: nombre de branches.

Untreeing: ω*A*

L'argument droit doit être de niveau supérieur à 0 et ses éléments des scalaires homogènes.

Shape: ρA



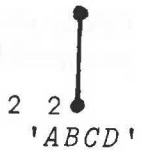
Extension du vecteur des dimensions classique. Le résultat est une matrice de dimensions (N,2), N étant le nombre de feuilles, représentant la profondeur de chaque feuille.

2.3.6.2 Centre de Palo Alto [APL13]:

Les structures arborescentes sont là aussi représentées sous forme d'arbres, chaque feuille contient une information à structure classique; mais contrairement au modèle précédent, les nœuds possèdent une structure (des dimensions).
Une structure classique est représentéepar un arbre de profondeur 0.

Seal: < monadique

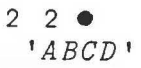
```
A←<2 2ρ'ABCD'
```



Cette primitive accroît de 1 la profondeur de l'arbre, le nœud introduit ayant une structure de scalaire (rang nul).

Unseal: > monadique

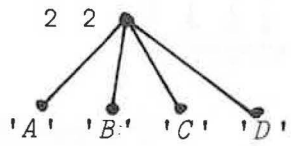
```
B←>A
```



Cette primitive décroît de 1 la profondeur de l'arbre, dans la mesure où la racine a une structure de scalaire ou a une dimension égale à 1.

Simple-raise: ↑ monadique

```
C←↑2 2ρ'ABCD'
```



Cette primitive regroupe séparément les éléments d'une structure en accroissant de 1 la profondeur de l'arbre, le nœud introduit prend la structure de l'opérande.

Lower: ⍋ monadique

```
D←⍋C
      2 2 ●
        'ABCD'
```

```
E
        3
      1   1
    2   3   2 3
  'AB' 'ABC'  1 2 3
              4 5 6
```

Cette primitive décroît de 1 la profondeur de l'arbre dans la mesure où tous les sous-arbres de niveau 1 ont:
- une structure de scalaire ou ont une dimension égale à 1,
- sont de type homogène, c'est-à-dire sont soit des arbres, soit des tableaux numériques, soit des tableaux alphanumériques.

```
F←⍋E
        3
      2   3   2 3
    'AB' 'ABC'  1 2 3
                4 5 6
```

Catenate: , dyadique

```
(<⍳3),<2 2⍴'ABCD'
        2
      3   2 2
    1 2 3  'AB'
           CD
```

Extension de la concaténation classique. Les deux opérandes doivent avoir des dimensions compatibles et un type homogène, c'est-à-dire:
- type arbre,
- type tableau numérique (classique),
- type tableau alphanumérique.

```
(<⍳3),2 2⍴'ABCD'      INTERDIT
```

Type non homogène (arbre , tableau)

```
(⍋⍳3),<2 2⍴'ABCD'     INTERDIT
```

Type homogène (arbre , arbre) mais dimensions non compatibles.

Itemwise and Leafwise:

```
F
        2
      3   2 2
    1 2 3  'AB'
           CD
```

Gull et Jenkins proposent d'étendre la sémantique des fonctions APL classiques de la façon suivante:
Soit une fonction monadique ou dyadique f
- f s'applique à la racine de l'arbre (itemwise),
- ¨f s'applique jusqu'aux feuilles de l'arbre (leafwise).

```
¨⍴F
        2
      1   2
    3     2 2
```

```
⍴F
  ● 1
  2
```

2.3.6.3 Avantages et inconvénients

Ces deux propositions, du point de vue de la mise en œuvre pratique nécessiteraient d'intervenir au niveau de chacune des opérations et fonctions usuelles, afin de rajouter des tests de validité et des extensions qui pénaliseraient fortement le temps d'exécution de ces fonctions dans leur usage courant.

Les problèmes de partage d'information et de taille des variables ne sont pas abordés dans ces deux propositions.

Enfin, pour ces quelques raisons, ces propositions présentent plus un intérêt pédagogique que pratique.

2.3.7 CONCLUSION:

De cette présentation de quelques extensions proposées à *APL* dans les domaines du partage de l'information et de la structuration des données, nous pouvons retenir qu'un grand nombre d'idées ont été émises et que ces deux domaines constituent aux yeux de nombreuses équipes, un handicap certain au développement d'*APL* dans les applications de type bases de données.

3

LE MODELE PROPOSE - ASPECT EXTERNE

Ce chapitre présente le côté utilisateur de notre réalisation:

- création de structures relationnelles,
- présentation d'un langage d'interrogation de base de données.

C H A P I T R E I I I

3.1 INTRODUCTION:

3.1.1 NOTRE BUT:

Notre objectif est d'offrir un noyau de système indispensable à l'exploitation 'grandeur nature' d'une base de données industrielle plutôt que de fournir la possibilité d'illustrer des concepts de cas d'écoles.

Ce point nous a amené à développer plutôt l'aspect système: structures internes de données, partage de l'information, plutôt que l'aspect langage externe de manipulation très complet et sophistiqué.

Le Modèle Relationnel nous a séduit par le fait qu'il propose une structure d'information (les tableaux) et des opérations synthétiques sur les tableaux très voisines de la philosophie du langage *APL*.

Signalons dès maintenant que le modèle proposé est loin d'englober toutes les notions définies dans un modèle relationnel, ce n'est qu'un noyau minimal adjoint au système *APL*.

3.1.2 NOS HYPOTHESES:

Nous disposions d'un interprète du langage *APL* classique avec ses limitations:

- pas de possibilité de partage simultané,
- pas de structures complexes de données,
- faible volume des données en ligne lors d'une exploitation.

Pour des raisons de simplification du processus d'interprétation du système, nous ne voulions pas introduire:

- de structures internes nouvelles directement manipulables par le système (arbres, réseaux...),
- de fonctions primitives nouvelles ni d'extension des fonctions classiques .

Pour des raisons de transportabilité des programmes (ou de convertibilité), nous ne voulions pas introduire:

- de fonctions primitives nouvelles,

- d'ordres explicites d'entrées-sorties à insérer tout au long des programmes.

3.1.3 HISTORIQUE DU PROJET:

Notre réalisation s'est effectuée en trois temps:

a) modification de l'interprète *APL* afin de lui adjoindre un processeur de stockage d'objets de grande taille hors du bloc de travail. Ce processeur s'appuie sur un mécanisme de mémoire virtuelle paginée. Plutôt que de remplacer le processeur de stockage préexistant, il nous a semblé préférable, pour des raisons d'ordre politique et économique, d'adjoindre ce second processeur au précédent;

b) adjonction d'un ensemble de fonctions système permettant de définir des structures nouvelles d'information compatibles avec le système *APL* et s'appuyant sur ce nouveau processeur de stockage;

c) réalisation sous forme de fonctions *APL*, d'un supra-langage d'interrogation du type 'langage algébrique'.

Ces trois parties nous ont amenés à définir les notions de:

a) lien dans le bloc de travail vers une variable stockée en mémoire virtuelle appelée aussi variable virtuelle (aspect partage d'une BDD);

b) relation bâtie sur des domaines qui ne sont autre chose que des variables virtuelles (aspect BDD relationnelle);

c) opérations relationnelles sur une relation: sélection ou projection, ou sur deux relations: produit.

3.2 VARIABLES VIRTUELLES:

Une courte présentation des fonctions système introduites, permettra au lecteur de saisir aisément le mécanisme; une description plus détaillée figure en annexe (pages 106 à 112).

3.2.1 PRESENTATION:

Une variable virtuelle est stockée dans l'espace virtuel de son propriétaire (créateur); un lien vers cette variable peut être obtenu dans le bloc de travail. Supposons que cet utilisateur ait le numéro de compte 1111:

```
      □VCRT 'B A'                  Création dans l'espace virtuel 1111 d'une
0                                  variable virtuelle de nom A et d'un lien B
                                   dans le bloc de travail vers cette variable;
                                   un compte-rendu de l'opération est fourni.
```

Les variables virtuelles sont dites 'publiques', c'est-à-dire qu'elles sont accessibles par l'ensemble des utilisateurs. (*)

Un deuxième utilisateur de numéro de compte 2222 peut demander l'obtension d'un lien vers une variable virtuelle de l'espace 1111:

```
      □VLIB 1111                   liste des variables virtuelles de l'espace
A                                  de numéro 1111
      □VTIE 'C 1111A'              obtention d'un lien C vers la variable virtuelle
0                                  de l'espace virtuel 1111
```

Les deux utilisateurs peuvent manipuler la variable A sous les noms respectifs des liens B et C.

```
   Bloc de travail          Bloc de travail
   de l'utilisateur         de l'utilisateur
        1111                     2222

   B ← 1 2 3 4 5                          affectation de la variable par 1111
                                 C
                                          lecture de la variable par 2222
                          1  2  3  4  5
                                 C ← C÷2  modification de la variable par 2222
      B
                                          lecture de la variable par 1111
0.5   1     1.5   2     2.5
```

(*) un autre mécanisme (paragraphe 3.3.2) permet de déclarer une variable privée (non partageable).

Dans sa syntaxe d'utilisation, un lien est rigoureusement identique à une variable classique.

La règle de stockage du résultat d'une opération est la suivante:

- lors de l'exécution d'une opération, si l'un des opérandes est virtuel, le résultat est stocké en mémoire virtuelle (ce qui réduit considérablement les cas de *WS FULL ERROR*),
- lors d'une affectation, si l'opérande affecté a fait l'objet d'une déclaration (lien), l'affectation a lieu dans l'espace virtuel.

Les commandes système relatives aux objets *APL* (*)VARS*,*)ERASE*...) n'ont d'effet que sur les liens ; des fonctions système permettent d'exécuter des opérations analogues sur les variables virtuelles; toutefois, seul le propriétaire d'une variable peut la détruire.

```
      VAR← 3 5⍴'A AA B BB C     '   création d'une variable de 3 lignes et 5
      VAR                           colonnes
A AA
B BB
C
      ⎕VCRT VAR                     création des trois variables AA BB et C et
0  0  0                             de leurs liens A B et C
      )VARS                         liste des variables et liens du bloc
A    B    C    VAR
      ⎕VLIB 1111                    liste des variables virtuelles de l'espace
AA
BB
C
      )ERASE A C                    effacer les liens A et C
      )VARS                         liste des variables et liens du bloc
B    VAR
      ⎕VERA 'BB'                    détruire la variable virtuelle BB
0
      ⎕VLIB 1111                    liste des variables virtuelles de l'espace
AA
C
      ⎕VERA ⎕VLIB 1111              destruction de toutes les variables virtuelles
0  0                                de l'espace
      B
VS DEFN ERROR                       erreur: le lien référence une variable
      B                             virtuelle qui n'existe plus
      ^
```

Cette courte session *APL* est destinée à familiariser le lecteur avec la manipulation des variables virtuelles.

□VCRT M virtual create	*création de variable virtuelle*
□VTIE M virtual tie	*ouverture de variable virtuelle*
□VERA M virtual erase	*destruction de variable virtuelle*
□VLIB N virtual library	*liste des variables virtuelles*

3.2.2 CONCLUSION:

Une procédure de synchronisation complète (équivalente à *□FHOLD* du système de gestion de fichiers *APL-PLUS*) manque à ce système; mais une telle possibilité ne peut être conçue qu'au niveau du moniteur de temps partagé supervisant *APL*.Cette facilité sera réalisée lors d'une prochaine version du moniteur.

Une synchronisation sommaire est actuellement réalisée: lorsqu'un utilisateur exécute une affectation , il bloque automatiquement, le temps de l'affectation, l'accès à cette variable virtuelle en cours d'affectation.

3.3 LES RELATIONS:

Une description détaillée figure en annexe (pages 113 à 125).

3.3.1 PRESENTATION:

Le système propose à ses utilisateurs désireux de créer une base de données, un ensemble de fonctions système qui permettent de créer des structures hétérogènes de données. Une arborescence à deux niveaux peut être ainsi créée:

- une racine appelée relation, qui ne supporte pas d'information ,
- des feuilles appelées constituants qui supportent de l'information structurée en tableaux indépendant

Ces structures sont créées dans l'espace virtuel de l'utilisateur, indépendamment de l'interprète *APL*. Une fonction permet ensuite de créer dans le bloc de travail un lien vers les constituants qui sont alors assimilables à des variables virtuelles.

Ces constituants sont manipulables séparément à l'aide du langage *APL*; toutefois, dans certains cas, il sera bon de définir en *APL* un supra-langage permettant de manipuler les constituants à partir du nom de leur relation. Un tel exemple de langage d'interrogation est proposé au

paragraphe 3.4.

Les constituants d'une relation sont partageables ou non; un contrôle de l'accès en lecture ou lecture/modification est défini par le propriétaire de la relation, au niveau de l'ensemble de ses constituants. Si aucun contôle n'est spécifié, la relation est non-partageable.

A tout moment, le propriétaire d'une relation peut ajouter ou supprimer des constituants à une relation. Cette facilité contredit le fait que dans un modèle relationnel, tous les constituants d'une relation ont même nombre de lignes (ce qui n'est plus vrai si l'on ajoute un constituant alors vide) mais présente l'intérêt de pouvoir modifier la structure d'une relation sans avoir à la recréer puis la charger.

Illustrons ces quelques concepts à l'aide d'une session *APL*:

```
      ⎕RSET 'EMPLOYE(NOM,SEXE,MATRICULE,AGE)'        création d'une relation
0

      ⎕RLIB 1111                                     liste des relations de 1111
EMPLOYE

      ⎕RLIB '1111 EMPLOYE'                           liste des noms des constituants de la relation
NOM                                                  EMPLOYE de 1111
SEXE
MATRICULE
AGE

      '1111 EMPLOYE' ⎕RGET 'A AGE'                   obtention d'un lien A vers le constituant AGE de la
0                                                    relation EMPLOYE de 1111

      'EMPLOYE' ⎕RGET ⎕RLIB 'EMPLOYE'                obtention des liens vers les constituants de EMPLOYE
0  0  0  0

      )VARS                                          liste des variables (classiques) et liens
A         AGE      MATRICULE NOM       SEXE
```

Remarquons que A et AGE sont deux liens vers le même constituant AGE de EMPLOYE.

```
      ((SEXE=1)∧AGE<30)/NOM                          question: nom des employés
DUPONT                                               homme de moins de 30 ans.
ISIGNY
```

```
      □RUPD 'EMPLOYE(PRENOM)'                    adjonction du constituant PRENOM à
0                                                la relation EMPLOYE

      'EMPLOYE' □RARW 2222 2244                  autorisation d'accès en lecture-
                                                 modification donnée aux utilisateurs
                                                 2222 et 2244

      □RARW 'EMPLOYE'                            liste des accès à la relation EMPLOYE
2222  2244                                       en lecture-modification

      'EMPLOYE' □RERA 'MATRICULE'                suppression du constituant MATRICULE
0                                                dans la relation EMPLOYE

      □RLIB 'EMPLOYE'                            liste des noms des constituants de
NOM                                              la relation EMPLOYE
SEXE
AGE
PRENOM

      □RERA 'EMPLOYE'                            destruction de la relation EMPLOYE
0
```

La fonction □RARO a la même syntaxe que □RARW, mais s'occupe des accès en lecture.

□RSET <chaîne> REL □RSET DOM	} set relation	création d'une relation
□RUPD <chaîne> REL □RUPD DOM	} update relation	addition de constituants à une relation
REL □RGET DOM	} get link	obtention d'un lien vers un constituant
□RLIB N □RLIB REL	} library	liste des relations d'un espace liste des constituants d'une relation
□RERA REL REL □RERA DOM	} erase	destruction d'une relation destruction de constituants d'une relation
□RARW REL REL □RARW V	} read-write access	liste des accès en modification à une relation modification de cet accès
□RARO REL REL □RARO V	} read only access	liste des accès en consultation à une relation modification de cet accès

3.3.2 UNE AUTRE VISION DU SYSTEME:

Ce système n'exécute aucune vérification automatique d'intégrité entre les différents constituants d'une relation. Ces constituants, en particulier peuvent n'avoir absolument aucun rapport entre eux, ils peuvent être considérés comme des variables virtuelles indépendantes.

Une relation sera alors considérée comme un groupe ou un annuaire de variables, la signification logique de ce groupe étant laissée à l'imagination de l'utilisateur.

Dans ce dernier cas, des primitives de contrôle de l'accès à ces variables sont offertes au propriétaire de celles-ci.

3.3.3 CONCLUSION:

Cette extension souple et générale, permet de répondre aux problèmes posés par les systèmes *APL* classiques.

Cette extension peut être directement utilisée pour créer une base de données: après avoir défini la structure des relations, l'utilisateur peut remplir la base en complétant indépendamment chacun des constituants à l'aide des fonctions *APL* primitives (la concaténation en particulier). Pour d'autres utilisateurs, moins confirmés, il sera préférable de fournir un ensemble de fonctions *APL* de saisie de données, permettant de valider les informations rentrées. L'aspect conversationnel du langage *APL* révèlera alors toute sa puissance.

3.4 PRESENTATION D'UN SUPRA-LANGAGE DE MANIPULATION:

Dans ce paragraphe, nous allons illustrer sur un exemple, comment les opérations algébriques sur les relations définies au chapitre I-4-4, permettent d'interroger une base de données (*).

Considérons la base définie par les trois relations:

- EMP(NOM,SAL,MGR,DPT)qui permet de décrire un employé à l'aide des caractéristiques: le nom, le salaire, le nom de son responsable, le nom du département où il travaille;

- VENTE(DPT,ART,VOL)qui permetde décrire le volume des ventes à l'aide des caractéristiques: un département, un article, le volume de sa vente;

- LOC(DPT,ETA)qui permet de fournir l'étage où se trouve un département.

(*) Cet exemple a été programmé, il figure en annexe (pages 131 à 136); le supra-langage défini ici est constitué de fonctions *APL* qui s'appuient sur l'extension réalisée (pages 128 à 136).

3.4.1 LA SELECTION:

La syntaxe de cette opération est la suivante:

'[< *domaines désirés* >]' *GET* '< *relation* >[< *expression conditionnelle* >]'

Q1: Trouver les noms et salaires des employés du département des jouets:

'[*NOM,SAL*]' *GET* '*EMP*[*DPT*=''*JOUET*'']'

Q2: Trouver les noms des employés et de leur responsable qui travaillent au département des jouets et gagnent 2000 F ou moins:

'[*NOM,MGR*]' *GET* '*EMP*[(*DPT*=''*JOUET*'')∧(*SAL*≤2000)]'

Q3: Trouver le volume des ventes des poupées dans le département des jouets:

'[*VOL*]' *GET* '*VENTE*[(*DPT*=''*JOUET*'')∧(*ART*=''*POUPEE*'')]'

Dans certains cas, il sera nécessaire de créer une variable intermédiaire:

Q4: trouver le nom et le salaire du responsable de l'employé DUPONT:

X←'[*MGR*]' *GET* '*EMP*[*NOM*=''*DUPONT*'']' *détermine le nom du responsable, puis*
'[*NOM,SAL*]' *GET* '*EMP*[*NOM*=*X*]'

3.4.2 LA PROJECTION:

La syntaxe de cette opération est la suivante:

'[< *domaines désirés* >]' *GET* '< *relation*>[< *domaines projetés* >]'

Q5: Trouver la liste des responsables de l'entreprise:

'[*MGR*]' *GET* '*EMP*[*MGR*]'

Q6: Trouver les noms et salaires des responsable de l'entreprise:

X←'[MGR]' GET 'EMP[MGR]' permet d'obtenir les noms des responsables, puis '[NOM,SAL]' GET 'EMP[NOM=X]'

3.4.3 LE PRODUIT DE DEUX RELATIONS:

La syntaxe de cette opération est la suivante:

*'[<dom. de rel1>][<dom. de rel2>]' GET '<rel1> *< rel2>[<θ-produit>]'*

Q7: Trouver les étages où travaillent les employés:

*'[NOM][ETA]' GET 'EMP * LOC[DPT=DPT]'*

Q8: Trouver les noms et salaires des responsables de l'entreprise (cette question est identique à Q6):

*'[NOM,SAL][]' GET 'EMP *EMP[NOM=MGR]'*

Cette expression fournira la même réponse que Q6 mais la façon de l'obtenir est différente.

3.4.4 COMPOSITION D'OPERATIONS:

La syntaxe de cette composition d'opérations est la suivante:

*'[<dom.rel1>][<dom.rel2>]' GET '<rel1> *<rel2>[<θ-produit>][<sélection>]'*

Q9: Trouver le salaire du responsable de l'employé DUPONT (cette question est identique à la question Q4):

*'[NOM,SAL][]' GET 'EMP *EMP[NOM=MGR]['' DUPONT''=NOM]'*

Cette expression appelle deux remarques:

- on ne peut inverser l'expression conditionnelle (entre crochets après l'étoile) car il faut indiquer que dans la relation temporaire obtenue, DUPONT est comparé au deuxième champ NOM, celui provenant de la deuxième relation EMP;

- si le résultat de cette expression est le même que celui obtenu à partir de l'expression Q4, il fait intervenir des manipulations beaucoup plus importantes, puisqu'il crée une relation intermédiaire.

3.4.5 INTRODUCTION D'OPERATIONS ARITHMETIQUES;

Il est aisé, en *APL*, de définir quelques opérations arithmétiques. Nous donnerons ici quelques exemples d'utilisation ainsi que les listings des opérations utilisées:

Q10: Trouver les noms des employés qui gagnent plus que n'importe quel employé du département des jouets:

```
X←MAX '[SAL]' GET 'EMP[DPT=''JOUET'']'
'[NOM]' GET 'EMP[SAL>X]'
```

donne le maximum des salaires du département JOUET, puis ...

```
     ∇ R ← MAX V
[1]  R ← ⌈/⍎ V
     ∇
```

Dans cette fonction monadique (argument V), l'opérateur de réduction associé à la fonction scalaire dyadique 'maximum' (⌈) et opère sur l'argument V préalablement transformé en un vecteur de numérique (fonction exécute:⍎).

Q11: Trouver la moyenne des salaires perçus par les employés du département des jouets:

```
MOYENNE '[SAL]' GET 'EMP[DPT=''JOUET'']'
```

```
     ∇ R ← MOYENNE V
[1]  R ← (+/V)÷⍴V ← ⍎ V
     ∇
```

Dans cette fonction monadique (argument V), on transforme V en numérique (⍎), puis on divise la somme des éléments de V (+/V) par le nombre d'éléments (⍴V).

Q12: Compter le nombre d'employés du département des jouets:

```
COMPTER '[NOM]' GET 'EMP[DPT=''JOUET'']'
```

```
     ∇ R ← COMPTER V
[1]  R ← 1↑⍴V
     ∇
```

Dans cette fonction monadique (argument V), on prend la première composante du vecteur des dimensions (⍴V) de V. La première dimension d'une variable APL est son nombre de lignes.

4

LE MODELE PROPOSE - QUELQUES ASPECTS INTERNES

Ce chapitre présente le côté réalisation pratique de l'extension proposée.

C H A P I T R E I V

Rappelons que la première partie de notre réalisation a consisté à adjoindre au sein de notre interprète *APL* un processeur de stockage d'informations en mémoire virtuelle.

4.1 PRESENTATION SUCCINTE DE LA MEMOIRE VIRTUELLE:

Un espace virtuel est une aire logique adressable de très grande taille. Cet espace supposé peu rempli (plein de trous) est replié sur une mémoire physique de dimension beaucoup plus restreinte , le processeur devant être capable, à partir d'une adresse virtuelle de retrouver l'adresse réelle (physique) de l'information désirée (fig.1).

Dans notre réalisation, un espace virtuel est une aire de 2^{31} doubles-mots, soit quelques 8 milliards d'octets (de caractères). Un champ d'adresse virtuelle de 31 bits (2 mots) permet de référencer un double-mot (4 octets). Le processeur permet de gérer plusieurs espaces virtuels.

Un espace virtuel est replié sur un espace physique de stockage (mémoire secondaire), l'unité de stockage sur cet espace physique étant la page (512 mots)(fig.1).

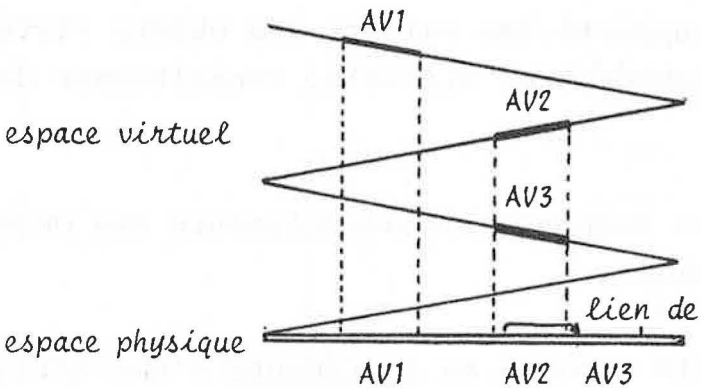


FIG.1 REPLIEMENT D'UN ESPACE VIRTUEL SUR L'ESPACE PHYSIQUE

Les collisions éventuelles lors du repliement (pages virtuelles de même adresse physique) sont gérées à l'aide de liens de chaînage dans les entêtes des pages.

Une page pleine de zéros est considérée comme vide et donc non existante, elle n'occupe, de ce fait, aucune place physique.

Le processeur virtuel dispose en mémoire centrale d'une zone tampon, découpée en un certain nombre de cadres de manœuvre, dans lesquels sont transférées des pages à la demande.

Le processeur virtuel est chargé de lire ou d'écrire le double-mot dont l'adresse virtuelle est fournie. Il découpe cette adresse en un numéro de pli, un numéro de page dans le pli, puis un indice du double-mot dans la page. Si la page désirée ne figure dans aucun des cadres de manoeuvre, une page est éjectée vers la mémoire secondaire suivant un algorithme LRU (la page la plus anciennement référencée est éjectée). La page désirée est ensuite transférée vers le cadre ainsi libéré.

Le procédé est classique, il est équivalent dans son principe à celui utilisé par la mémoire du système SOCRATE [SYS2].

Dans notre réalisation, le processeur de stockage en mémoire virtuelle installé dans l'interprète, s'appuie sur une zone de cadres unique pour l'ensemble des utilisateurs et sur deux espaces virtuels communs à tous les utilisateurs:

- un espace données qui supporte les valeurs des objets virtuels stockés: variables et temporaires virtuels, constituants des relations,

- un espace structures qui supporte les descripteurs des objets précédents et des relations.

Chacun de ces deux espaces a été découpé en incréments d'une taille de 2^{16} doubles-mots (soit 256 K octets). Chaque espace contient donc 2^{15} (soit 32768) incréments. Le numéro d'un incrément, dans une adresse virtuelle est inscrit dans la partie poids-fort de cette adresse:

adresse du premier incrément (base 16) : '0000 0000
adresse du deuxième incrément(base 16) : '0001 0000
... ...
adresse du dernier incrément (base 16) : '7FFF 0000

4.2 ORGANISATION DE L'ESPACE DONNEES:

Lorsqu'un objet virtuel est créé, un incrément donnée lui est alloué afin de permettre le stockage de sa valeur. De ce fait, l'ensemble des utilisateurs dispose de 32767 objets virtuels (ce nombre correspond au nombre des incréments de l'espace données).

La gestion des incréments libres et occupés se fait à l'aide d'une table d'allocation située dans le premier incrément de l'espace structures. Lors de la recherche d'un incrément libre, la table est scrutée de façon cyclique(fig.2).

Cette méthode d'allocation des incréments a été préférée à la méthode consistant à chaîner entre eux les incréments libres d'une part et les incréments occupés d'autre part pour deux raisons:

- l'initialisation de l'espace virtuel (chaînage de tous les incréments alors libres) est évitée,

- les incréments inoccupés, ne contenant aucune information, n'ont pas d'existence physique et on minimise ainsi l'encombrement de la mémoire secondaire.

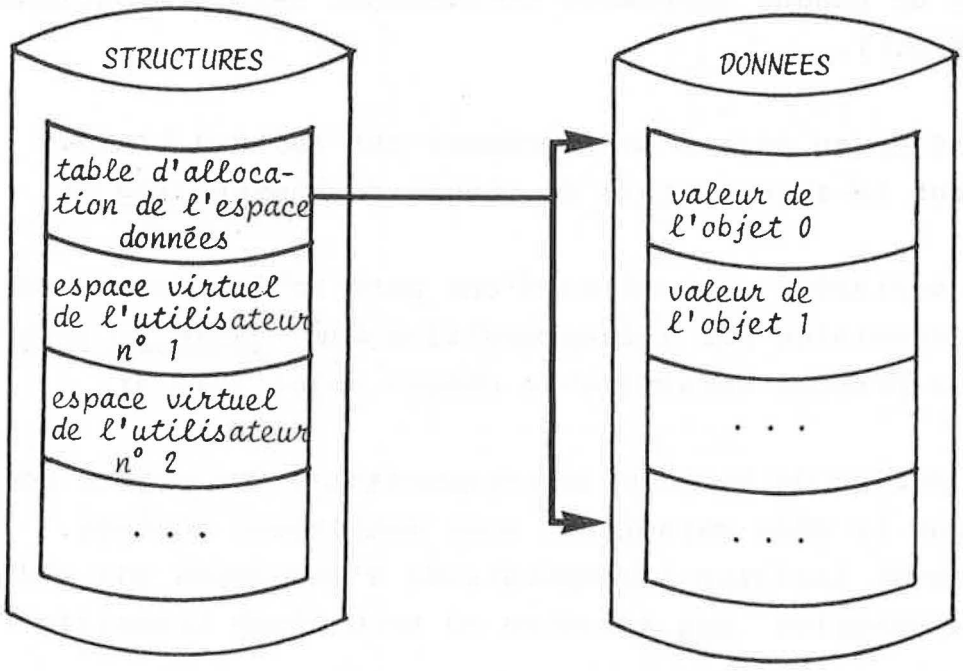


FIG.2 ORGANISATION GENERALE DES DEUX ESPACES VIRTUELS EN INCREMENTS

4.3 ORGANISATION DE L'ESPACE STRUCTURES:

Cet espace est aussi découpé en incréments de la façon suivante (fig.2):

- incrément 0: table d'allocation des incréments de l'espace données,

- incrément i ($i \neq 0$): zone virtuelle de l'utilisateur de numéro de compte i, lui permettant de stocker les descripteurs des objets virtuels qu'il a créés; par la suite, nous appellerons souvent cette zone: espace virtuel de l'utilisateur i.

Tous les utilisateurs de numéro de compte compris entre 1 et 32767 disposent donc d'un espace virtuel et ont accès au processeur de stockage adjoint au système *APL*.

Laissons dorénavant l'incrément 0 qui joue un rôle particulier et détaillons la structure d'un incrément utilisateur.

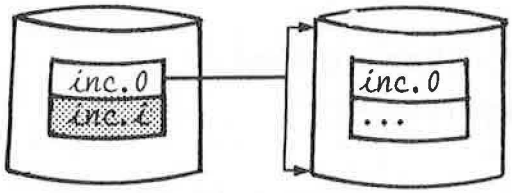
Chaque incrément est découpé en articles de vingt mots permettant de stocker les descripteurs des objets virtuels.

Les premiers articles de chaque incrément contiennent des informations propres au système (fig.3).

L'allocation des articles au sein d'un incrément est faite à l'aide d'une table située dans la zone système de chaque incrément (fig.3).

Les descripteurs des variables virtuelles d'une part et des relations d'autre part, sont accessibles par l'intermédiaire d'une table de hachage (hashcoding) sur le premier caractère de chaque objet (fig.3).

Les articles de même numéro de hachage (correspondant à des objets dont les noms commencent par le même caractère) sont doublement chaînés. Ce chaînage amont et aval facilite les opérations d'insertion par ordre alphabétique et de suppression des articles au sein d'une liste(fig.4).



rappel de la figure 2
la partie grisée est détaillée figure 3

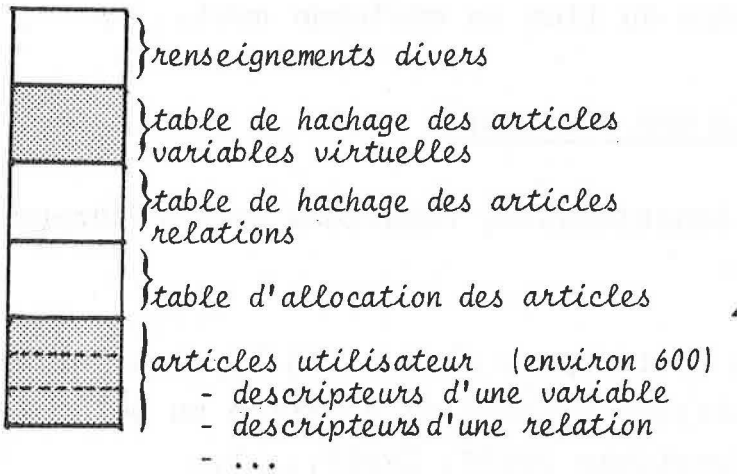


FIG. 3 ORGANISATION D'UN INCREMENT UTILISATEUR
la partie grisée est détaillée figure 4

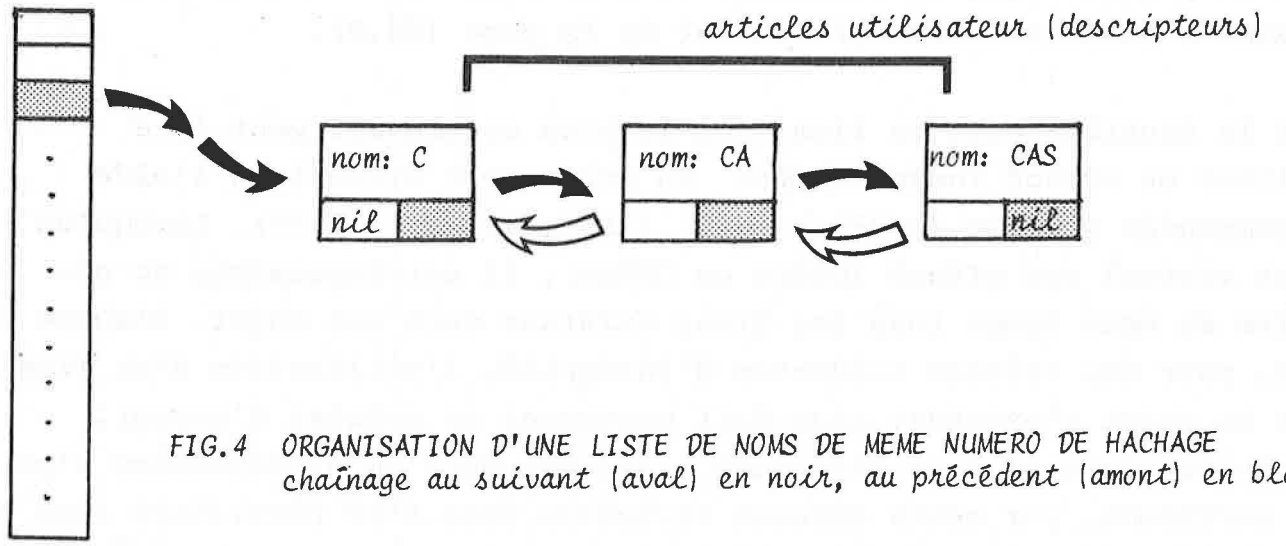


FIG. 4 ORGANISATION D'UNE LISTE DE NOMS DE MEME NUMERO DE HACHAGE
chaînage au suivant (aval) en noir, au précédent (amont) en blanc

La recherche de l'article correspondant à un objet virtuel référencé par son nom est faite en deux temps (fig.4):

- accès par l'intermédiaire de la table de hachage à la tête de liste de numéros de hachage donnés,

- recherche séquentielle de l'article désiré dans la liste sélectionnée, par l'intermédiaire du lien de chaînage aval.

4.4 LA DESIGNATION DES OBJETS: LE NOM UNIVERSEL:

Les objets virtuels: variables, constituants, relations sont référencés par l'utilisateur de deux façons:

- soit par leur nom externe et l'espace virtuel où ils sont définis; c'est le cas des fonctions système de création ou de modification de structures (fonctions □*VCRT*, □*RSET*,...),

- soit par un lien; c'est le cas d'opérations *APL* entre objets virtuels.

Dans le premier cas, la recherche de l'article désiré est faite suivant l'algorithme défini en haut de la page (§4.3).

Dans le deuxième cas, ce lien dans le bloc de travail peut être dupliqué ou effacé indépendamment du processeur virtuel, à l'aide de commandes système (*)COPY*, *)LOAD*, *)DROP*, *)SAVE*, *)CLEAR*). Lorsqu'un objet virtuel est effacé (□*VERA* ou □*RERA*), il est impossible de détruire en même temps tous les liens existant vers cet objet. D'autre part, pour des raisons évidentes d'intégrité, l'utilisation d'un lien vers un objet n'existant plus doit provoquer un constat d'erreur. Or stocker dans le lien l'adresse virtuelle de l'objet référencé n'est pas suffisant, car cette adresse virtuelle peut être réutilisée lors de la création d'un nouvel objet virtuel.

La solution adoptée pour résoudre ce problème a été d'introduire la notion de nom universel ou historique [SYS1].

Un nom universel est un moyen de désigner un objet de façon unique.

Si cet objet est détruit, son nom universel ne sera plus jamais réutilisé à la désignation d'un nouvel objet.

Dans notre réalisation, un nom universel est un ensemble de 64 bits composé de deux parties:

- un champ contenant l'adresse virtuelle du descripteur de l'objet (32 bits),

- un champ contenant la valeur d'un compteur du système incrémenté lors de chaque création d'objet virtuel, ce champ est de 32 bits, offrant ainsi environ 4 milliards de possibilités. Ce champ pourrait être rempli d'une autre façon, par la valeur de l'horloge du système au moment de la création ou par toute autre méthode permettant d'assurer que le nom est bien universel.

Chaque objet virtuel possède donc un nom universel, un lien vers cet objet contient son nom universel. L'algorithme d'accès à un objet virtuel à partir d'un lien devient extrêmement simple (fig.5):

- lecture de l'article référencé par le champ adresse virtuelle,

- comparaison du numéro interne inscrit dans l'article et du deuxième champ du lien.

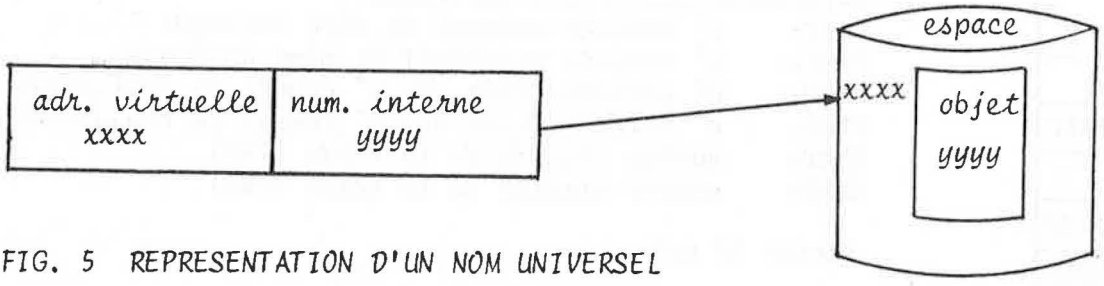
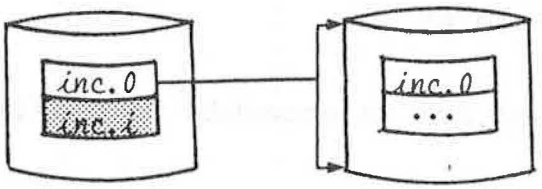
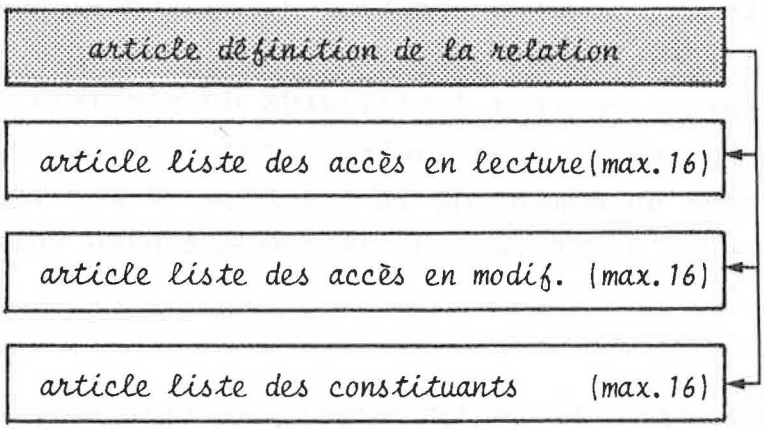


FIG. 5 REPRESENTATION D'UN NOM UNIVERSEL



rappel de la figure 2
la partie grisée est détaillée
ci-dessous



la partie grisée
est détaillée
ci-dessous

FIG.6 ORGANISATION DU DESCRIPTEUR D'UNE RELATION

ETAT	REQ
PTFI	NBFI
NOM	
UNIVERSEL	
SUIV	PREC
-	-
PTRW	PTRO
NBRW	NBRO

ETAT: article type relation
REQ : verrou (accès concurrentiel)
PTFI: n° article liste des constituants
NBFI: nombre de constituants
NOM : nom externe de la relation (12 caractères)
UNIVERSEL: numéro interne unique
SUIV: n° article suivant de même hashcode
PREC: n° article précédent de même hashcode
PTRW: n° article liste des n° compte en modification
PTRO: n° article liste des n° compte en consultation
NBRW: nombre (taille de la liste PTRW)
NBRO: nombre (taille de la liste PTRO)

taille 20 mots

FIG.7 DETAIL DE L'ARTICLE DEFINITION D'UNE RELATION

4.5 DESCRIPTION D'UNE RELATION:

Lors de la création d'une relation par un utilisateur (□*RSET*), un ensemble de quatre articles dans l'incrément structure de celui-ci est alloué; cet ensemble constitue le descripteur de la relation. Ces quatre articles contiennent les caractéristiques suivantes (fig.6):

- définition de la relation:

  cet article contient entre autre le nom de la relation, son numéro interne unique, les pointeurs vers les relations suivante et précédente de même numéro de hachage...(fig.7);

- liste des fils:

  cet article contient la liste des numéros des articles décrivant les constituants de la relation;

- liste des accès en lecture:

  cet article contient la liste des numéros de compte des utilisateurs ayant le droit de consulter les constituants de la relation. Cette liste est modifiable par le propriétaire de la relation à l'aide de la fonction □*RARO*;

- liste des accès en écriture:

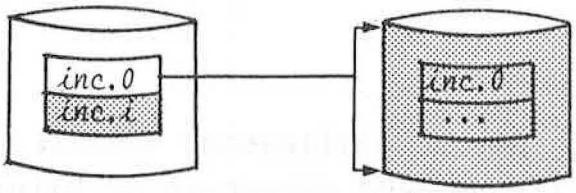
  cet article contient la liste des numéros de compte des utilisateurs ayant le droit de consulter et de modifier les constituants de la relation. Cette liste est modifiable par le propriétaire de la relation à l'aide de la fonction □*RARW*.

4.6 DESCRIPTION D'UN CONSTITUANT:

Lors de la création d'un constituant (□*RSET* ou □*RUPD*), un article dans l'incrément structure de l'utilisateur est alloué. Cet article constitue le descripteur du constituant.

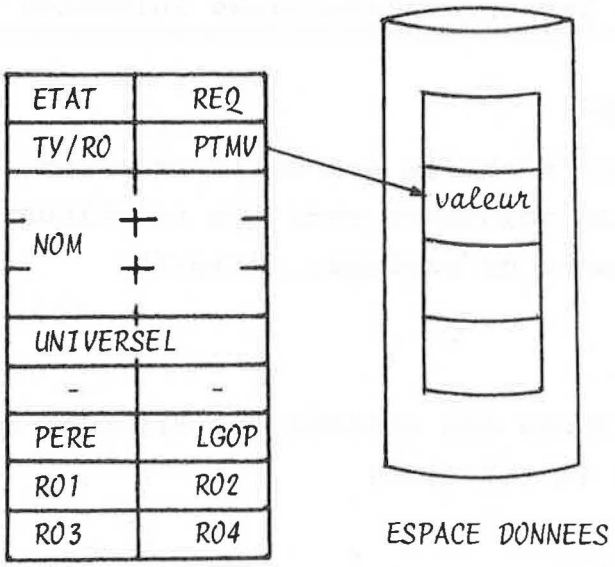
Lors de l'affectation d'une valeur à ce constituant, les caractéristiques de l'objet affecté sont inscrites dans cet article (fig.8).

Une variable virtuelle possède un descripteur analogue à celui d'un constituant.



ESP. STRUCTURES ESP. DONNEES

rappel de la figure 2
la partie grisée est détaillée
ci-dessous



ESPACE DONNEES

ETAT: article type constituant
REQ : verrou (accès concurrentiel)
TY : type de l'objet (indéfini, caractère, logique, entier, réel simple ou double)
RO : rang (nombre de dimensions) de l'objet
PTMV: n° incrément espace données
NOM : nom externe du constituant (12 caractères maximum)
UNIVERSEL: numéro interne unique
PERE: n° de l'article relation
LGOP: nombre d'éléments de l'objet
RO1,RO2,RO3,RO4: dimensions de l'objet

FIG.8: DETAIL D'UN ARTICLE CORRESPONDANT A UN CONSTITUANT

4.7 DESCRIPTION SOMMAIRE DE LA MACHINE D'EXECUTION:

Primitivement, l'interprète *APL* ne reconnaissaitque trois types internes d'objets:

- le type logique (stocké sur bits),
- le type caractère (stocké sur octets),
- le type réel (stocké sur 4 octets).

De nombreuses opérations avaient été programmées en tenant compte d'une part des types particuliers des opérandes et d'autre part des facilités offertes par le code d'ordre de la machine, afin d'optimiser les temps de réponse de l'interprète.

La représentation d'un nombre réel sur quatre octets permet d'obtenir une précision relative de sept chiffres; si dans la plupart des cas, cette précision est suffisante, elle se révèle trop faible dans une application de type comptabilité ou gestion. Nous avons donc été amenés à introduire un type réel double-précision (stocké sur 8 octets), qui porte la précision d'un calcul à environ 15 chiffres.

Dans de nombreux cas, les utilisateurs manipulent des nombres entiers positifs ou négatifs de valeur absolue faible (inférieure à $2^{15}-1$), ce sont des indices dans des tableaux par exemple. Il a donc semblé intéressant d'introduire aussi le type entier (stocké sur 2 octets), ce qui permet un gain de mémoire de stockage.

L'introduction dans l'interprète d'un deuxième processeur de stockage (en mémoire virtuelle) et de deux nouveaux types de représentation interne des objets (types entier et double-précision), ont nécessité une réorganisation profonde de l'interprète, afin d'adopter pour la machine d'exécution une procédure standard.

On peut recenser dix types d'accès aux éléments:

- à partir de la mémoire centrale (bloc de travail) ou de la mémoire virtuelle,
- à partir de bits, octets, mots, doubles-mots ou quadruples-mots.

Il existe dans une opération arithmétique, douze types de conversion de la représentation interne des éléments numériques (fig.*11*).

```
Procédure EXECUTION ( A,B,f)
     A,B: adresses des descripteurs compactés des opérandes;
     f  : opération à effectuer;
     ACTIVATION DE L'OPERANDE A;
     ACTIVATION DE L'OPERANDE B;
     TESTS DE COMPATIBILITE DES OPERANDES
     ET DE L'OPERATION;                              (*)
     CREATION D'UN TEMPORAIRE RESULTAT;
     INITIALISATION DE L'OPERATION;                  (*)
     tant que L'OPERATION N'EST PAS TERMINEE faire;
          CHARGER L'ELEMENT D'INDICE i DE A;
          CHARGER L'ELEMENT D'INDICE j DE B,
          EXECUTER L'OPERATION ELEMENTAIRE;          (*)
          DECHARGER L'ELEMENT D'INDICE k DU RESULTAT;
     fintant;
     DESACTIVATION DE L'OPERANDE A;
     DESACTIVATION DE L'OPERANDE B;
     DESACTIVATION DU RESULTAT;
finproc;
```

(*) Ces lignes dépendent de l'opération effectuée.

FIG.10 ALGORITHME DE LA MACHINE D'EXECUTION

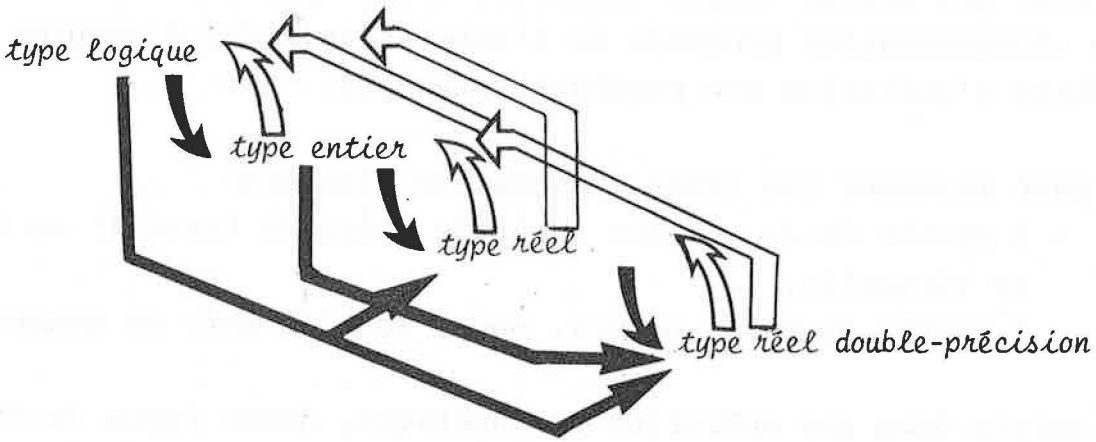


FIG.11 12 PROCEDURES DE CONVERSION

L'algorithme de la machine d'exécution peut se détailler suivant le schéma 10, dans le cas d'une opération dyadique (à deux opérandes). Parmi les différentes étapes, nous nous proposons d'expliciter celles indépendentes de l'opération effectuée.

4.7.1 PRELIMINAIRE:

Les objets *APL* dans le bloc de travail (variables classiques, liens, fonctions utilisateur) sont stockés sous forme de descripteurs dits compactés, afin de minimiser l'encombrement de la mémoire. Lors de l'activation des opérandes d'une fonction, les descripteurs compactés de ces opérandes sont mis sous une forme plus directement exploitable par la machine d'exécution, on dit alors qu'ils sont mis sous une forme éclatée (fig.12).

4.7.2 ACTIVATION D'UN OPERANDE:

On éclate le descripteur compacté de l'objet dans un descripteur "éclaté." Il existe deux types de procédure d'éclatement suivant que l'objet est classique ou virtuel (fig.12).
Si l'objet est virtuel, une demande de contrôle exclusif à cet objet est faite.
L'adresse de la procédure de chargement des éléments de l'objet est déterminée à ce moment.

4.7.3 CREATION D'UN OBJET TEMPORAIRE:

Après la phase de tests de compatibilité des opérandes avec l'opération, on peut déterminer la taille du résultat.
Si l'un des opérandes est virtuel, alors le résultat est stocké en mémoire virtuelle, dans le cas contraire, il est stocké dans le bloc de travail.
La phase de création du résultat comme temporaire consiste à obtenir une zone en vue de stocker les valeurs du résultat, et à remplir à partir des caractéristiques des opérandes et de l'opération, le descripteur éclaté du résultat.

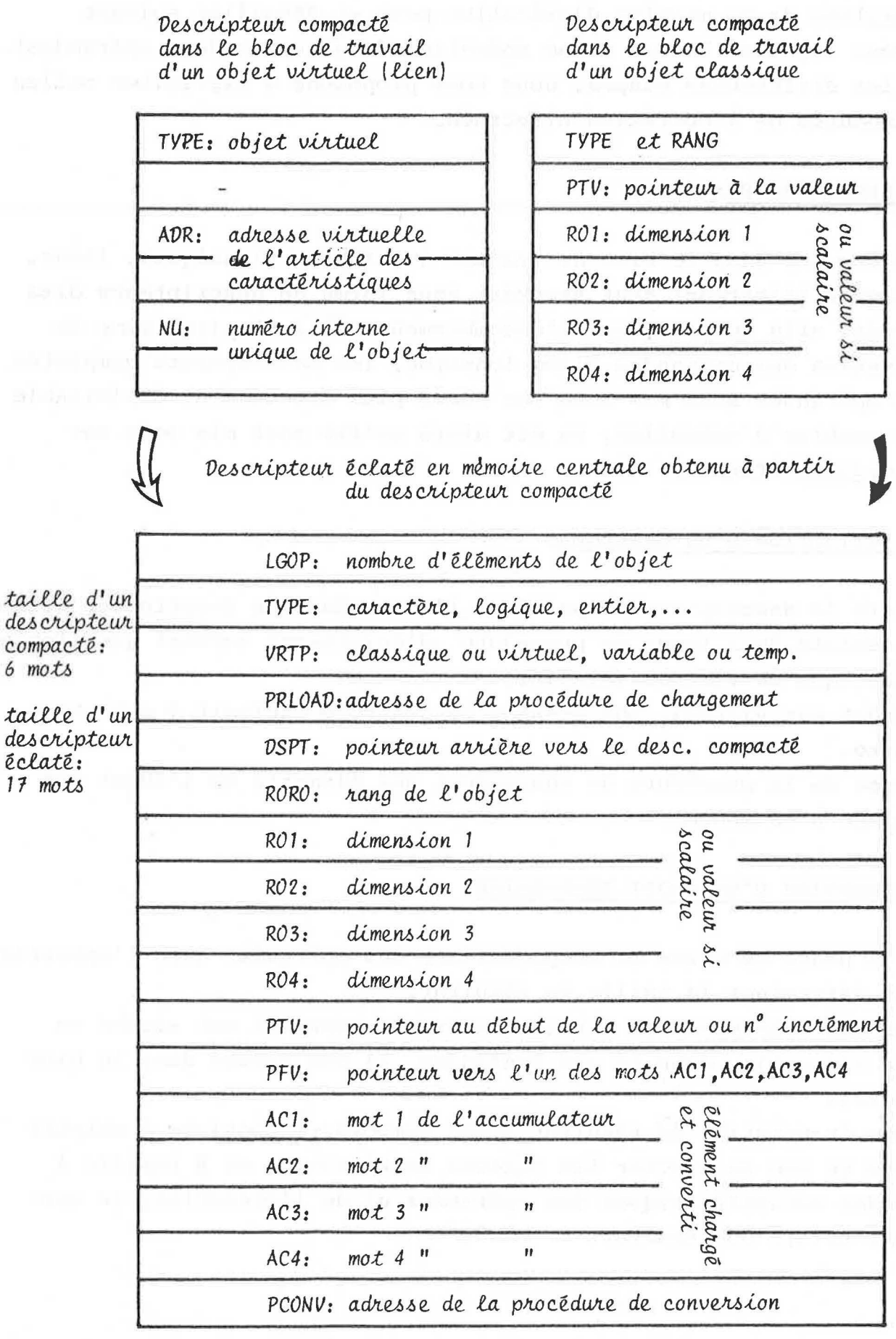


FIG. 12 ACTIVATION D'UN OPERANDE: ECLATEMENT D'UN DESCRIPTEUR

4.7.4 CHARGEMENT D'UN ELEMENT:

Cette phase consiste à appeler la procédure de chargement (déterminée lors de l'activation de l'opérande) avec les paramètres:

- adresse du descripteur éclaté concerné,
- indice de l'élément à charger.

La procédure appelée va chercher l'élément désiré en mémoire centrale ou virtuelle, convertir éventuellement cet élément puis le déposer dans l'accumulateur de l'opérande (quatre derniers mots du descripteur éclaté).
L'adresse de la procédure de conversion éventuelle est remplie lors de la phase initialisation de l'opération.

4.7.5 DESACTIVATION D'UN OPERANDE:

Lorsque l'opération est terminée, il faut déverrouiller l'accès à l'opérande s'il est virtuel, puis le détruire s'il est temporaire.

Cette dernière opération nécessite de détruire:

- la valeur de l'opérande (en mémoire centrale ou virtuelle),
- le descripteur compacté de l'opérande; l'adresse de celui-ci est indiquée dans le descripteur éclaté,
- l'article définissant l'opérande si celui-ci est virtuel.

4.7.6 DESACTIVATION DU RESULTAT:

Le résultat est un objet temporaire auquel ne correspond aucun descripteur compacté. Cette phase nécessite donc:

- d'obtenir une zone en mémoire centrale en vue de compacter le descripteur,
- d'obtenir en outre, si le résultat est virtuel, un article libre dans l'espace structures de l'utilisateur, en vue de compacter le descripteur.

L'opération d'affectation suit un algorithme différent.

5

CONCLUSION

5.1 NOTRE CHOIX D'ASSOCIATION APL ET MODELE RELATIONNEL:

Le langage *APL* nous a séduit par les larges facilités qu'il offre dans la manipulation des tableaux, certaines fonctions primitives sont même directement orientées vers l'interrogation d'une base de données.

Il est bien sûr concevable de disposer de possibilités équivalentes dans d'autres langages plus classiques (FORTRAN,COBOL...), si l'on dispose alors de bibliothèques de sous-programmes très complètes. Mais dans le cas d'*APL*, la syntaxe d'appel de ces 'sous-programmes' est totalement transparente, point n'est besoin de déclarer les types et dimensions des arguments passés. D'autre part, dans leur conception, les systèmes *APL* sont très orientés vers le dialogue 'homme-machine' qualité indispensable que doit posséder un langage de manipulation de base de données en temps réel.

Ces deux avantages permettent d'écrire et de mettre au point rapidement des programmes concis et d'autant plus lisibles que le programmeur aura su dégager au niveau de chaque ligne d'instructions, chaque action sémantique de son application.

Il est parfois reproché à *APL* la lenteur d'exécution des programmes écrits dans ce langage, lenteur dûe au fait que le langage est interprété et non compilé. Cette lenteur n'est que très relative, dans la mesure où de nombreuses boucles écrites explicitement dans les langages compilés, sont évitées en *APL*, simplifiant ainsi notablement la phase interprétative.

Le Modèle Relationnel nous a séduit par la simplicité des structures d'information qu'il propose: des tableaux liés par des relations.

De là, il ne restait qu'à franchir le pas entre d'un côté un bon langage de manipulation de tableaux, et de l'autre, un modèle de représentation du monde réel proche du langage dans la structuration des informations.

Notre travail a consisté à essayer de résoudre certaines difficultés

rencontrées lors de cette association:

- au niveau du langage *APL*:

  Il fallait adopter une structure relationnelle de données compatible avec *APL* et n'impliquant aucune extension sémantique du langage.

- au niveau des systèmes *APL*:

  Il fallait permettre le partage simultané des informations et offrir au créateur de la Base la possibilité d'exercer un contrôle des accès à la Base.

- au niveau de notre système particulier :

  Il fallait d'une part augmenter de façon notable la capacité de stockage des informations disponible en ligne, et d'autre part, augmenter la précision des données de type numérique.

5.1.1 REPRESENTATION D'UNE STRUCTURE RELATIONNELLE:

La solution choisie a été de fournir à l'utilisateur, au moyen de fonctions système, la possibilité de définir des structures hiérarchisées à deux niveaux:

- une racine (nom de la relation),
- des feuilles (constituants) contenant l'information et ayant une structure de tableaux *APL*.

Le langage de base de manipulation est alors *APL*; les opérations sont déclenchables sur les constituants pris séparément à l'aide des fonctions primitives d'*APL*. En outre, un mini-langage d'interrogation permet d'interroger la base à partir des noms de relation et de leurs constituants.

5.1.2 PARTAGE SIMULTANE D'INFORMATIONS:

La solution adoptée a été de rejeter dans une zone commune à l'ensemble des utilisateurs(dans l'espace virtuel et les cadres de manœuvre), les descripteurs et valeurs des informations mises en commun.La désignation d'un objet, à partir du bloc de travail, est faite au moyen d'un lien vers l'objet commun, stocké dans le bloc de travail de l'utilisateur.

Un moyen de contrôler l'accès aux informations mises en commun est fourni au créateur de la Base, au niveau des relations. Ce contrôle peut être affiné, si l'information n'est accessible que par l'intermédiaire de fonctions *APL* d'accès à la Base définies par le concepteur.

Une synchronisation est effectuée automatiquement au niveau de chaque opération *APL*. En outre, il serait souhaitable que l'utilisateur dispose d'une primitive de demande d'accès exclusif à un ensemble de constituants, afin de lui permettre d'effectuer des mises-à-jour sein d'une relation sans risquer de nuire à l'intégrité des informations. Une telle possibilité a été envisagée et prévue dans notre système; sa réalisation a été différée jusqu'à parution d'une nouvelle version du superviseur du système qui lui aussi doit offrir de telles possibilités.

5.1.3 EXTENSIONS DUES A NOTRE SYSTEME PARTICULIER :

La juxtaposition à notre système d'un processeur de stockage de données en mémoire virtuelle a permis d'augmenter de façon importante la taille des informations directement manipulables à l'aide des fonctions *APL*.

La précision des calculs a été portée à seize chiffres permettant ainsi d'utiliser ce système en vue de programmer des applications de type comptabilité ou gestion; parallèlement, un type interne entier a été introduit afin de minimiser la taille de la plupart des tableaux de nombres entiers.

5.2 CONCLUSION:

Notre réalisation constitue un noyau de système nécessaire et suffisant à l'ouverture d'*APL* vers les Bases de Données.

D'autre part, cette extension est à ce jour opérationnelle et doit être très prochainement distribuée par le constructeur de notre matériel.

REFERENCES BIBLIOGRAPHIQUES

BASES DE DONNEES

[SGBD1] ADIBA M., DELOBEL C.
Les modèles relationnels de bases de données
Rocquencourt, IRIA.Sefi, Avril 1976

[SGBD2] *Banques de Données - Aix en Provence,*
4-6 Juin 1973 - 2 tomes - Colloques IRIA
Rocquencourt, IRIA, 1973

[SGBD3] DATE (C.J.)
An introduction to data base systems
Readings, Mass. Addison - Wesley, 1975

[SGBD4] C.I.I.
Socrate sous SIRIS7/SIRIS8 - Manuel d'utilisation - Brochure C.I.I.
N° 4338 E1/FR.
Louveciennes,C.I.I., 1974

[SGBD5] CODASYL Sustems Committe
Introduction to feature analysis of generalized data base management systems
Communications of the A.C.M., XIV, 5, pp.308-318 - May 1971

[SGBD6] CODD E.F.
A relational model of data for large shared data banks
Communications of the A.C.M., XIII, 6, pp. 377-387
June 1970

[SGBD7] DELOBEL (claude)
Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion
Thèse Etat : Sciences : Grenoble, Univ. Scientifique et Médicale : 1973

[SGBD8] ASH W.L. et SIBLEY E.H.
TRAMP, an interpretive associative processor with deductive capabilities
Proceedings of 23rd National Conference, A.C.M., (pp. 143-156)Princeton, N.Y., Brandon/Systems Press Inc., 1968

[SGBD9] ABRIAL J.R.
Data Semantics
IFIP. TC2 working conference, Carghèse Corse, April 1974

[SGBD10] PIROTTE A.
Comparaison de langages d'interrogation de base de données relationnelle
(Communication colloque "Les bases de données relationnelles, Institut de Programmation, 21 - 23 mars 1978 - Paris).

[SGBD11] CODD E.F.
A data base sublanguage founded on the relational calculul.
Proceedings of the 1971 A.C.M. - SIGFIDET Workshop on Data Description, Access and Control, San Diego, Ca., Nov. 1971

[SGBD12] BOYCE R. and al.
Specifying queries as relational expressions square.
Proceedings of the ACM. SIGPLAN-SIGIR Interface Meeting, Gaithersberg, Md., Nov. 1973.

[SGBD13] STONEBRAKER M.
Implementation of integrity contraints and views query modification
Proceedings of the 1971 ACM SIGMOD .Workshop on management of data, San José, May 1975.

A.P.L.

[APL1] IVERSON K.E.
A programming language
New-york, Wiley & sons, 1962

[APL2] Ecole Nationale Supérieure des Mines de Saint-Etienne - Département Informatique
Une introduction au langage APL-
Paris, Dunod, 1978 (à paraître)

[APL3] NAKACHE Michel
APL/16 - Manuel de référence - 2ème cd - Saint-Etienne, Ecole des Mines - Mars 1977

[APL4] GIRARDOT J.J. et MIREAUX F.
Réalisation d'un interprète complet du langage APL sur un mini-ordinateur
Thèse Doct. Ing : Nancy, Univ. Nancy I : 1976

[APL5] SLIGOS
APL PLUS - Manuel de référence - Paris

[APL6] MARTIN H.
Etude et réalisation d'un système de gestion de fichiers APL
Thèse Doct. Ing. : Toulouse, Univ. Paul Sabatier : 1975

[APL7] LEBORGNE Y.
APL-SV : an extension to APL/360
IBM. FRANCE

[APL8] IBM
V.S. APL for CMS:Writing Auxiliary Processors
Ref : SH20 - 9068-0

[APL9] NEVIANS J.
APL/mitra 15 - Concept de mémoire virtuelle -
Optimisation, notions de structure -
Thèse 3ème cycle : Orsay, Univ. Paris Sud : 1975

[APL10] GHANDOUR Z. et MEZEI J.
General arrays, operations and functions
IBM Journal of Research and Development, pp. 335-352, July 1973

[APL11] ROBICHAUD (Louis P.A.
Structures arborescentes - Manuel de référence APL. SV. Laval
Université de Laval, Québec, 1977

[APL12] ALFONSECA M. et TAVERA M.L.
Extension of APL to tree- structured information
G. Truman HUNTER ed, APL 76, Ottawa

[APL13] GULL W.E. et JENKINS M.A.
Recursive data structures in APL
(Draft soumis à Communications of the ACM, 1977, 47 p.)

[APL14] VASSILIOU Y., TSICHRITZIS D.
A front end data base system
University of Toronto, Dpt of Computer Science.

SYSTEME

[SYS1] GUIBOUD-RIBAUD S.
Mecanismes d'adressage et de protection dans les systèmes informatiques Application au noyau GEMAU.
Thèse d'Etat : Sc : Grenoble, Univ Scientifique et Médicale : 1975.

[SYS2] ABRIAL J.R. et al.
Projet SOCRATE (1) Spécifications générales - Grenoble, Institut de Mathématiques Appliquées, 1970.

6

ANNEXES

EXTENSION APL 16 : ESPACES VIRTUELS

JANVIER 1978 Michel NAKACHE

CREATION ET MANIPULATION DES VARIABLES VIRTUELLES

CREATION ET UTILISATION DES RELATIONS

MANUEL DE L'UTILISATEUR

Cette étude a été réalisée avec la participation financière du Centre Technique Informatique (CTI), sous la convention 76.182.

CREATION ET MANIPULATION DES VARIABLES VIRTUELLES

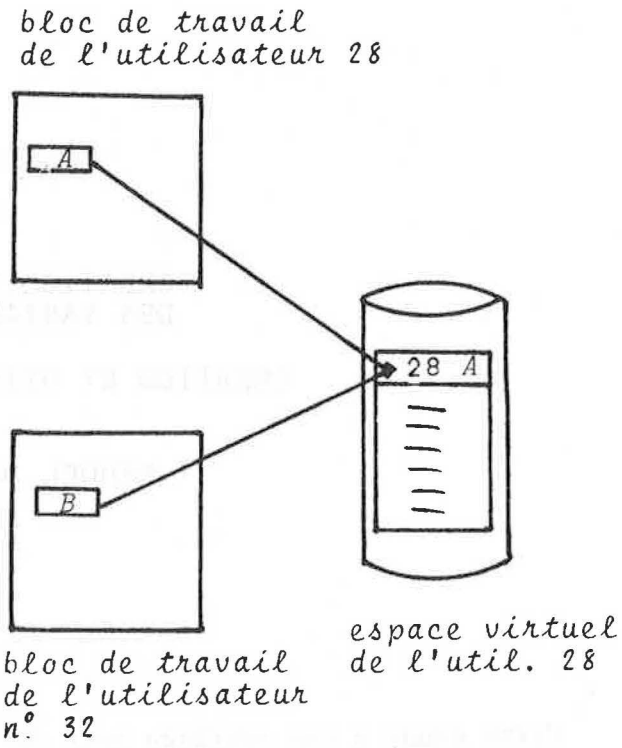
Sous le système APL 16, un utilisateur peut, pour stocker des informations, utiliser soit des variables classiques, soit des *variables virtuelles*. Il a à sa disposition, un ensemble de fonctions système permettant de gérer les variables virtuelles.

Du point de vue de l'utilisateur, il n'existe pas de différence notable entre variables classiques et variables virtuelles.
L'intérêt pratique des variables virtuelles est:

- d'une part, de repousser de façon très sensible les cas de saturation du bloc de travail, car les valeurs de ces variables ne sont pas stockées dans le bloc de travail actif,
- d'autre part, de permettre le partage d'informations entre plusieurs utilisateurs actifs.

I PRINCIPE DE FONCTIONNEMENT

Une variable virtuelle est stockée hors du bloc de travail actif, sous un nom choisi par son propriétaire.
Une fonction système permet d'établir dans le bloc actif un lien vers cette variable; ce lien peut s'appeler du même nom que la variable virtuelle ou d'un autre nom.
Par la suite, l'utilisateur pourra manipuler la variable virtuelle sous le nom de son lien



Un lien ayant dans le bloc actif une taille quasi nulle par rapport à celle de la variable virtuelle (jusqu'à 32767 éléments), les cas de *WS FULL* sont donc repoussés dans de larges limites.

II ACTION DES FONCTIONS PRIMITIVES ET DES OPERATEURS

Il n'existe aucune différence au niveau des opérations, que les variables soient classiques ou virtuelles.
Dans l'exécution d'une opération, si l'un des opérandes est virtuel, le résultat de l'opération est virtuel (stocké hors du bloc actif).
Dans le cas d'une affectation, le stockage est fait dans le bloc actif si la variable affectée n'a pas été l'objet d'une déclaration de variable virtuelle.

ex:

```
      □VCRT 'A'      ←——— déclaration A virtuelle
0
      A←1 2 3        ←——— stockage dans l'espace virtuel
```

III CAS DES FONCTIONS UTILISATEUR

Il existe, dans ce cas, une seule restriction par rapport au cas classique: si l'un des opérandes, lors de l'appel d'une fonction, est virtuel, l'argument correspondant dans la fonction ne peut être modifié par une affectation.

ex:

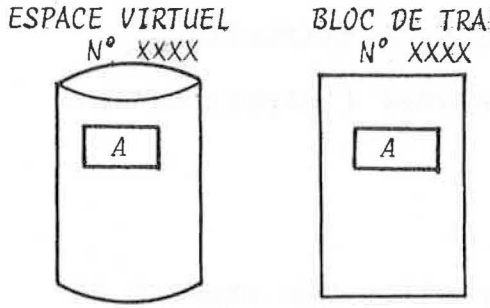
```
      ∇PLUS[□]∇
      ∇ R←PLUS B
[1]   R←B-2          ←——— listing d'une fonction simple
[2]   B←B+4
      ∇
      PLUS A         ←——— appel de la fonction, opérande
VS ACCESS ERROR            virtuel
PLUS[2]  B←B+4
         ∧           ←——— l'argument ne peut être modifié
```

IV ACTION DES COMMANDES SYSTEME

Les commandes système relatives aux variables (*)VARS*, *)ERASE*...) n'ont d'effet que dans le bloc actif, elles permettent d'obtenir la liste ou d'effacer les liens vers les variables virtuelles.
Il existe des fonctions système permettant d'exécuter des actions analogues relativement aux espaces virtuels.

Les commandes système relatives aux blocs de travail (*)LOAD*, *)SAVE*,*)COPY*...) n'ont d'effet que sur les liens aux variables virtuelles; ces liens sont chargés, sauvés, recopiés... au même titre que des variables classiques.
Il existe des fonctions système permettant d'exécuter des actions analogues sur les variables virtuelles.

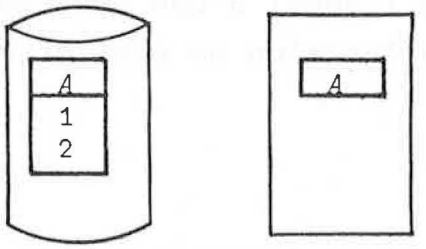
Seul un utilisateur dont le n° de compte est compris entre 1 et 32767 a le droit d'exploiter un espace virtuel (de même numéro).



Exemple d'une session APL:

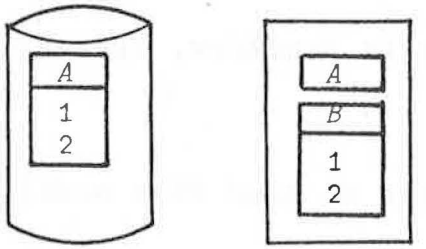
```
      ⎕VCRT 'A'
0
```

création d'une variable virtuelle




```
      A ← 1 2
      A
1  2
```

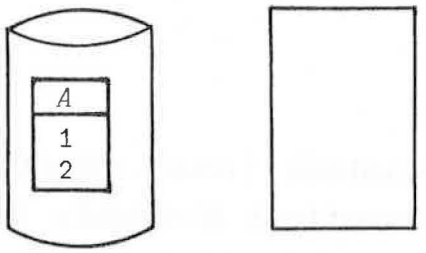
affectation dans l'espace virtuel




```
      B ← A
      )VARS
A     B
```

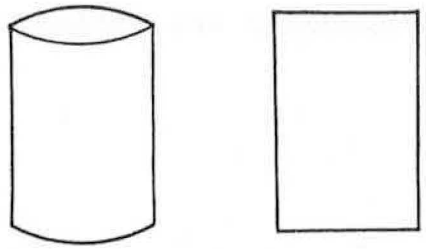
affectation dans le bloc de travail

liste des variables et liens




```
      )ERASE A B
      )VARS
```

destruction des variables et liens



```
      ⎕VERA 'A'
0
```

destruction de la variable virtuelle

Le cylindre et le rectangle représentent respectivement l'espace virtuel et le bloc de travail après l'exécution de l'instruction.

CREATION D'UNE VARIABLE VIRTUELLE: *R* ← □*VCRT* '<idws idvv >'

ACTION:

Création dans l'espace virtuel de l'utilisateur d'une variable virtuelle de nom: 'idvv' (12 caractères maximum). Création dans le bloc actif de l'utilisateur d'un lien de nom: 'idws' (32 caractères maximum) vers cette variable virtuelle.
'idws' ne doit correspondre à aucun objet classique (variable ou fonction).

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUES:

Si 'idws' est omis, le lien dans le bloc actif aura le même nom que la variable virtuelle.

Si l'opérande de la fonction est un tableau de dimensions (n,p), il y a alors création de n variables virtuelles et le résultat est alors un vecteur de dimension n (un compte-rendu par création).

Une erreur pour l'une des variables n'empêche pas la création des autres.

EXEMPLES:

```
      □VCRT 'LIEN  VIRTUEL'
0
      M←3 1⍴'ABC'
      □VCRT M
0  0  0
```

création d'une variable virtuelle de nom VIRTUEL, et d'un lien vers celle-ci de nom LIEN

création de trois variables virtuelles de noms A,B,C et des liens respectifs A,B,C vers ces variables

OBTENTION D'UN LIEN: *R* ← □*VTIE* '<idws N idvv >'

ACTION:

Création dans le bloc actif de l'utilisateur d'un lien de nom: 'idws' (32 caractères maximum) vers la variable virtuelle de nom: 'idvv' de l'utilisateur de n° de compte N. 'idws' ne doit correspondre à aucun objet classique (variable ou fonction).

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUES:

Si N est omis, la recherche a lieu dans ce cas dans l'espace virtuel de l'utilisateur.

Si 'idws' est omis, le lien dans le bloc actif a le même nom que la variable virtuelle.

Si l'opérande de la fonction est un tableau de dimensions (n,p), il y a alors création de n liens et le résultat est alors un vecteur de dimension n (un compte-rendu par obtention).

Une erreur dans l'obtention d'un lien n'empêche pas la création des autres.

EXEMPLES:

```
      □VTIE '1111 LIEN VIRTUEL'
0
```

obtention d'un lien de nom LIEN vers la variable virtuelle de nom VIRTUEL définie dans l'espace virtuel de l'utilisateur de numéro de compte 1111

```
      □VTIE 3 1ρ'ABC'
8  0  5
```

obtentions de liens de noms: A,B,C vers les variables virtuelles de noms:A,B,C définies dans l'espace virtuel de l'utilisateur;
- *la variable virtuelle A n'est pas définie (compte-rendu égal à 8),*
- *l'obtention du lien vers B s'est effectuée correctement (compte-rendu égal à 0)*
- *C correspond déjà à un identificateur défini dans le bloc de travail (compte-rendu égal à 5).*

DESTRUCTION D'UNE VARIABLE VIRTUELLE: *R ← □VERA* '<ident >'

ACTION:

Destruction dans l'espace virtuel de l'utilisateur de la variable virtuelle de nom: 'ident '.
L'utilisation éventuelle de liens vers cette variable provoquera l'erreur: *VS DEFN ERROR.*

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUES:

Si l'opérande de la fonction est un tableau de dimensions (n,p), il y a alors destruction de n variables virtuelles, le résultat est alors un vecteur de dimension n (un compte-rendu par destruction).

Une erreur dans la destruction d'une variable virtuelle n'empêche pas la destruction des autres.

EXEMPLES:

```
      □VERA 'VIRTUEL'
0
```
effacement de la variable virtuelle de nom VIRTUEL

```
      M← 3 2ρ'AABBCC'
      M
AA
BB
CC
      □VERA M
0  0  8
```
création d'une matrice de 3 lignes et 2 colonnes

effacement des variables virtuelles de noms: AA,BB,CC; le compte-rendu indique qu'il n'existe pas de variable virtuelle de nom CC dans l'espace virtuel de l'utilisateur

LISTE DES VARIABLES D'UN ESPACE VIRTUEL: *R ← □VLIB N*

ACTION:

Liste des variables virtuelles d'un espace virtuel.

OPERANDE:

L'opérande est un scalaire numérique entier compris entre 1 et 32767; il indique le n° de l'espace à lister.

RESULTAT:

Le résultat est un tableau de dimensions (n,12)de type caractère; chaque ligne est le nom d'une variable virtuelle.

EXEMPLES:

```
      □VLIB 1111
A
B
C
VIRTUEL

      ρ□VLIB 1111
4  12

      ρ□VLIB 2222
0  12
```

CREATION ET UTILISATION DES RELATIONS

Le système propose à ses utilisateurs désireux de créer une base de données, un ensemble d'outils basés sur les concepts des modèles relationnels de bases de données.

Ces outils permettent de regrouper une importante masse d'informations (jusqu'à 5 M-octets) sous forme d'ensembles cohérents, du point de vue logique.
Le principe offert est de pouvoir partager ou non entre plusieurs utilisateurs des informations structurées en constituants, eux-mêmes regroupés en relations.

Un exemple:
Soit la relation: *EMPLOYE* (*SEXE*, *NOM*, *PRENOM*, *MATRICULE*, *AGE*)
permettant de décrire les employés d'une société, ses constituants sont: *SEXE*, *NOM*, *PRENOM*, *MATRICULE*, *AGE*.

SEXE	*NOM*	*PRENOM*	*MATRICULE*	*AGE*
1	DUPONT	PIERRE	1120	28
1	DURAND	ANTOINE	1189	29
0	GAYANT	FRANCOISE	2009	35
....	....	....	....	....

Une relation est la réunion d'un certain nombre de constituants de telle sorte qu'une ligne fixée pour tous les constituants, permette de décrire une entité de la relation.

ligne 3: Madame Françoise Gayant agée de 35 ans de matricule 2009 ...

Les constituants sont en quelque sorte des tableaux de caractères, de nombres, etc...

Sous le système APL 16, les constituants d'une relation sont assimilés à des variables virtuelles. Ils sont donc manipulables à l'aide des opérateurs et fonctions APL, au même titre que les variables virtuelles (après vérification du droit d'accès défini par le propriétaire de la relation).

Les constituants d'une relation sont partageables entre plusieurs utilisateurs (en consultation, en consultation et mise-à-jour), ou non partageables, au gré du créateur de la base.

A tout moment, le créateur de la base peut ajouter ou supprimer des constituants à une relation, créer ou détruire des relations.

CREATION D'UNE RELATION: *R* ← □*RSET* '<idrel>(<id1>,<id2>,...)'

R ← '<idrel>' □*RSET* '<id1>'
...

ACTION:

Création dans l'espace virtuel de l'utilisateur de la relation de nom: 'idrel' (12 caractères maximum) et de ses constituants de noms:'id1','id2',...(12 caractères maximum).
Une relation peut posséder au maximum 16 constituants.

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUES:

Deux relations peuvent être bâties sur des constituants de même nom; aucune ambigüité n'est à craindre dans la désignation de ceux-ci.

Cette fonction est aussi définie sous forme dyadique, elle effectue les mêmes actions; l'opérande gauche est alors une chaîne de caractères contenant le nom de la relation, l'opérande droit est une matrice de caractères contenant les noms des constituants.

EXEMPLES:

```
      □RSET 'RELATION(CH1,CH2,CH3)'      création dans l'espace de l'utili-
0                                        sateur d'une relation

      CONSTITUANT← 3 4⍴'CH1 CH2 CH3 '    création d'une matrice de caractères
      CONSTITUANT
CH1
CH2
CH3
      'RELATION' □RSET CONSTITUANT       autre façon de créer une relation
0
```

MODIFICATION D'UNE RELATION: *R* ← □*RUPD* '<idrel>(<id3>,<id4>,...)'

R ← '<idrel>' □*RUPD* '<id1>'
...

ACTION:

Cette fonction permet d'ajouter à la relation de nom:'idrel' les constituants de noms: 'id3','id4',...(12 caractères maximum). Une relation peut posséder au maximum 16 constituants.

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUES:

Deux relations peuvent être bâties sur des constituants de même nom; aucune ambigüité n'est à craindre dans la désignation de ceux-ci.

Cette fonction est aussi définie sous forme dyadique, elle effectue les mêmes actions; l'opérande gauche est alors une chaîne de caractères contenant le nom de la relation, l'opérande droit est une matrice de caractères contenant les noms des constituants à ajouter.

EXEMPLES:

```
      □RUPD 'RELATION(CH4,CH5)'      adjonction à la relation RELATION des
0                                    constituants CH4 et CH5

      CONSTITUANT←2 4⍴'CH4 CH5'
      CONSTITUANT
CH4                                  autre façon d'exécuter cette adjonction
CH5
      'RELATION' □RUPD CONSTITUANT
0
```

LIEN VERS UN CONSTITUANT: *R ← 'N <idrel>' ⎕RGET '<idws idl>'*

ACTION:

Création dans le bloc actif de l'utilisateur d'un lien de nom: 'idws' (32 caractères maximum) vers le constituant de nom: 'idl' de la relation de nom: 'idrel' de l'espace virtuel de l'utilisateur de n° de compte N.

'idws' ne doit correspondre à aucun objet classique (variable ou fonction) dans le bloc de travail.

Si l'utilisateur a le droit d'avoir accès au constituant, la demande est exécutée, sinon la demande provoque l'erreur: *VS ACCESS ERROR.*

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUES:

N peut être omis, la recherche a lieu dans ce cas dans l'espace virtuel de l'utilisateur.

Si 'idws' est omis, le lien dans le bloc actif, aura le même nom que le constituant.

Si l'opérande droit de la fonction est un tableau de dimensions (n,p), il y a alors création de n liens et le résultat est alors un vecteur de dimension n (un compte-rendu par obtention).

Une erreur dans l'obtention d'un lien n'empêche pas l'obtention des autres.

EXEMPLES:

```
      '1111 RELATION' ⎕RGET ' A  CH1'
0
```

obtention d'un lien de nom A vers le constituant CH1 de la relation RELATION définie chez l'utilisateur de numéro de compte 1111

```
      MAT←2 5⍴'A CH1B CH2'
      MAT
A CH1
B CH2
      'RELATION' ⎕RGET MAT
0  0
```

création d'une matrice de caractères

obtention de liens A et B vers les constituants CH1 et CH2 de la relation RELATION de l'utilisateur

DESTRUCTION D'UNE RELATION: *R ← ⎕RERA* !<idrel>'

ACTION:

Destruction dans l'espace virtuel de l'utilisateur de la relation de nom :'idrel' et de tous ses constituants.

L'utilisation éventuelle de liens vers ses constituants provoqueront l'erreur: *VS DEFN ERROR*.

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur.

REMARQUE:

Si l'opérande de la fonction est un tableau de dimensions (n,p), il y a alors destruction de n relations, le résultat est alors un vecteur de dimension n (un compte-rendu par destruction).

Une erreur dans la destruction d'une relation n'empêche pas la destruction des autres.

EXEMPLES:

```
      ⎕VERA 'RELATION'
0

      ⎕VERA 'TOTO'
8
```

le compte-rendu signifie ici que la relation TOTO n'est pas définie dans l'espace virtuel de l'utilisateur

SUPPRESSION DE CONSTITUANTS: *R* ← '<idrel>' □*RERA* '<idl>'

ACTION:

Suppression et destruction dans la liste des constituants de la relation de nom: 'idrel', du constituant de nom 'idl'. L'utilisation éventuelle d'un lien vers le donstituant provoquera l'erreur: *VS DEFN ERROR*.

RESULTAT:

Le résultat est un vecteur numérique entier de 1 élément fournissant le compte-rendu de la demande; s'il est différent de 0, se reporter au tableau des compte-rendus d'erreur .

REMARQUE:

Si l'opérande droit de la fonction est un tableau de dimensions (n,p), il y a alors destruction de n constituants, le résultat est alors un vecteur de dimension n (un compte-rendu par destruction).

Une erreur dans la destruction d'un constituant n'empêche pas la destruction des autres.

EXEMPLES:

```
      'RELATION' □RERA 'CH1'
0
      MAT←2 3⍴'CH1CH2'
      MAT
CH1
CH2
      'RELATION' □RERA MAT
8  0
```

effacement du constituant CH1 de la relation RELATION chz l'utilisateur

création d'une matrice de caractères

effacement des constituants CH1 et CH2; le constituant CH1 de la relation n'est pas défini, CH2 a été effacé

LISTE DES RELATIONS D'UN ESPACE VIRTUEL: *R ← □RLIB N*

ACTION:

Liste des relations d'un espace virtuel.

OPERANDE:

L'opérande est un scalaire numérique entier compris entre 1 et 32767; il indique le n° de l'espace virtuel à lister.

RESULTAT:

Le résultat est un tableau de dimensions (n,12) de type caractère; chaque ligne est le nom d'une relation.

LISTE DES CONSTITUANTS D'UNE RELATION: *R* ← □*RLIB* *'N* <idrel>'

ACTION:

Liste des constituants d'une relation.

OPERANDE:

L'opérande est un vecteur de caractères contenant le n° de l'espace virtuel dans lequel est définie la relation, suivi du nom de la relation dont on désire lister les constituants. Si le n° de l'espace est omis, la recherche a lieu dans l'espace virtuel de l'utilisateur.

RESULTAT:

Le résultat est un tableau de dimensions (n,12) de type caractère; chaque ligne est le nom d'un constituant la relation

Si la relation n'est pas trouvée dans l'espace considéré, le résultat est un vecteur nul.

Si la relation est vide, le résultat est un tableau de dimensions (0,12).

EXEMPLES:

```
      □RLIB '1111 RELATION'          liste des constituants de la relation
CH1                                  RELATION définie dans l'espace de l'uti-
CH2                                  lisateur 1111
CH3
CH4
CH5

      ρ□RLIB 'RELATION'              la relation RELATION n'est pas définie
0                                    dans l'espace de l'utilisateur

      ρ□RLIB '1111 RELATION'         cette relation possède cinq constituants
5 12
```

LISTE DES ACCES EN LECTURE SEULE A UNE RELATION: *R* ← □*RARO* ' N <idrel>'

ACTION:

Liste des n° de comptes ayant le droit d'établir un lien en lecture seulement vers les constituants de la relation de nom: 'idrel' définie dans l'espace virtuel de l'utilisateur N.

OPERANDE:

L'opérande est un vecteur de caractères contenant le n° de l'espace virtuel dans lequel est définie la relation, suivi du nom de la relation. Si le n° de compte est absent ,la recherche a lieu dans l'espace virtuel de l'utilisateur.

RESULTAT:

C'est un vecteur numérique entier.

EXEMPLES:

```
      □RARO '1111 RELATION'
1112  1122  3443
```

ACCES EN LECTURE SEULE A UNE RELATION: *R* ← '<idrel>' □*RARO V*

ACTION:

Fixe pour la relation de nom: 'idrel' la liste des utilisateurs ayant le droit d'établir des liens vers ses constituants en lecture seulement.

OPERANDE GAUCHE:

C'est un vecteur de caractères contenant le nom de la relation.

OPERANDE DROIT:

C'est un scalaire ou un vecteur de 16 nombres entiers positifs au plus. Ces nombres représentent la liste des n° de comptes des utilisateurs auxquels on autorise l'accès.

RESULTAT:

C'est un vecteur numérique entier donnant l'ancienne valeur du vecteur des accès.

EXEMPLES:

```
      'RELATION' □RARO 3333 4444
1112  1122  3443
```

modification des numéros de compte ayant accès à la relation RELATION en lecture seulement

```
      □RARO 'RELATION'
3333  4444
```

liste des numéros de comptes ayant...

```
      'RELATION' □RARO ¯0
3333  4444
```

suppression de tout accès en lecture seulement à la relation RELATION

LISTE DES ACCES EN LECTURE-ECRITURE A UNE RELATION: *R* ← □*RARW* ' N <idrel>'

ACTION:

Liste des n° de comptes ayant le droit d'établir un lien en lecture-écriture vers les constituants de la relation de nom: 'idrel', définie dans l'espace virtuel de l'utilisateur N.

OPERANDE:

L'opérande est un vecteur de caractères contenant le n° de l'espace virtuel dans lequel est définie la relation, suivi du nom de la relation. Si le n° de compte est absent , la recherche a lieu dans l'espace virtuel de l'utilisateur.

RESULTAT:

C'est un vecteur numérique entier.

ACCES EN LECTURE-ECRITURE A UNE RELATION: *R* ← '<idrel>' □*RARW V*

ACTION:

Fixe pour la relation de nom: 'idrel' la liste des utilisateurs ayant le droit d'établir des liens vers ses constituants en lecture-écriture.

OPERANDE GAUCHE:

C'est un vecteur de caractères contenant le nom de la relation.

OPERANDE DROIT:

C'est un scalaire ou un vecteur de 16 nombres entiers positifs au plus. Ces nombres représentent la listes des n° de comptes des utilisateurs auxquels on autorise l'accès.

RESULTAT:

C'est un vecteur numérique entier donnant l'ancienne valeur du vecteur des accès.

EXEMPLES:

```
      'RELATION' □RARW 4555 1000
1132  1133  1134
```

modification de la liste des ayant droit d'accès en lecture-écriture à la relation RELATION

```
      'RELATION' □RARW ¯1 0
4555  1000
```

suppression de tous droits d'accès en lecture-écriture à la relation RELATION

TABLEAU DES COMPTE-RENDUS D'ERREURS :

0 : La fonction s'est exécutée correctement.

1 : Erreur de syntaxe dans l'opérande droit.

2 : inutilisé.

3 : inutilisé.

4 : inutilisé.

5 : Le nom du lien correspond déjà à un objet dans le bloc de travail.

6 : La table des symboles du bloc de travail est saturée.

7 : Le nom de la variable ou de la relation est déjà défini dans l'espace virtuel considéré.

8 : Aucune variable ou relation de ce nom dans l'espace virtuel considéré.

9 : inutilisé.

10 : La table des symboles de l'espace virtuel considéré est saturée.

EXTENSION APL 16 : ESPACES VIRTUELS

MARS 1978

Michel NAKACHE

DEFINITION D'UN LANGAGE ALGEBRIQUE D'INTERROGATION DE BASES DE DONNEES RELATIONNELLES ECRIT EN APL

PRESENTATION D'UN LANGAGE ALGEBRIQUE D'INTERROGATION DE BASES DE DONNEES RELATIONNELLES ECRIT EN APL

Ce langage permet d'interroger une base de données définie sous forme de relations (*); ce langage est constitué d'un ensemble de six fonctions *APL* qui permettent d'effectuer les opérations algébriques fondamentales définies sur les relations (**).

Cet ensemble de fonctions peut être complété par l'utilisateur, suivant les cas particuliers qu'il désire traiter, le langage de programmation de base étant *APL*.

I PRESENTATION DES DIFFERENTES FONCTIONS:

I-1 fonction d'affichage: *R ← Y AFF X*

L'argument gauche est un vecteur de caractères; il contient le nom de la (ou des) relation(s) que l'on désire lister. Dans le cas de plusieurs relations, chaque nom est séparé du suivant par un espace. Si l'on désire lister uniquement certains constituants d'une relation, il faut faire suivre le nom de la relation de la liste des constituants désirés, chaque nom étant séparé du suivant par une virgule et la liste étant entourée d'une paire de crochets.

L'argument droit est de type numérique entier; s'il est scalaire, l'ensemble des occurrences est listé , s'il est une matrice, chaque colonne contient les numéros d'occurrences (de lignes) de la relation correspondante à lister. Le nombre de relations de l'opérande gauche est donc égal au nombre de colonnes de l'opérande droit.

(*) voir brochure du même auteur:
EXTENSION APL16: ESPACES VIRTUELS - CREATION ET UTILISATION DES RELATIONS janvier 1978

(**) voir publication IRIA de M.ADIBA et C.DELOBEL
LES MODELES RELATIONNELS DE BASES DE DONNEES avril 1976

Le résultat est un tableau de caractères où chaque constituant est séparé du précédent par une colonne d'espaces.

Exemples:

'REL' AFF 0	*affichage de toutes les occurrences de tous les constituants de la relation REL*
'REL1 REL2' AFF 0	*même action mais pour les relations REL1 et REL2; dans ce cas, les deux relations doivent avoir même nombre d'occurrences*
'REL[CH1,CH2]' AFF X	*affichage des occurrences définies dans le vecteur X des constituants CH1 et CH2 de la relation REL*
'REL1[CH1,CH2] REL2[]' AFF X	*affichage des occurrences définies dans la première colonne de X des constituants CH1 et CH2 de la relation REL1; dans cet exemple, rien n'est à afficher dans la relation REL2.*

I-2 fonction d'interrogation de la base: *R ← W GET V*

C'est une fonction dyadique, elle permet d'obtenir le résultat d'une sélection, d'une projection, d'un produit ou d'une sélection sur une relation temporaire obtenue à partir d'un produit. Le résultat est un tableau de caractères identique à celui obtenu à l'aide de la fonction *AFF*.

L'argument gauche est un vecteur de caractères contenant la liste des constituants dont on désire les occurrences répondant au prédicat défini dans l'argument droit.

L'argument droit est une chaîne de caractères, il défini l'opération désirée.

I-2-1 la sélection:

L'argument droit débute par le nom de la relation sur laquelle porte la sélection, suivi entre crochets de l'expression conditionnelle définissant la sélection.

Cette expression conditionnelle est constituée d'une suite de comparaisons de constituants à des variables ou des constantes séparées par des conjonctions ou des disjonctions. L'évaluation s'effectue de gauche à droite, les comparaisons ayant une priorité supérieure à celle des conjonctions ou disjonctions.

Exemples:

```
'[CH1,CH2]' GET 'REL[CH1<18∨CH2≠''TOTO''∧CH3=X]'
```

Cette expression permet d'obtenir les occurrences des constituants CH1 et CH2 de la relation REL vérifiant l'expression conditionnelle.

L'expression conditionnelle peut être éventuellement parenthésée:

```
'[CH1,CH2]' GET 'REL[(CH1<18)∨(CH2≠''TOTO'')∧(CH3=X)]'
```

I-2-2 la projection:

L'argument droit débute par le nom de la relation sur laquelle porte l'opération, suivi entre crochets des constituants projetés, séparés par des virgules.

Exemple:

```
'[CH1,CH2]' GET 'REL[CH1,CH2]'
```

Obtention des occurrences des constituants CH1 et CH2 de la relation REL après projection de celle-ci sur les constituants CH1 et CH2.

I-2-3 le produit de deux relations:

L'argument droit débute par les noms des deux relations dont on désire faire le produit, séparés par un astérisque L'expression du produit figure ensuite entre crochets. C'est une expression conditionnelle où les opérandes gauches sont des constituants de la première relation et les arguments de droite, des constituants de la deuxième relation.

Exemples:

```
'[CH1,CH2][CH3]' GET 'REL1*REL2[(CH1=CH3)∨(CH2≠CH3)]'

'[CH1,CH2][CH1,CH3]' GET 'REL1*REL2[CH1=CH1]'
```

I-3 fonctions arithmétiques:

maximum	`R ← MAX V`
moyenne	`R ← MOYENNE V`
compter	`R ← COMPTER V`

L'argument des fonctions *MAX* et *MOYENNE* est un tableau de caractères numériques qui peut être considéré comme un vecteur de nombres.

L'argument de la fonction *COMPTER* est un tableau de type quelconque.

Le résultat des fonctions *MAX* et *MOYENNE* est un scalaire numérique représentant respectivement le maximum et la moyenne de l'opérande. Le résultat de la fonction *COMPTER* est un scalaire numérique représentant le nombre de lignes de l'argument.

II EXEMPLE D'UTILISATION DE CE LANGAGE D'INTERROGATION:

Nous nous proposons d'illustrer l'utilisation de ce langage en posant douze questions à une base de données constituée des relations suivantes:

- relation EMP bâtie sur les constituants:
  - NOM: nom d'un employé;
  - SAL: salaire de l'employé,
  - MGR: nom de son responsable direct,
  - DPT: nom du département auquel il appartient;

- relation VENTE bâtie sur les constituants:
  - DPT: nom du département où l'on peut trouver un article,
  - ART: nom de l'article,
  - VOL: volume vendu de cet article;

- relation LOC bâtie sur les constituants:
  - DPT: nom d'un département,
  - ETA: numéro de l'étage où se trouve le département.

Dans la suite de cette présentation, les instructions frappées par l'utilisateur sont écrites dans des cadres grisés.

Les réponses fournies proviennent de l'ordinateur.

EMP *Listing de la relation EMP*

```
'EMP' AFF 0
```

```
◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: SAL
CONSTITUANT: MGR
CONSTITUANT: DPT
GRAAL       9000                    DIRECTION
BOURGE      5000    GRAAL           JOUET
DUPONT      1800    BOURGE          JOUET
DURAND      1900    BOURGE          JOUET
BASTE       2000    BOURGE          JOUET
PILLON      6000    GRAAL           MENAGER
LALIC       1900    PILLON          MENAGER
BOUIG       2200    PILLON          MENAGER
SITO        5500    GRAAL           JARDIN
ARON        2400    SITO            JARDIN
GARAND      3000    SITO            JARDIN
MEYER       1900    GARAND          JARDIN
DUPONT      1900    GARAND          JARDIN
```

VENTE *Listing de la relation VENTE*

```
'VENTE' AFF 0
```

```
◇◇ RELATION: VENTE
CONSTITUANT: DPT
CONSTITUANT: ART
CONSTITUANT: VOL
JOUET       POUPEE      300
JOUET       BICYCLETTE  2000
JOUET       CAMION      250
JOUET       VOITURE     230
MENAGER     ASSIETTE    1000
MENAGER     BOL         500
MENAGER     COUTEAU     800
JARDIN      POUPEE      100
JARDIN      CHAISE      1000
JARDIN      TABLE       1500
JARDIN      PELLE       500
JARDIN      PIOCHE      800
```

LOC *Listing de la relation LOC*

```
'LOC' AFF 0
```

```
◇◇ RELATION: LOC
CONSTITUANT: DPT
CONSTITUANT: ETA
DIRECTION   1
JOUET       2
MENAGER     1
JARDIN      2
```

Q1 *Trouver les noms et salaires des employés du département des jouets:*

```
'[NOM,SAL]' GET 'EMP[DPT=''JOUET'']'
```

```
◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: SAL
BOURGE      5000
DUPONT      1800
DURAND      1900
BASTE       2000
```

Cette question fait intervenir une sélection

Q2 *Trouver les noms des employés et de leur responsable direct qui travaillent au département des jouets et gagnent 2000 F ou moins:*

```
'[NOM,MGR]' GET 'EMP[DPT=''JOUET''∧SAL≤2000]'
```

```
◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: MGR
DUPONT      BOURGE
DURAND      BOURGE
BASTE       BOURGE
```

Cette question fait intervenir une sélection

Q3 *Trouver le volume des ventes des poupées dans le département des jouets:*

```
'[VOL]' GET 'VENTE[(DPT=''JOUET'')∧(ART=''POUPEE'')]'
```

```
◇◇ RELATION: VENTE
CONSTITUANT: VOL
300
```

Cette question fait intervenir une sélection

Q4 *Trouver le nom et le salaire du responsable de l'employé DUPONT:*

```
'[NOM,SAL]' GET 'EMP[NOM=X]' α X←'[MGR]' GET 'EMP[NOM=''DUPONT'']'
```

```
◇◇ RELATION: EMP
CONSTITUANT: MGR

◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: SAL
BOURGE      5000
GARAND      3000
```

Remarquer que la première expression exécutée est celle située à droite du symbole α qui joue ici le rôle de séparateur d'expressions

Deux résultats sont fournis car il existe deux employés nommés DUPONT

Q5 *Trouver la liste des responsables de l'entreprise:*

```
'[MGR]' GET 'EMP[MGR]'
```

```
◇◇ RELATION: EMP
CONSTITUANT: MGR
```

Cette question fait intervenir une projection

```
GRAAL
BOURGE
PILLON
SITO
GARAND
```

Q6 *Trouver les noms et salaires des responsables de l'entreprise:*

```
'[NOM,SAL]' GET 'EMP[NOM=X]' α X←'[MGR]' GET 'EMP[MGR]'
```

```
◇◇ RELATION: EMP
CONSTITUANT: MGR

◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: SAL
GRAAL       9000
BOURGE      5000
PILLON      6000
SITO        5500
GARAND      3000
```

Cette question fait d'abord intervenir une projection: obtention dans la variable X de la liste des noms des responsables, puis une sélection: sélection des employés dont les noms figurent dans la variable X

Q7 *Trouver pour chaque employé, l'étage où il travaille:*

```
'[NOM]⌊ETA]' GET 'EMP*LOC[DPT=DPT]'
```

```
◇◇ RELATION: EMP
CONSTITUANT: NOM

◇◇ RELATION: LOC
CONSTITUANT: ETA
GRAAL       1
BOURGE      2
DUPONT      2
DURAND      2
BASTE       2
PILLON      1
LALIC       1
BOUIG       1
SITO        2
ARON        2
GARAND      2
MEYER       2
DUPONT      2
```

Cette question fait intervenir un produit de deux relations

Q8 *Trouver les noms et salaires des responsables de l'entreprise:*

```
'[NOM,SAL][]' GET 'EMP*EMP[NOM=MGR]'
```

```
◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: SAL

◇◇ RELATION: EMP
GRAAL      9000
GRAAL      9000
GRAAL      9000
BOURGE     5000
BOURGE     5000
BOURGE     5000
PILLON     6000
PILLON     6000
SITO       5500
SITO       5500
GARAND     3000
GARAND     3000
```

Cette question est identique à la question Q6; mais ici, bien que plus synthétique dans son écriture, l'algorithme d'obtention du résultat engendre un nombre beaucoup plus important d'opérations

On remarquera que le résultat obtenu est équivalent à celui obtenu en réponse à la question Q6, à quelques duplications près

Q9 *Trouver le nom et le salaire du responsable de l'employé DUPONT:*

```
'[NOM,SAL][]' GET 'EMP*EMP[NOM=MGR][''DUPONT''=NOM]'
```

```
◇◇ RELATION: EMP
CONSTITUANT: NOM
CONSTITUANT: SAL

◇◇ RELATION: EMP
BOURGE     5000
GARAND     3000
```

Cette question est identique à la question Q4; mais là aussi, le nombre d'opérations engenarées par cet algorithme est beaucoup plus important, une relation temporaire a été en particulier définie afin d'effectuer la sélection sur cette dernière

Q10 *Trouver les noms des employés qui gagnent plus que n'importe quel employé du département des jouets*

```
'[NOM]' GET 'EMP[SAL>X]' α X←MAX '[SAL]' GET 'EMP[DPT=''JOUET'']'
```

```
◇◇ RELATION: EMP
CONSTITUANT: SAL

◇◇ RELATION: EMP
CONSTITUANT: NOM
GRAAL
PILLON
SITO
```

La première opération permet de calculer dans la variable X la valeur du maximum des salaires des employés du département des jouets, la deuxième opération (à gauche du symbole α) est une sélection

Q11 *Trouver la moyenne des salaires perçus par les employés du département des jouets*

```
MOYENNE '[SAL]' GET 'EMP[DPT=''JOUET'']'
```

```
◇◇ RELATION: EMP
CONSTITUANT: SAL
2675
```

Q12 Compter le nombre des employés du département des jouets:

```
COMPTER'[NOM]' GET 'EMPLDPT='''JOUET''']'
```

◇◇ RELATION: EMP
CONSTITUANT: NOM
4

III ANNEXE: LISTINGS DES FONCTIONS PRECITEES:

```
     ∇ R←W GET V;G;D;I;J;CH1;CH2;RL1;RL2;MSK;OP;EXP;RO
[1]    RL1←⍕¯1+I←⌊/V⍳'*['⍬V;RL2←''
[2]    RL2←⍕¯1+I←V⍳'['⍬V←I↓V ⍝IF V⌊I]='*'
[3]    V←I↓V;MSK←1;OP←'∧'
[4]    →(UN,DE)[1+0≠⍴RL2]
[5]   UN:→(PROJ,SEL)[1+∨/'<≤=≥>≠'∈V] ⍝ PROJECTION OU SELECTION
[6]   PROJ:⎕31 ⍝IF 0≠RL1 RGET 'CH1 ',CH1←(¯1+I←⌊/V⍳',')↑V
[7]    EXP←'MSK←MSK∧CH1∘.= CH1' ⍝IF 1=⍴⍴CH1 ⍝ CAS VECTEUR
[8]    EXP←'MSK←MSK∧CH1∧.=⍉CH1' ⍝IF 2=⍴⍴CH1 ⍝ CAS MATRICE
[9]    ⍎EXP
[10]   ⍞')ERASE CH1'
[11]   V←I↓V;OP←V[I]
[12]   →PROJ ⍝IF ']'≠OP
[13]   MSK←~∨/MSK∧R∘.>R←⍳1⌷⍴⍴MSK
[14]   →FIN⍺R←⌊,MSK)/R
[15]  SEL:CH1←(CH1≠'(')/CH1←⍕¯1+J←⌊/V⍳'<≤=≥>≠')↑V
[16]   CH2←(CH2≠')')/CH2←J↓⍕¯1+I←⌊/V⍳'∨∧]')↑V
[17]   ⍞'CH1←',CH1 ⍝IF 0≠RL1 RGET 'CH1 ',CH1
[18]   ⍞'CH2←',CH2 ⍝IF 0≠RL1 RGET 'CH2 ',CH2
[19]   RO←(⍴⍴CH1),⍴⍴CH2;R←(V⌊J]='≠')⌽'∨∧'
[20]   EXP←'CH1',V[J],'CH2' ⍝IF∧/0 1∈RO
[21]   EXP←R[1],'/CH1∘.',V[J],'CH2' ⍝IF∧/1 1=RO
[22]   EXP←'((⍴CH2)⌊2])↑CH1)',R[2],'.',V⌊J],'⍉CH2' ⍝IF∧/1 2=RO
[23]   EXP←'CH1',R[2],'.',V⌊J],'(⍴CH1)[2]↑CH2' ⍝IF∧/2 1=RO
[24]   EXP←R[1],'/CH1',R[2],'.',V⌊J],'⍉CH2' ⍝IF∧/2 2=RO
[25]   ⍞'MSK←MSK',OP,EXP
[26]   ⍞')ERASE CH1 CH2'
[27]   V←I↓V;OP←V[I]
[28]   →SEL ⍝IF ']'≠OP
[29]   →FIN⍺R←MSK/⍳1⌷⍴⍴MSK
[30]  DE:CH1←⍕CH1≠'(')/CH1←⍕¯1+J←⌊/V⍳'<≤=≥>≠')↑V
[31]   CH2←⍕CH2≠')')/CH2←J↓⍕¯1+I←⌊/V⍳'∨∧]')↑V
[32]   ⎕31 ⍝IF 0≠RL1 RGET 'CH1 ',CH1
[33]   ⎕31 ⍝IF 0≠RL2 RGET 'CH2 ',CH2
[34]   RO←(⍴⍴CH1),⍴⍴CH2;R←1↑(V⌊J]='≠')⌽'∧∨'
[35]   EXP←'CH1∘.',V[J],'CH2' ⍝IF∧/1 1=RO
[36]   EXP←'CH1',R,'.',V[J],'⍉CH2' ⍝IF∧/2 2=RO
[37]   ⍞'MSK←MSK',OP,EXP
[38]   ⍞')ERASE CH1 CH2'
[39]   V←I↓V;OP←V[I]
[40]   →DE ⍝IF ']'≠OP
[41]   G←⍳,MSK)/,MSK×⍉(⍳⌽⍴MSK)∘.≥⍳1↑⍴MSK
[42]   D←⍳,MSK)/,MSK×(⍴MSK)∘.≥¯1↑⍴MSK
[43]   R←FGET R←G,[1.5]D
[44]  FIN:R←⍕(RL1,W[⍳OP],RL2,⍕OP←W⍳']')↓W)AFF R
     ∇
```

```
     ∇ R←COMPTER V
[1]    R←1↑⍴V
     ∇
```

```
     ∇ R←MAX V
[1]    R←⌈/⍞V
     ∇
```

```
     ∇ R←MOYENNE V
[1]    R←(+/V)÷⍴V←⍞V
     ∇
```

```
     ∇ R←V AFF X;RL1;CH1;RL;CH;EXP;NB;I;J;K
[1]    RL←CH←0 12⍴⊂'';NB←⍳0;J←K←1;R←' '
[2]   ET:RL1←12↑⍕¯1+I←⌊/V⍳'[ ')↑V;CH1←0 12⍴' '
[3]    ' ',[.5]'◇◇ RELATION: ',RL1
[4]    →PART ⎕IF '['=1↑(I-1)↓V
[5]    CH1←RLIB RL1
[6]    →SUIT
[7]   PART:CH1←CH1,[1]12↑⍕¯1+I←⌊/V⍳',]')↑V←I↓V
[8]    →PART ⎕IF ','=V[I]
[9]    →TE ⎕IF∧/' '=,CH1
[10]  SUIT:RL←RL,[1]RL1;CH←CH,[1]CH1;NB←NB,1⍴⍴CH1
[11]   ((⍕1⍴⍴CH1),13)⍴'CONSTITUANT: '),CH1
[12]  TE:→ET ⎕IF∧/0≠⍴V←I↓V
[13]  SEL:⍎'CH1',CH[K;]⎕IF 0≠RL⌊J;]RGET 'CH1 ',CH[K;]
[14]   EXP←'⍕CH1'
[15]   EXP←'⍕(⍕⍴CH1),1)⍴CH1' ⎕IF 1=⍴⍴CH1
[16]   →FIN ⎕IF 0=⍴⍴X
[17]   X←((⍴X),1)⍴X ⎕IF 1=⍴⍴X
[18]   EXP←'⍕CH1[X[;,J]]'
[19]   EXP←'⍕CH1[X[;J];]' ⎕IF 2=⍴⍴CH1
[20]  FIN:⍎'R←R,'' '','',EXP
[21]   ⍎')QERASE CH1'
[22]   →SEL ⎕IF NB[J]≥K←K+1
[23]   →SEL ⎕IF(1⍴⍴RL)≥J←J+1
[24]   R←0 2↓R
     ∇
```

```
     ∇ R←FGET W;TB
[1]    R←W;MSK←1;OP←'∧';V←1↓V
[2]    →0 ⎕IF 0=⍴V
[3]    TB←101 111 112 122 1010 1011 1021 1022 ⍝ TABLE DES RO
[4]   FSEL:CH1←(CH1≠'⍳')/CH1←⍕¯1+J←⌊/V⍳'<≤=≥>≠')↑V;EXP←''
[5]    CH2←(CH2≠')')/CH2←J↓⍕¯1+I←⌊/V⍳'∨∧]')↑V;RO←0 0
[6]    ⍎'CH1←',CH1 ⎕IF~RO[1]←0=RL1 RGET 'CH1 ',CH1
[7]    ⍎'CH2←',CH2 ⎕IF~RO[2]←0=RL2 RGET 'CH2 ',CH2
[8]   ET:→ET+TB⍳RO←10⊥RO,(⍴⍴CH1),⍴⍴CH2⍺R←(V⌊J]='≠')⌽'∨∧'
[9]    →FIN⍺EXP←'CH1',V[J],'CH2[D]'
[10]   →FIN⍺EXP←R[1],'/CH1∘.',V⌊J],'CH2[D]'
[11]   →FIN⍺EXP←'((⍴CH2)⌊2])↑CH1)',R⌊2],'.',V⌊J],'⌽CH2[D;]'
[12]   →FIN⍺EXP←R⌊1],'/CH2',R⌊2],'.',V⌊J],'⌽CH2[D;]'
[13]   →FIN⍺EXP←'CH1[G]',V[J],'CH2'
[14]   →FIN⍺EXP←R[1],'/CH1[G]∘.',V[J],'CH2'
[15]   →FIN⍺EXP←'CH1[G;]',R[2],'.',V[J],'(⍴CH1)⌊2])↑CH2'
[16]   →FIN⍺EXP←R[1],'/CH1[G;]',R[2],'.',V⌊J],'⌽CH2'
[17]  FIN:⍎'MSK←MSK',OP,EXP
[18]   ⍎')QERASE CH1 CH2'
[19]   V←I↓V;OP←V[I]
[20]   →FSEL ⎕IF 0≠⍴V
[21]   I←(⍳,MSK)/,MSK×⌽(⍳⌽⍴MSK)∘.=⍳⍴MSK
[22]   J←(⍳,MSK)/,MSK×(⍴MSK)∘.=⍳¯1↑⍴MSK
[23]   R←G⌊I],[1.5]D[J]
     ∇
```