MASTER EN SCIENCES INFORMATIQUES

Langage naturel de consultation de banque de données

Paulus, Walther

Award date:
1973

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur

Faculté d'Informatique

Année académique 1972-1973

Langage naturel de consultation de banque de données

WALTHER PAULUS

Mémoire présenté en vue de l'obtention du grade de
Licencié et Maître en Informatique

FACULTES UNIVERSITAIRES NOTRE - DAME DE LA PAIX NAMUR

FACULTE D' INFORMATIQUE

Année académique I972 - I973

LANGAGE NATUREL DE CONSULTATION DE BANQUE DE DONNEES

Walther Paulus

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.

Avant - propos.

Il nous faut avant tout remercier l'équipe " Grands Fichiers " dont les travaux et les conseils nous ont été précieux.

Nos remerciements s'adressent aussi au personnel ECA - AUTOMATION, travaillant à Grenoble, qui nous a permis de connaître le système Socrate.

Toute notre reconnaissance va également aux personnes qui ont participé à la réalisation technique de ce travail.

Table des matières.

I - Comparaison entre les deux langages de consultation de Data Base. 5

Socrate : langage rigide.

Langage orienté naturel proposé par Woods.

1. Introduction. 6
2. Structure de la base. 23
3. Rapport entre le langage et la base. 28
4. Possibilités offertes par ces langages. 31
5. Extensions. 34
6. Influence de la taille de la base sur les deux systèmes étudiés. 36
7. Tableau récapitulatif. 46
8. Conclusion. 49

II- Essai de définition d'un langage naturel.

1. Introduction. 52
   - I.1 Présentation du problème. 52
   - I.2 Description de l'application choisie. 53
2. Définition d'un langage naturel. 59
   - 2.1. Présentation d'une grammaire. 59
   - 2.2 La citation. 61
   - 2.3 Exemples. 64
3. Etude d'une structure logique générale. 67
   - 3.1 Structure vue utilisateur. 67
   - 3.2 Relations nommées. 70
   - 3.3 Table vue utilisateur. 71
   - 3.4 Table des liens entre entités. 77
4. Recherche d'un langage rigide intermédiaire. 79
   - 4.1 Caractère de ce langage. 79
   - 4.2 Le langage lui-même. 80
   - 4.3 Essai de formalisation et exemples. 82

5. Description du passage du langage naturel
au langage rigide intermédiaire. 89
5.I Description de différentes tables. 89
5.2 Correspondance LN - LR. 9I
5.3 La table de travail. 92
5.4 Les restrictions. 95
5.5 Aspect conversationnel de ce passage. 96
5.6 Analyseur de requêtes LN. 98
5.7 Exemples. II8

6. Description du passage du langage rigide
à Socrate. I27
6.I Notion de poids. I28
6.2 Description de ce passage. I29
6.3 Requêtes à une entité. I32
6.4 Requêtes à une relation. I36
6.5 Expression SI...ALORS...SINON... I4I
6.6 Les fonctions. I45
6.7 Analyseur de requêtes LR. I49
6.8 Exemples. I6I
6.9 Combinaisons de relations de la
"table vue utilisateur". I8I

7. Conclusion. I9I

Bibliographie.

Thèse de William A. Woods.

Le Projet Socrate.

Eca - automation, IMAG.

Generalized file processing.

William C.Mc Gee.

On the role of naturel langage in man - machine communication.

Article de J. Bruce Fraser.

Procedural semantic for a question - answering machine.

Article de William A. Woods.

A relational Model of Data for Large Shared Data Banks.

E.F. Codd.

Sources de renseignements.

Renseignements obtenus lors du stage à Grenoble.

Travaux et réunions de l'équipe de recherche de l'Institut.

Table des figures.

- Application 53
- Tableau descriptif de la base 55
- Table des caractéristiques 56
- Structure de définition 57
- Structure vue utilisateur et sa forme transposée. 68
- Table vue utilisateur 74
- Table des liens entre entités 78
- Table des quantificateurs 89
- Table des caractéristiques virtuelles 90
- Table des fonctions 90
- Table des synonymes fonctionnels 90
- Table de travail TV, composée de TR 92
  TD
  TF
  TQ
  TL
- Table de travail TTR, composée de référence I36
  astérisque
  entité
  filtre
  domande

Introduction.

A une époque où les banques de données s'avèrent de plus en plus indispensables, il nous a semblé intéressant d'analyser deux systèmes de gestion de ces nouveaux "êtres" informatiques. Il s'agit d'un système complet, Socrate, mis au point à Grenoble par une société de software en association avec l'Institut de Mathématiques Appliquées, et d'un langage orienté "naturel", prposé par W.A.Woods. Ce sont deux systèmes assez différents par leur aspect langage. Woods présente un langage naturel subset de l'Anglais tandis que Socrate est un langage plus rigide ne pouvant être manipulé correctement qu'après une période d'adaptation relativement longue.

Dans une première étape, nous allons comparer ces deux softwares, sur le plan de la consultation de la banque de données. Nous ne nous intéresserons qu'à cet aspect consultation, car c'est le seul point traité par Woods, bien que Socrate permet la définition, la création, la mise à jour d'une base de données. Nous constaterons au cours de cette comparaison que ces deux systèmes peuvent très bien se compléter. En effet, il serait intéressant de permettre à un utilisateur d'interroger la base via un langage qui est le plus proche possible de sa langue habituelle. Notons qu'il ne faut pas croire qu'un langage naturel laisse à tout utilisateur la possibilité d'interroger la base en toute liberté. Il y a bien sûr un certain vocabulaire admis. Le langage est naturel en ce sens qu'il est à peu près équivalent à un langage qu'on obtiendrait en regroupant les questions posées par les utilisateurs dans le cadre d'une application bien déterminée.

C'est dans une deuxième partie que nous traiterons de ce problème : passer d'un langage naturel (LN) à un langage rigide tel que Socrate. Dans le cadre d'un projet de recherche de l'institut d'Informatique, il était intéressant de vérifier s'il serait possible de réaliser ce passage par l'intermédiaire d'un langage rigide (LR) auquel on associera une structure. Ce LR et sa structure associée seraient proches de l'utilisateur, c'est-à-dire plus proche de LN que de Socrate.

I. COMPARAISON - CRITIQUE.

Sujet : Socrate, langage structuré
Langage naturel de Woods.

1. Introduction.

Dans cette première partie, nous allons analyser les deux types de langages que sont Socrate et le langage orienté naturel de Woods.

Le Langage de Woods est restreint à la consultation des data bases. Socrate, quant à lui, est un système complet de gestion de bases de données.

Pour garder une certaine homogénéité à cet exposé, nous allons nous attacher uniquement à un aspect commun aux deux langages : l'interrogation de la banque de données.
Nous parlerons ensuite des autres fonctions qu'on peut leur adjoindre.
Nous allons décrire succinctement chacun des deux langages. Certes, cette description ne sera pas exhaustive, elle a seulement pour but de préciser certaines notions qui seront utilisées dans la suite.

Chacun des points concernant cette première partie sera exposé pour chacun des deux langages étudiés. L'ensemble des conclusions sera repris dans un tableau récapitulatif qui clôturera cette comparaison.

Il nous faut aussi signaler que la présentation de Socrate qui va suivre est certainement déjà dépassée. En effet, ce système de gestion de data base en est encore au stade de la mise au point et est donc en continuelle évolution.

A. Description du langage.

S O C R A T E.

Socrate est un langage non dédicacé à une application particulière. On pourrait classer Socrate parmi les langages évolués au même titre que Fortran, Cobol. Ce langage relativement simple au départ, s'est peu à peu compliqué, et n'est maintenant programmable que par des programmeurs de métier et non par des individus ne connaissant que peu de chose de l'Informatique.

Chaque base de données est définie au moyen d'une structure dite structure de définition, qui est stockée sur un fichier physique et qui va permettre de vérifier les requêtes posées par les utilisateurs. Cette "structure de définition" est l'interface qui existe entre le langage et la base.

Socrate ne consultera la base de données qu'au moment de l'exécution de la requête. Les vérifications syntaxiques se font grâce à une grammaire, tandis que l'analyse sémantique se fait via la "structure interne", forme codifiée de la structure de définition.

La formulation d'une requête doit être hiérarchisée, c'est-à-dire qu'il est nécessaire de citer dans une "citation" toutes les entités auxquelles appartient la caractéristique concernée.

Nous allons donner un exemple de structure de définition, ainsi qu'une requête relative à cette structure. Nous verrons aussi qu'il y a un moyen qui permet d'abréger une telle citation, qui est généralement longue. C'est là le rôle des démonstratifs **XI.**

Exemple d'une structure de définition.

```
DEBUT
  ENTITE session
     DEBUT
       nom                MOT
       message-du-jour    TEXTE
       ENTITE base
          DEBUT
            nom           MOT
            nmax          DE 1 A 100
            presence      ( 0 1 )
            ENTITE utilisateur
               DEBUT
                 nom          MOT
                 mot-passe    MOT
                 ENTITE attachement
                    DEBUT
                      périf    REFERE un périphérique
                    FIN
               FIN
       FIN
     FIN
  ENTITE périphérique
     DEBUT
       nom              MOT
       état             ( libre occupé)
     FIN
  session-encours       MOT
FIN
```

On peut ainsi remarquer les types utilisés en Socrate pour spécifier une caractéristique.

La caractéristique est en fait le niveau le plus bas auquel on peut définir un objet de la base. Ses différents types sont : MOT

TEXTE	
LISTE DE VALEUR	(A B C D)
NUMERIQUE	DE ... A ...
REFERENCE	REFERE

Ces différentes caractéristiques peuvent être régroupées soit en bloc, soit en entité qui n'est autre qu'un bloc à plusieurs occurences. Un bloc n'est qu'un regroupement de caractéristiques : adresse

DEBUT

rue	TEXTE
numéro	DE 1 A 1000
ville	MOT

FIN

Une relation relative à périf aurait la forme :

PERIF DE UN ATTACHEMENT DE UN UTILISATEUR DE UNE BASE DE UNE SESSION

Si en cours de travail, on a déterminé X1 de telle manière qu'il pointe vers un utilisateur de une base de une session, on pourrait formuler la citation précédente comme suit :

PERIF DE UN ATTACHEMENT DE X1

Cette obligation de rendre toute citation complètement hiérarchisée complique la programmation en Socrate. Mais elle permet de lever les ambiguïtés qui peuvent exister par la dénomination de caractéristiques différentes par le même nom.

Le processus d'accès à la base est directement basé sur la structure interne qui contient tous les renseignements nécessaires pour déterminer l'adresse virtuelle de chaque caractéristique. Ces renseignements sont la taille de chaque réalisation d'entité, le déplacement d'une caractéristique par rapport à son entité mère, le type de la caractéristique, sa taille et divers pointeurs.

Grâce à une chaîne de présence d'une réalisation d'entité, chaîne associée à chaque entité, on peut calculer l'adresse virtuelle de début de chaque réalisation. Connaissant le déplacement de chaque caractéristique par rapport à l'entité mère, on peut alors déterminer l'adresse virtuelle de la caractéristique. Vu que le système de projection de l'espace virtuel sur l'espace réel procède par sous-page de 8 mots en général, on déterminera l'adresse virtuelle de la sous-page contenant la caractéristique que l'on veut tester.

De plus, Socrate possède son propre vocabulaire, et une grammaire comprenant ce vocabulaire terminal (mots réservés). Sa programmation est donc stricte du point de vue syntaxique.

Nous pouvons dire que Socrate est fixé

- syntaxiquement par une grammaire
- sémantiquement par la structure de définition

Le schéma 1 représente le processus de compilation et d'interprétation de Socrate.

Cette approche de Socrate nous montre que ce langage est assez compliqué, et que le système ne peut être rentable que si l'opérateur-console connait bien le langage ou s'il ne travaille que par macros.

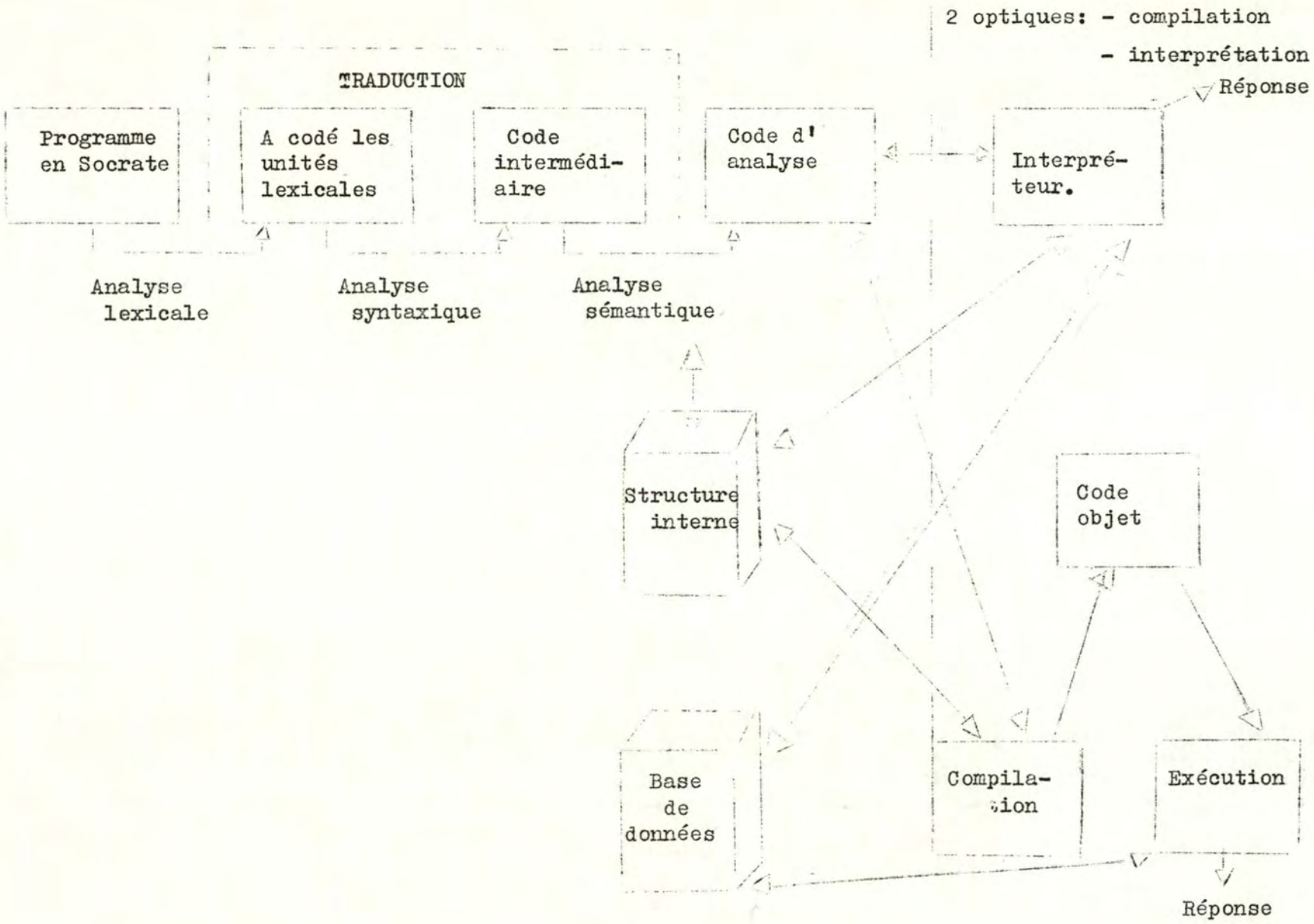

2 optiques: - compilation
- interprétation
TRADUCTION
Programme en Socrate
A codé les unités lexicales
Code intermédiaire
Code d' analyse
Interpréteur.
Réponse
Analyse lexicale
Analyse syntaxique
Analyse sémantique
Structure interne
Code objet
Base de données
Compilation
Exécution
Réponse

Schéma 1.

W O O D S .

Le langage proposé par Woods est un langage orienté naturel, et qui est fortement lié à l'application choisie. C'est évidemment un langage très simple pour l'utilisateur, puisqu'il n'est autre que la langue parlée de l'opérateur.

Woods ne s'intéresse pas à la structure de la base lors de l'analyse des phrases anglaises. Il n'y a pas de structure de définition comme dans Socrate.

Cependant, pour analyser sémantiquement les questions, le système n'accède pas à la base elle-même, mais à un dictionnaire qui donne la classe du mot (déterminatif, noms, préposition, verbe) ainsi que les règles sémantiques qui sont susceptibles d'interpréter les questions contenant ce mot (N, V) .

L'analyse syntaxique produite par Woods consiste simplement à créer un phrase marker correspondant à la phrase anglaise donnée en se basant sur une grammaire context free de la langue anglaise simplifiée.

L'analyse sémantique consiste à rechercher la règle sémantique, qui à la forme Pattern-Action, et qui interprétera correctement la question. Cela est réalisé à l'aide de deux processeurs (S et N). Pour réaliser cette analyse, le système possède une série de règles sémantiques, des arbres partiels et des prédicats. Evidemment, il faut déterminer tous ces éléments, et c'est là un travail assez pénible. Cette recherche de règles et de primitives est très longue, mais peut être réalisée avec de bons résultats par un employé de la société qui désire un tel système. En effet, une telle personne est plus à même qu'un informaticien pour rechercher les relations nécessaires entre les objets de la base pour obtenir un système valable (exemple 1 illustre celà).

On peut dire qu'en général chaque caractéristique (item) de la base sera une primitive. Cela permettra de tester si des éléments donnés appartiennent à la base. D'autres primitives seront obtenues en construisant des propositions qui reflètent les relations qui vont nous intéresser entre les objets de la base.

Enfin, nous complèterons ces primitives par une série de fonctions, qui vont nous permettre de sortir un résultat de la consultation de la base, et non seulement de tester la véracité d'une proposition. Quelques commandes élémentaires termineront ce set de primitives.

Cette détermination des primitives semble être un travail pénible et surtout aléatoire. Cependant, il ne faut pas oublier que ce genre de langage est orienté vers une application bien déterminée, donc, dont on peut connaître assez facilement les exigences.

L'interpréteur qui réalise cette analyse sémantique donne comme résultat un appel de routine, qui n'est autre qu'une primitive ou un ensemble de celles-ci reliées par un opérateur logique, le tout servant de "paramètre" à une des fonctions commandes du système.

C'est le "retriever" qui va, au moyen de cette signification sémantique, rechercher les informations demandées dans la base.

Le schéma 2 retrace cet exposé.

La procédure de l'analyseur sémantique est relativement simple:

- l'analyseur syntaxique nous a fourni un phrase marker représentant la phrase anglaise qui sert à formuler la question.
- l'analyseur sémantique va d'abord rechercher le verbe de cette phrase. A ce verbe est associé un ensemble de règles sémantiques.
- on va alors analyser successivement chacune de ces règles. Cette analyse consiste à rechercher dans le phrase marker des sous-arbres qui ont la structure de ceux décrits dans la règle (templates). Grâce au dictionnaire on peut vérifier si les extrémités de ces sous-arbres ou feuilles, satisfont aux conditions des templates. On arrive ainsi à sélectionner une ou plusieurs règles sémantiques qui interprètent la phrase.

Cet analyseur produit le nom et les paramètres, c'est-à-dire l'appel de la routine, ou un ensemble d'appels de ces routines, qui va servir au retriever pour consulter la base.

Cet ensemble , analyse syntaxique et analyse sémantique, correspond à la phase de traduction dans Socrate.

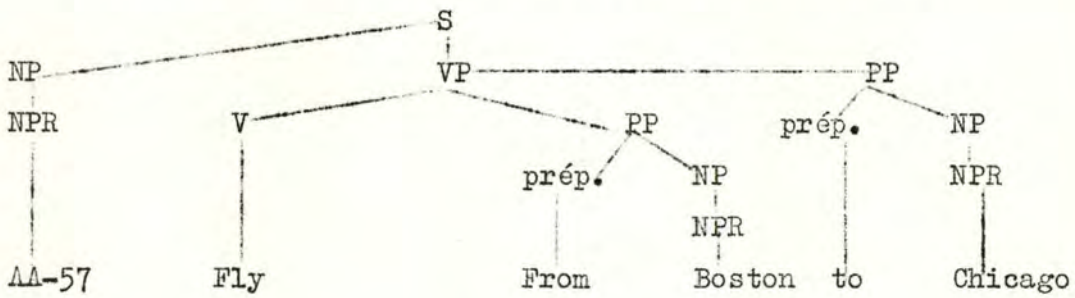
Notons qu'ici aussi on construira des macros qui permettront simplement de grouper un set de phrases anglaises et de lui donner un nom. Cela uniquement parce que l'opération que réalise cet ensemble de phrases est souvent appelée, et non parce que le langage lui-même est peu accèssible à des non-informaticiens comme c'est le cas avec Socrate.

Exemple 1.

L'exemple se rapporte à une Data Base, relative à une compagnie d'aviation, et qui contient divers renseignements au sujet des vols.

Soit la phrase AA - 57 flies from Boston to Chicago.

L'analyseur syntaxique va générer un phrase marker.



N , NPR , V , prép. , sont des classes de mots.

ex : N : flight, plane, jet, city, day

NPR : Boston, JFK Airport, AA-57, breakfast

V : fly, arrive

Dans un dictionnaire, toutes les valeurs des caractéristiques de la base sont reprises, ainsi que tout le vocabulaire admis.

A chaque mot de classe V ou N est associée une série de règles sémantiques :

ex : FLY / V / SS1, SS4, SS5, SS7

On teste chacune de ces règles, dans l'ordre où elles se trouvent dans le dictionnaire, jusqu'au moment où on trouve la règle correcte.

Une règle, soit SS1 , se présente ainsi :

1 - (G1 : FLIGHT ((1)) and ((2)) = FLY or
((2)) = DEPART or
((2)) = GO) and

2 - (G3 : (1) = FROM and PLACE ((2))) and

3 - (G3 : (1) = TO and PLACE ((2)))
= CONNECT (1-1, 2-2, 3-3)

1-1 : - Template 1
- Element 1

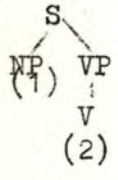
Cette forme de présentation est dite :

(PATTERN - ACTION)

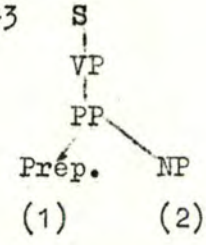
où PATTERN est un ensemble de templates (ici 3)

G1, G3 sont des arbres partiels.

G1



G3



Cette règle interprète la phrase prise pour exemple.

L'interprétation en est :

CONNECT (AA-57 , BOSTON, CHICAGO)

CONNECT est une primitive du langage.

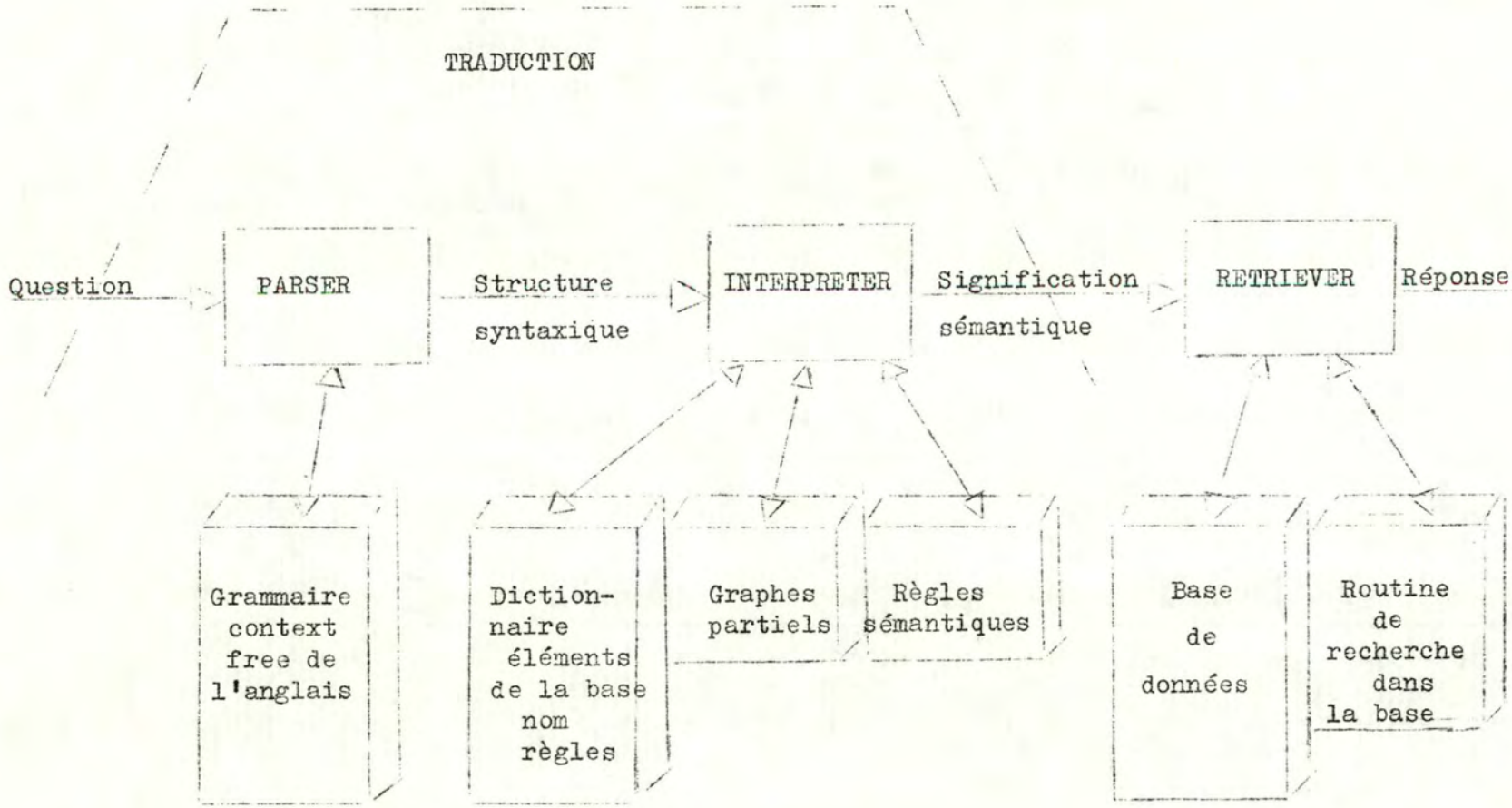

TRADUCTION
Question
PARSER
Structure
syntaxique
INTERPRETER
Signification
sémantique
RETRIEVER
Réponse
Grammaire context free de l'anglais
Dictionnaire éléments de la base nom règles
Graphes partiels
Règles sémantiques
Base de données
Routine de recherche dans la base

Schéma 2.

B. Point de vue utilisateur.

S O C R A T E.

Même si Socrate est assez compliqué de par sa syntaxe, il n'en demeure pas moins qu'il est très intéressant pour un utilisateur. En effet, Socrate n'est pas dédicacé à une application particulière. Un utilisateur ayant employé ce système pour gérer une data base pourra reprendre ce même Socrate pour construire une autre base. Les problèmes qui lui seront posés à ce moment ne concerneront que la structure de définition. Il s'agira de prévoir une organisation rentable pour le travail que l'on envisage. Du côté langage, il n'y aura aucune modification à entreprendre, c'est-à-dire que le vocabulaire reste le même.

W O O D S.

Le langage proposé par Woods ne pose aucun problème à l'utilisateur. En effet, le vocabulaire est toujours une partie du vocabulaire anglais. L'utilisateur aura seulement à connaître le vocabulaire admis par le système, ainsi que les entités essentielles de sa base. En effet, bien que le langage ne soit pas structuré, la connaissance élémentaire des éléments de la base permettra à l'utilisateur de formuler sa question dans une syntaxe qui sera interprétable. Ainsi, il ne risque pas de se voir refuser sa requête parce que l'accès à cette information n'est pas reconnu sémantiquement (par exemple, un synonyme d'un mot accepté n'est pas lui-même reconnu par le système).

C. Point de vue réalisateur.

Définissons ce que nous entendons par ces notions de réalisateur et d'utilisateur. Un utilisateur sera par exemple un opérateur-console, ou toute personne qui manipule le langage sans avoir à en réaliser des programmes complexes. Un réalisateur sera l'équipe qui est à même de programmer dans ce langage, de construire des macros (Socrate) ou de rechercher les primitives et règles sémantiques (Woods).

S O C R A T E.

Quand on envisage la création d'une nouvelle base, il y a des macros à prévoir, et elles ne peuvent être rédigées que par un personnel compétent de façon à les rendre rentables.

Même pour des questions relativement simples, il est intéressant de construire des macros. Cela éviterait à l'utilisateur de connaître les astuces du langage. Seulement s'il faut faire appel à une équipe-réalisation pour ces macros, le système risque de devenir très coûteux.

Tout ce qui touche aux méthodes d'accès est programmé une fois pour toute.

Une coopération avec l'utilisateur pour déterminer la structure de définition n'est pas impossible, et permettrait de rendre celle-ci la plus performante possible.

Il ne faut cependant pas voir un trop grand handicap dans Socrate, à cause de son vocabulaire et de sa syntaxe. Ce langage sera manipulé souvent, et l'opérateur finira par en connaître les éléments fondamentaux. Même si on le classe au même niveau que Fortran, il est cependant plus facile. Sa formulation se fait dans un Français qui n'est pas grammaticalement correcte, mais qui s'adopte assez vite par un utilisateur.

Donnons un exemple de macro, en se basant sur la structure définie au début de cet exposé.

Supposons qu'on ait fréquemment besoin de connaître, au cours d'une session dont le nom est dans session-encours, le nom de l'utilisateur de mot - passe = OK, travaillant sur la base de nom "assurance".

On définit la macro suivante.

```
! DEFMAC          Recherche ! !
! EXP             M X 1 = une session ayant nom=session-encours ;
                  M X 2 = une base ayant nom = " ! 1 ! " de X 1
                  Si existe un utilisateur X3 ayant mot-passe =
                  " ! 2 ! " ; de X2
                  alors I nom de X3
                  sinon I " Cet utilisateur n'existe pas "
! FDEF ?
```

Remarquons que toute requête Socrate se termine par " ? "

Ici, il suffirait de demander :

Recherche assurance OK ?

pour avoir le résultat demandé.

W O O D S.

Ce genre de langage étant dédicacé à une application particulière, tout le travail de détermination des primitives et des règles sémantiques est à refaire si on veut créer une nouvelle base. Cette détermination peut être classée comme équivalente, au point de vue difficulté, à la recherche de la structure de définition chez Socrate.

La "storage structure" doit aussi être redéterminée pour rendre rentable ce nouveau système. Cette storage structure va influencer fortement la programmation des routines de recherche. Cette programmation est également à refaire lors de la création de toute nouvelle base. Dans Socrate, la connaissance de la structure de définition suffit. Ne connaissant pas l'implémentation du système de Woods, la remarque ci-dessus est peut-être osée, mais d'après les notes, il semble que les routines de recherche soient les seuls éléments faisant intervenir l'accès à la base. C'est certainement cet aspect du système de Woods qui est le plus négatif.

Pour une base déjà existante, dès qu'on veut ajouter de nouvelles routines de recherche, on est confronté avec la programmation des routines. Or, une telle ajoute peut être fréquente, ne serait-ce que parce que l'on a besoin de nouvelles relations entre les objets.

Pour des travaux souvent utilisés, demandant un nombre assez élévé de questions, il est intéressant de rédiger des macros. Remarquons que dans ce style de langage, les macros ne sont pas aussi impératives que chez Socrate où le langage est plus compliqué. Elles peuvent cependant améliorer le rendement. Une macro consisterait à attribuer un numéro de référence à une requête complexe, stockée sous forme de primitives.

On éviterait ainsi la retraduction de la requête à chaque appel.

Exemple : si on a la requête :

Does American have a flight which goes from Boston to Chicago

Le résultat de l'analyse sémantique et syntaxique donne :

Test (for some X1 / flight : connect (X1, Boston, Chicago)
and arrive (X1, Chicago), equal (owner (X1), american)).

Si on met American, Boston, Chicago, comme variable, on pourrait avoir :

```
10 Test ( for some X1 / flight :
          connect ( X1, ! 2 !, ! 3 ! )
          and arrive ( X1, ! 3 ! )
          equal ( owner (X1), ! 1 ! ) )
```

et on aurait l'appel ;

```
          10 American Boston Chicago
```

2. Structure de la base.

Nous allons nous intéresser à la "data structure" et non à la "storage structure". Dans la plupart des systèmes de consultation de data bases, cet aspect de "storage structure" est transparent à l'utilisateur. C'est réalisé directement par le système qui optimise ainsi l'emplacement mémoire secondaire.

S O C R A T E.

Grâce à la structure de définition, structure logique des caractéristiques de la base, le système a, à sa portée, tout ce qui lui est nécessaire pour retrouver les informations qui lui sont demandées.

Elle est à définir par l'utilisateur, éventuellement avec l'aide de l'équipe-réalisation, et appartient au même fichier physique que la base. Elle doit être construite en fonction du travail que l'on veut réaliser, de façon à faciliter la formulation des citations et à simplifier les programmes utilisateurs de recherche. Notons qu'ici, recherche signifie situation de la caractéristique dans l'espace virtuel, via la structure de définition. Nous avons vu dans le paragraphe précédent que la structure interne contenait certains renseignements qui permettaient cela.

L'analyse sémantique est également réalisée à partir de cette structure. Il s'agit essentiellement de vérifier si les caractéristiques, entités citées sont prévues dans la structure, et, si les citations sont correctement hiérarchisées.

On n'accèdera à la base que pour rechercher l'information demandée et pour vérifier les filtres imposés aux caractéristiques.

Cette structure peut être réalisée par un employé de la société qui possède ce système, en coordination avec l'équipe réalisation, afin d'optimiser cette organisation logique.

W O O D S.

On pourrait assimiler la structure de définition Socrate à l'ensemble (dictionnaire, graphes partiels, règles sémantiques) et ce, du point de vue analyse sémantique uniquement. Il est évident que cet ensemble est moins riche que la structure de définition dans Socrate.

Cet ensemble est évidemment difficile à construire. Cela suppose qu'on possède une série de questions suffisamment représentative de ce qu'on pourrait demander à la base. On doit aussi rechercher, parmi les phrases markers que donneraient ces phrases par analyse syntaxiques, les parties communes de façon à pouvoir définir le nombre minimum de templates. Ensuite, il nous faut déterminer une série de règles sémantiques qui pourraient répondre aux besoins de la base et qui seraient construites au moyen de templates.

Phrases représentatives.
|
Phrase marker. (par analyse syntaxique.)
|
Sous - arbres. (recherche des élements principaux, communs.)
|
Règles sémantiques. (Construire les templates (conditions sur les feuilles)

Etablir le dictionnaire est aussi une opération pénible. Il s'agit, pour chaque verbe et nom, de rechercher quelles sont les règles sémantiques susceptibles d'interpréter des phrases contenant ces mots.

Il est possible que dans l'analyse que l'on fait pour construire cette "pseudo-structure", on oublie des règles, des phrases de départ.

Dans ce cas, on restreint les possibilités du langage. Cela ne se présenterait pas dans un langage structuré, indépendant de l'application.

Ce procédé est à répéter chaque fois que l'on veut créer une nouvelle base. Il en est de même pour la structure de définition, mais là, le risque d'erreur est moins grand.

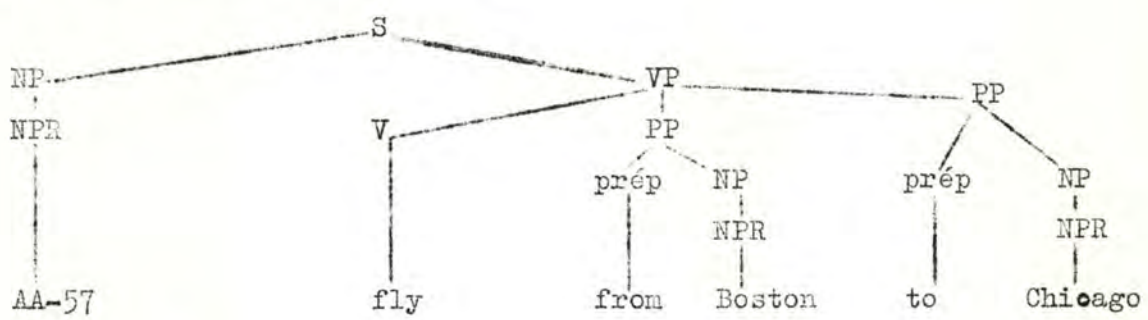
Cette méthode qui consiste à partir d'un ensemble de phrases est aidée par le fait que l'on possède déjà un analyseur syntaxique. Si on ne possède pas ce dernier, on pourrait procéder comme suit:

Règles sémantiques.
|
Sous - arbres.
|
Phrase - marker.
|
Phrase anglaise.

Cette méthode suppose que l'on construit son propre analyseur syntaxique, afin de décomposer les phrases en sous-arbres déterminés par l'analyse des règles sémantiques.

Regardons l'exemple classique :

AA - 57 FLY FROM BOSTON TO CHICAGO



On en connait le résultat :

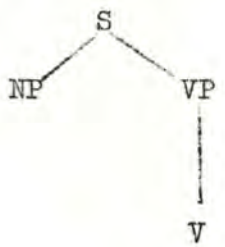
CONNECT (AA - 57, BOSTON, Chicago)

Partons de l'analyse des relations entre les objets de la base. Nous avons une proposition qui pourrait relier le vol, son lieu de départ et son lieu d'arrivée. C'est CONNECT.

Nous allons essayér de décomposer cela en sous-ensembles tout en respectant la structure d'une phrase anglaise simple. Cela nous donne : groupe sujet - verbe

groupe verbe - complément

Nous pouvons mettre ce résultat sous forme d'arbre :

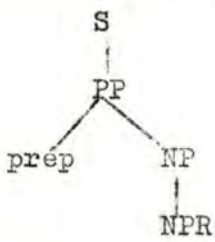


(1)

(2)

Le groupe (2) peut se répéter un certain nombre de fois.

Pourquoi ne pas prendre pour (2), la forme suivante :

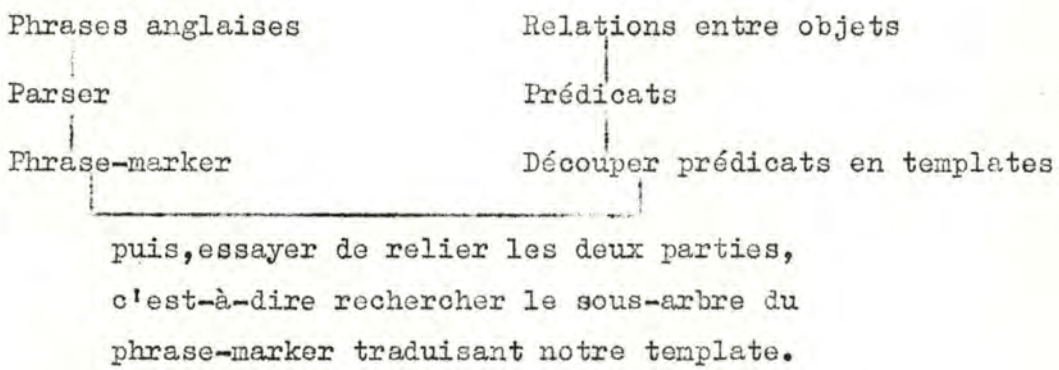


On a supprimé la relation avec le verbe (VP). Cela ne traduirait pas correctement la relation entre les objets puisque le complément en question est manifestement relié au verbe. En regroupant ces sous-arbres, on obtient la structure classique donnée à la page précédente.

Tout le problème consiste alors à construire un analyseur syntaxique qui serait capable, en partant de la phrase anglaise,

de nous générer l'arbre correct. Il faudrait faire cette opération pour toutes les règles sémantiques que nous aurions déterminées auparavant en partant des relations que l'on désire établir entre les objets de la base. Si nous partons d'un analyseur existant, il n'est pas certain que nous trouverons le phrase-marker que nous avons construit.

Nous pourrions aussi essayer un processus se déroulant en deux phases parallèles :



La démarche consistant à partir des phrases anglaises nous évite de reconstruire un analyseur syntaxique. Cet être informatique existant déjà, on peut évidemment s'en servir. Lorsque l'on sélectionne les phrases anglaises, on le fait en fonction des relations que l'on désire trouver entre les objets de la base. Il y a toujours le risque de ne pas obtenir un ensemble de phrases suffisament représentatif, et aussi d'oublier certaines règles sémantiques. La démarche en deux processus parallèles semble minimiser cet état.

3. Rapport entre le langage et la base.

On appelera langage, au sens de Woods, le subset anglais qui est reconnu par l'analyseur syntaxique.

S O C R A T E.

Le langage Socrate n'est pas influencé par le contenu de la base de données. Il est évident que certains ordres seront refusés s'ils n'ont pas d'interprétation sémantique. Mais c'est du point de vue syntaxe que nous allons regarder ces langages.

Le même langage Socrate peut être employé pour une infinité de banques de données différentes, tout comme Fortran peut être utilisé pour mille et un programmes différents.

De plus, si Socrate permet de formuler différemment une même requête, toutes ces expressions seront acceptées sémantiquement si la hiérarchie est respectée.

Cette forme de langage, bien que difficile en première étude, devient assez vite familière à l'utilisateur. C'est évidemment un ensemble d'expressions qui ne sont pas automatiques, mais nous croyons qu'il n'est pas exclu de voir un utilisateur de Socrate, après un certain temps de travail courant avec ce langage, arriver à s'exprimer plus ou moins facilement dans la syntaxe imposée.

Nous avons souvent assimilé Socrate à Fortran pour classer son niveau dans la suite des langages de programmation. Mais il est certainement plus facile que ces langages de haut niveau, même que Cobol qui est pourtant un langage très parlant.

Il est certain que Socrate est lié à la base par sa structure de définition, mais c'est un lien sémantique, comme il en existe dans tout langage. Il n'en demeure pas moins que Socrate garde une certaine indépendance par rapport à la base, indépendance que l'on pourrait caractériser en déclarant que la grammaire est indépendante du contenu de la base de données. Cela est certainement un avantage considérable à l'actif de ce langage.

W O O D S.

Nous avons aussi, parmi le subset anglais admis, une limitation par l'analyseur sémantique, mais encore une fois, cela se retrouve dans tous les langages.

Nous ne pouvons pas affirmer que toute phrase acceptée par la syntaxe et reconnue comme équivalente à une autre déjà acceptée, ne sera pas rejetée par l'analyseur sémantique. Cela simplement parce qu'une règle sémantique n'aura pas prévu l'emploi d'un certain verbe comme synonyme d'un autre, oubli d'une règle.

C'est ce problème de synonymes qui peut être assez important. En effet, un terme A peut être synonyme d'un autre terme B pour une certaine catégorie d'individus et ne pas l'être pour une autre.

Exemple : ordre et instruction seront synonymes pour une majorité d'individus, mais auront un sens complètement différent pour un militaire.

Evidemment, ce problème pourrait être résolu en laissant faire cette analyse par les personnes intéressées par le projet, mais est-ce toujours possible? Cela montre encore que ce langage naturel est fortement lié à la base, relativement à son contenu.

Certes, on pourrait envisager une grammaire engendrant la presque totalité de la langue, du moins dans son vocabulaire usuel. Mais il faudrait quand même tenir compte de l'environnement des utilisateurs pour déterminer d'éventuels synonymes plutôt que sur des définitions académiques.

Il est évident que le subset choisi est lié au contenu de la base, de par sa grammaire. En effet, dans cette grammaire, on décrit l'alphabet terminal en fonction du vocabulaire anglais que l'on désire admettre, en se limitant à ce qui est nécessaire.

Si on désire reprendre l'anglais pour consulter une autre base de données, il faudra déterminer un nouveau subset de cette langue. La grammaire devra donc être assez fortement modifiée, du moins en ce qui concerne le vocabulaire technique. Cela n'est cependant pas un travail très lourd, mais plutôt ennuyeux. La création d'une structure de définition comme dans Socrate serait certainement plus sûr, mais demanderait une analyse assez longue, et aussi précise que le serait la recherche de prédicats et de règles sémantiques.

Les opérateurs qui auront à manipuler ce langage devront de toute façon connaître le vocabulaire admis, mais n'auront pas comme dans Socrate, une structure à mémoriser dans son entièreté. Il leur suffira, en général, de s'exprimer le plus simplement possible dans le vocabulaire reconnu.

Quant aux synonymes, y-a-t-il un réel handicap à admettre par exemple : AA - 57 VOLE DE BOSTON A CHICAGO

" VA "

et non " VOYAGE "

" TRANSITE "

" ASSURE LE TRANSPORT "

4. Possibilités offertes par ces langages.

S O C R A T E.

Le système Socrate offre à l'utilisateur des actions autres que la consultation d'une base de données. Il permet les opérations de définitions d'une base

création d'une base
mise à jour des objets de la base
suppression "
recherche "

La définition consiste simplement à entrer une structure dite "structure de définition" qui est un modèle, une image du fichier. C'est une image au point de vue réalisations entre objets et non au point de vue stockage sur unité secondaire.

La création consiste à garnir le fichier, c'est-à-dire à générer une ou plusieurs réalisations de toutes ou de certaines entités données par la structure de définition.

Ces deux opérations ne demandent pas un langage spécial, si ce n'est la spécification de la fonction de définition ou de création. On pourrait réaliser cela sans connaître le langage Socrate, si ce n'est les types de caractéristiques que ce système admet.

Pour ce qui est de la modification, suppression et recherche, ce sont toujours les opérations élémentaires d'un langage travaillant sur les bases de données. C'est seulement ici qu'intervient le langage lui-même. Il est à remarquer que, quelle que soit celle parmi ces opérations qui est demandée, c'est toujours le même langage qui est employé. En effet, le système regroupe toutes ses actions dans un seul et même langage qui est appelé "langage de requêtes" .

C'est seulement une suite de caractères qui servira à préciser l'action que l'on désire réaliser sur la banque de données. Cette suite peut évidemment être réduite à un seul caractère. La citation qui permet de désigner l'objet concerné est toujours la même.

En effet, soit la citation :

UNE PERSONNE AYANT NOM = "DUPONT";

Si on désire rechercher cette personne, on aura :

I UNE PERSONNE AYANT NOM = "DUPONT"; ?

Si on désire la supprimer, on aura :

S UNE PERSONNE AYANT NOM = "DUPONT" ; ?

Si on désire modifier la profession de cette personne, on aura :

M PROFESSION DE UNE PERSONNE AYANT NOM = "DUPONT";
= "EMPLOYE" ?

Donc, pour remplir les différentes fonctions, il suffit d'ajouter au langage de "citations" une série de caractères qui détermineront ces ordres.

W O O D S.

Dans ses notes sur un langage naturel, Woods ne s'intéresse qu'au langage de consultation. Essayons de voir sommairement ce qui pourrait être fait pour étendre ce langage à d'autres fonctions.

En ce qui concerne la définition et la création de la base, on pourrait utiliser une définition et génération de fichier selon une méthode déjà connue, par exemple, une procédure analogue à celle du Cobol, et même de Socrate. Rien n'empèche d'avoir une structure de définition, même si on ne structure pas le langage.

En ce qui concerne les opérations de modification, suppression et recherche, il nous suffirait de compléter le langage par quelques ordres relatifs à ces opérations. Nous avons vu dans Socrate que tout ce qui concerne les citations n'est pas modifié. Ce ne serait pas un grand problème que de faire ces ajouts.

Il ne serait donc pas très difficile de compléter Woods pour en faire un langage complet de data base orientée vers une application particulière. Cela est facilité par le fait que définition et création sont indépendantes du langage, et par le fait que modification, suppression et recherche sont sans influence sur le langage de citation.

5. Extensions.

S O C R A T E.

Une extension assez intéressante, bien qu'obligatoire, du système Socrate, est l'emploi de macros. Il est possible ainsi de définir un set d'instructions, de lui donner un nom. Cela permet de créer de petits programmes pour des opérations qui sont souvent utilisées et qui demandent un nombre d'instructions assez important, ou dont la programmation peut être complexe à réaliser.

Du côté langage lui-même, on peut réaliser les opérations types d'un langage de consultation de data base. Quelques extensions sont prévues quant à l'acceptation et l'adaptation de fichiers Cobol, nécanisme de protection des fichiers (droit de l'utilisateur).

En ce qui concerne la structure de définition, il est prévu que l'on pourra ajouter des entités en fin de structure définie initialement. L'équipe Socrate envisage aussi la possibilité de permettre à l'utilisateur de se définir une structure "de travail". C'est-à-dire, qu'il y aurait une structure commune à tous les utilisateurs, et à celle-ci s'ajouteraient des petites structures propres à chaque utilisateur.

Actuellement, une recherche est faite dans le but d'assurer une sécurité suffisante aux fichiers. Il s'agit d'un mécanisme qui sauverait périodiquement le contenu de chaque base en vue de garder une trace correcte de la base en cas d'ennuis.

W O O D S.

Dans ce système de langage naturel, on peut aussi utiliser des macros, mais ce n'est pas un besoin aussi impératif que dans Socrate. C'est simplement un outil mis à la disposition de l'utilisateur pour lui faciliter ce travail.

Du côté langage, il ne serait pas difficile d'étendre les possibilités relativement à une base. En effet, en ce qui concerne la grammaire, il suffirait de réajouter quelques termes anglais à l'alphabet terminal. Pour les règles sémantiques, la modification est aussi facile. Tout d'abord, il faudrait compléter le dictionnaire attaché à l'analyse sémantique. Ensuite, ajouter dans les règles existantes, le ou les termes nécessaires, et ce, par de simples opérateurs logiques qui les relieraient aux termes déjà présents dans les templates. Si on ajoute d'autres règles sémantiques, il s'agirait encore de les enregistrer dans le dictionnaire lié à l'analyse sémantique, et ce, pour chaque terme dont la règle ajoutée serait susceptible d'interpréter une phrase contenant ce terme.

On peut aussi être amené à réajouter de nouveaux prédicats, engendrés par des nouvelles relations entre des objets que l'on devrait ajouter à la base et ceux qui y sont déjà, ou entre ces nouveaux objets. Ici, les problèmes sont plus complexes. En effet, si la recherche de ces nouveaux prédicats est plus pénible et longue que difficile, il ne faut pas oublier que ces prédicats sont liés à une routine de recherche. C'est la programmation de cette routine qui reste le plus gros problème.

Le travail d'ajout sera peut-être long, mais il est peu probable que de telles opérations soient fréquentes.

6. Remarques sur l'influence de la taille de la base sur les deux systèmes étudiés.

Nous étudierons cette influence relativement aux recherches à faire pour réaliser les systèmes Socrate et Woods.

Il est certain que le contenu de chaque base est limité par la place qui est réservée à cette base. Ce point ne sera plus repris dans le court exposé qui va suivre.

Nous allons faire cette étude en examinant quatre possibilités de présentation des bases :

1. On possède un nombre de caractéristiques assez petit, mais les réalisations des entités contenant ces caractéristiques sont assez nombreuses.
2. On a toujours un ensemble de caractéristiques petit, mais on envisage un assez grand nombre de relations entre ces éléments.
3. Le nombre de caractéristiques est élevé, et par le fait même, le nombre de relations entre ces dernières l'est aussi.
4. Le nombre de caractéristiques, de relations et de réalisations d'entité est élevé.

Nous allons étudier les rapports entre ces quatres possibilités et les éléments du système, c'est-à-dire :

1. L'alphabet terminal de la grammaire.
2. Les primitives prédicats et fonctions.
3. Le dictionnaire lié à l'analyse sémantique.
4. Les règles sémantiques.
5. La recherche dans la base.

Nous pouvons dès à présent constater que cette analyse sera essentiellement orientée vers Woods. Les cinq éléments ci-dessus le laissent prévoir. Cela est dû au fait que Socrate est peu influencé par la taille de la base, du moins en ce qui concerne

l'analyse faite ici. Nous discuterons ce point par la suite.

Les arbres partiels ne seront pas influencés par les quatre possibilités vues ci-dessus, car nous pouvons admettre que l'analyseur syntaxique est capable de "traduire" la plupart des phrases anglaises en utilisant l'ensemble des arbres partiels qui est déjà défini. On atteint assez rapidement le nombre maximum d'arbres partiels, et ce nombre n'est pas très élevé.

Les primitives de commande ne sont pas sensibles à ces quatre possibilités. Elles ne dépendent pas de la taille du contenu des bases.

S O C R A T E.

Nous avons vu que dans Socrate, le langage est indépendant du contenu de la base. Le seul lien qui existe entre le langage et la base est la structure interne qui retrace le schéma logique de la base.

Au niveau analyse syntaxique, les quatre présentations de la banque de données que nous avons citées avant sont sans influence. En effet, cette analyse a pour but de vérifier si les requêtes sont exprimées en respect avec la syntaxe de Socrate. Le vocabulaire de Socrate étant un ensemble de mots réservés, non tributaire du contenu de la base, il sera toujours le même, et l'analyseur syntaxique n'est pas modifié (c'est-à-dire que la grammaire soutenant cette analyse n'est pas influencée par ces considérations).

Du point de vue analyse sémantique, une certaine influence peut se faire sentir. En effet, si le nombre de réalisations est sans importance, il n'en n'est pas de même quant au nombre de caractéristiques et de relations. Ces deux éléments sont repris par la structure de définition, et c'est via cette structure que va se réaliser l'analyse sémantique. Plus nous avons de caractéristiques plus les relations entre elles sont nombreuses, plus l'analyse sémantique est longue. Vérifier si la structure est bien respectée (hiérarchisation de la citation), vérifier si les noms de nos caractéristiques sont bien ceux repris dans la structure de définition, sera une opération assez longue si les caractéristiques sont fortement liées. L'emploi des démonstratifs, comme nous l'avons vu, permet de simplifier la citation, et aussi l'analyse de celle-ci.

Du point de vue recherche des éléments dans la base, en vue de sortir les résultats de la requête, le nombre de réalisations des entités est un facteur intervenant dans le temps de réponse.

Cela se comprend aisément, puisque les filtres intervenant dans les requêtes porteront sur un plus grand nombre d'éléments. D'où, recherche plus longue dans la base avant de trouver la caractéristique vérifiant ce filtre.

En ce qui concerne le travail utilisateur vis-à-vis de ce système Socrate, c'est toujours la création d'une structure de définition valable qui importe. Plus elle est pensée en fonction des problèmes à résoudre, plus la programmation des requêtes sera facile.

Quant à l'équipe - réalisation du système, elle ne sera pas confrontée avec des problèmes supplémentaires si le nombre de certains éléments du système varie et surtout augmente. Il y aura toujours le problème d'espace à ne pas saturer, mais ce n'est pas ce point de vue "storage" qui nous intéresse ici.

W O O D S.

Nous allons, dans le cas de Woods, réaliser l'étude prévue en fonction des différents points énoncés en tête de ce chapitre.

A. Nombre de réalisations élevé. Peu de caractéristiques.

1. Au niveau de l'alphabet terminal de la grammaire attachée à l'analyse syntaxique, on aura un nombre assez élevé de termes. En effet, la grammaire soutient un langage qui est un subset de la langue anglaise, et elle est donc soumise aux différentes valeurs des caractéristiques, donc au nombre de réalisations . Nous retrouvons ici le fait que cette grammaire est fortement liée au contenu de la base. Cependant, cela n'apporte pas une modification importante, puisqu'elle ne concerne que l'alphabet terminal.

2. Le nombre de primitives et de prédicats n'est pas fonction du nombre de réalisations. Il dépend simplement du nombre de caractéristiques et des relations entre celles-ci. Nous pouvons donc considérer ce nombre de primitives comme normal, puisque non influencé par l'aspect de la base que nous examinons.

3. Le dictionnaire lié à l'analyse sémantique va lui aussi être assez volumineux. En effet, il doit contenir chaque mot du langage et chaque valeur des caractéristiques. A ces termes, on fait correspondre leurs classes, et éventuellement une série de noms de règles sémantiques. Nous pouvons dire que ce dictionnaire possède une taille qui va croître verticalement en fonction du nombre de réalisations des entités de la base si on donne cette

représentation du dictionnaire :

Mots	Classes et règles sémantiques.

4. Les règles sémantiques quant à elles ne sont pas influencées par ce nombre de réalisations. En effet, dans une règle, au niveau templates, intervient la classe et non la valeur de la caractéristique.
   Ex: FLIGHT et non AA - 57

5. Au niveau de la recherche, on peut affirmer que le temps de réponse augmente avec le nombre de réalisations, bien que celà dépende du nombre de filtres que renferme la requête. De plus, le nombre de réalisations est sans influence sur le nombre de routines de recherche à programmer.

Une banque de données ayant cette configuration pourrait donc facilement être traitée par le système proposé par Woods. En effet, le travail de recherche de primitives et de règles sémantiques, qui est le plus lourd travail de réalisation de ce système, n'est pas influencé par le nombre de réalisations.

La taille du dictionnaire n'est pas un très gros handicap. Un dictionnaire est assez facile à construire, et moyennant une bonne organisation et une bonne méthode de consultation, le temps de réponse restera honnête.

B. Peu de caractéristiques, de réalisations.
   Beaucoup de relations.

1. Au niveau de l'alphabet terminal, aucune influence ne sera ressentie. La grammaire attachée à l'analyse syntaxique n'est en aucun cas soumise au nombre de relations, vu que ce dernier ne joue pas sur le vocabulaire du langage et les valeurs des caractéristiques.

2. Le nombre de primitives sera assez élévé. En effet, chaque prédicat et chaque fonction reflètent une relation entre les éléments de la base, ce en excluant les prédicats attachés uniquement à la classe des caractéristiques (FLIGHT, CITY, ...). Ceci nous oblige à réaliser un travail important de recherche de ces primitives.

3. Le dictionnaire lié à l'analyse sémantique est influencé par ce nombre de relations. En effet, beaucoup de relations implique beaucoup de règles sémantiques, donc grossissement horizontal du dictionnaire. Cette augmentation ne doit cependant pas être trop importante. On ajoute simplement quelques noms de règles à certaines entrées du dictionnaire.

4. Le nombre de règles sémantiques va lui aussi augmenter. Il est fortement lié au nombre de relations entre les éléments de la base, c'est-à-dire au nombre de prédicats. Un travail important sera encore à entreprendre ici relativement à la détermination de ces règles sémantiques.

5. En ce qui concerne la recherche dans la base, nous aurons un assez grand nombre de routines à programmer, vu que le nombre de prédicats est assez élevé. Ce point est évidemment le plus négatif dans le système de Woods, car il semblerait que les routines doivent tenir compte de l'accès. Il n'est pourtant pas impossible que ce problème soit reporté plus loin dans le retriever, via une méthode générale d'accès. La programmation de ces routines ne serait plus que l'établissement des relations logiques, relativement à une sorte de structure de définition. Nous ne possédons aucun renseignements supplémentaires pour traiter ce point.

Une base possédant de telles propriétés semble ne pas convenir pour être traitée par un système tel celui proposé par Woods. Un langage structuré, non lié à l'application semble préférable en ce sens qu'il nous libère du problème de détermination des primitives, et de l'accès.

C. Beaucoup de caractéristiques. Beaucoup de relations. Peu de réalisations.

1. L'alphabet terminal de la grammaire liée à l'analyse syntaxique va être assez volumineux, car il doit contenir le nom de toutes les caractéristiques de la base. Ceci n'est cependant pas très gênant car la construction d'un dictionnaire est assez simple, et il existe des méthodes d'organisation et de consultation optimisées.

2. Le nombre de primitives va lui aussi être assez élevé. Il y aura d'abord les prédicats destinés à vérifier l'appartenance d'un objet à la base (flight(X)) et ceux-ci sont nombreux vu qu'il y a beaucoup de caractéristiques. Ensuite, il y a tous les prédicats attachés à l'expression des relations entre les caractéristiques, et ces relations sont nombreuses.

3. Le dictionnaire lié à l'analyse sémantique va croître verticalement et horizontalement. En effet, il y a beaucoup de caractéristiques, donc d'entrées dans le dictionnaire (augmentation verticale), et il y a beaucoup de relations, donc vraisemblalement de règles sémantiques (augmentation horizontale). Encore une fois, cet aspect n'est pas pénalisant pour autant que l'on organise bien ce dictionnaire.

4. Le nombre de règles sémantiques va lui aussi être assez élevé. Ayant beaucoup de caractéristiques et de relations

entre ces caractéristiques, il est certain que l'on aura besoin d'un set assez important de règles sémantiques pour atteindre le but recherché. La détermination de ces règles restera bien sûr une opération assez lourde pour la réalisation de ce système.

5. Ayant un nombre de primitives assez élevé, il en sera de même pour les routines de recherche.

Une telle base serait peu intéressante si elle était exploitée par le système proposé par Woods. C'est toujours le nombre de primitives et de règles sémantiques qui est déterminant dans ce choix.

D. Nombre élevé de caractéristiques, de relations et de réalisation.

D'après ce qui précède, on peut déduire qu'une base possédant les caractères ci-dessus serait difficilement exploitable par le système proposé par Woods. Le travail de réalisation d'un tel système serait trop lourd, et surtout dépendrait trop de l'environnement. Tout serait à refaire si l'on désire créer une nouvelle base.

CONCLUSION.

Il semble donc que le système proposé par Woods ne soit intéressant que dans la mesure où on a peu de caractéristiques et peu de relations entre elles. Le nombre de réalisations des entités étant sans importance. Cela restreint assez fortement le champ d'application d'un tel système. Il n'en demeure pas moins que pour certaines applications du genre gestion de bibliothèques, gestion d'une compagnie d'aviation, gestion d'une société de banque, de chemin de fer... ce système reste valable. Notons que gestion est pris ici dans un sens restreint. Il ne s'agit pas d'une comptabilité, mais d'une gestion vue du côté service rendu à la clientèle (liste d'ouvrages, horaires, consultation d'un compte bancaire par l'employé avant la remise d'une somme, ...) .

Si le retriever possédait un système général pour l'accès à la base, et que les routines de recherche n'étaient plus qu'une expression des liens logiques entre les caractéristiques, la proposition de Woods serait plus attractive. Nous n'avons aucune connaissance de l'existence d'une"data structure" qui permettrait cette programmation dans un langage structuré style Socrate. Nous ne possédons pas de renseignements sur le retriever. Nous ne pouvons donc nous prononcer avec certitude sur la façon dont les problèmes d'accès sont résolus par Woods.

Remarque :

Le terme "beaucoup" ainsi que le terme "nombre élevé"sont souvent employés dans cet exposé. Ils sont évidemment vagues quant à leur signification quantitative, car il nous est difficile d'estimee ce "nombre frontière" entre l'acceptabilité et le refus du système sans en avoir étudié une réalisation pratique.

=-=-=-=-=-=-=-=-=-=-=-=

7. Tableau récapitulatif.

SOCRATE.	WOODS.
-- Langage structuré.	Langage naturel.
-- Langage relativement difficile.	Langage simple.
-- Grammaire propre à Socrate donc, langage non lié à la base quant à son contenu	Grammaire basée sur une structure des phrases anglaises. donc, le subset choisi pour la consultation est lié à l'application par l'alphabet terminal de la grammaire.
-- Macros indispensables.	Macros utiles, mais pas indispensables.
-- Langage valable pour mille et une applications.	le vocabulaire est à redéfinir chaque fois.
-- Utilisateur n'est pas concerné par les problèmes d'accès.	Utilisateur peut être amené à s'intéresser à l'accès s'il ajoute de nouveaux prédicats. Programmation de nouvelles routines de recherche d'où appel à l'équipe de réalisation.
-- Création d'une structure de définition. Ce travail demande une analyse assez complexe.	Recherche de prédicats, construction de règles sémantiques. Ce travail est plus ennuyeux que difficile et demande une analyse aussi complète que la structure de définition.
-- Pas de problème de synonymes.	Problème des synonymes. Les accepte-t-on tous ou pas? Sont soumis à l'environnement utilisateur.

SOCRATE.	WOODS.
-- Utilisateur doit connaître un langage assez difficile Remarquons qu'après un certain temps de manipulation, Socrate s'adopte facilement.	Utilisateur doit connaître le vocabulaire considéré comme admissible.
Analyse syntaxique.	
-- Vérification des requêtes via la grammaire Socrate	Production d'un phrase-marker, tout en vérifiant si la phrase contient le vocabulaire admis. Cela, via une grammaire basée sur la structure des phrases anglaises.
Analyse sémantique.	
-- Vérification de la hiérarchisation des citations via la structure de définition.	Analyse du phrase-marker via des règles sémantiques sous-arbres prédicats dont la détermination est assez fastidieuse.
Structure.	
-- Il existe une structure qui reflète les relations logiques entre les objets de la base, de façon à faciliter la vérification sémantique de la formulation des citations.	On possède Dictionnaire Graphes partiels Règles sémantiques Ennuyeux à construire. Moins riche que la structure de définition. Risque d'oubli plus grand. A refaire chaque fois que l'on crée une nouvelle base, et même lors de certains ajouts.

S O C R A T E.	W O O D S.
Possibilités.	
— Définition Création Modification Suppression Recherche C'est simplement un ajout d'ordres au langage de citation.	Ces opérations sont facilement réalisables par le langage naturel. Le langage lui-même n'est pas concerné par ces opérations. C'est simplement un ajout d'ordres.
Extension langage.	
— Langage complet.	On peut facilement ajouter de nouvelles phrases. Les modifications ne sont pas compliquées. Plus complexe si ajout de prédicats, ce qui implique une programmation des routines de recherche ainsi ajoutées.

8. Conclusion.

En conclusion, nous pouvons dire que les deux systèmes ne sont pas si opposés qu'on le croyait au départ. Il est certain qu'ils ont chacun leurs avantages et leurs inconvénients.
Socrate s'adaptera mieux à certains travaux et Woods à d'autres.

Nous pensons pouvoir dire que si l'on connait la base et les relations entre les objets de cette base, on peut facilement employer Woods, qui aura l'avantage de donner un langage naturel de consultation. Socrate serait tout aussi valable. Mais si nous supposons que, quel que soit le système choisi, on doit le construire, Woods me semble plus simple à réaliser.

Si on désire réaliser un langage non dédicacé à une application particulière, une méthode analogue à celle employée par Socrate semble préférable. En effet, ce langage n'est pas lié au contenu de la base, et le système possède la possibilité de consulter une structure de définition, interface entre la base et le langage.

Vu l'emploi de dictionnaire nécessité par la méthode de Woods, ce procédé semble assez lourd si la base est trop volumineuse au point de vue du nombre de ses objets et des relations qui les unissent. Un langage style Socrate est préférable.

D'autre part, si une société achète un système de gestion de data base, et qu'elle compte par la suite créer de nouvelles bases, de structures différentes, il est préférable à ce moment de s'orienter vers un langage qui pourrait être commun à toutes ces bases, donc vers un système du style Socrate.

Un langage naturel tel que le propose Woods poserait trop de problèmes à chaque nouvelle création de base.

Comme il l'est dit dans ce travail, Woods ne donne aucun renseignement quant à l'accès à la base de données. Les primitives ARRIVE, CONNECT... sont-elles programmées directement dans un langage de base? Sont-elles simplement prises en charge par un algorithme général d'accès? C'est toute la partie dite "retriever" sur laquelle on ne peut se prononcer avec certitude, vu que Woods ignore complètement cette partie.

Pour compléter ce travail, nous avons décidé de regarder comment on pourrait résoudre ce problème de la signification des primitives quant à l'accès. Nous voulons réaliser ce passage en fonction d'éléments connus. C'est précisément le rôle de la seconde partie de ce travail. Cependant ce deuxième volet va se détacher de Woods, de façon à rendre plus souple le langage orienté naturel, et prendra comme élément connu, le langage Socrate.

=-=-=-=-=-=-=-=-=-=-=-=

2. ESSAI DE DEFINITION D'UN LANGAGE ORIENTE NATUREL

I. Introduction.

I.I. Présentation du problème.

Le problème que nous allons analyser se résume à ceci: construire un langage orienté naturel, indépendant de tout langage rigide. Cette optique nous est imposée par le fait que, dans le cadre d'un travail réalisé par l'Institut, ce langage rigide et sa structure ne sont pas connus. Il nous fallait donc trouver un moyen nous permettant de définir le langage naturel en ne connaissant que les relations qui existent entre les éléments de la base.

Nous avons été amené à considérer un langage LR et une structure associée sur laquelle on pourrait se baser pour réaliser LN. Bien sûr, il est nécessaire que cette structure intermédiaire soit indépendante de toute autre stru cture imposée par un langage rigide existant. Seul l'accès attaché à cette structure tiendra compte du langage rigide terminal.

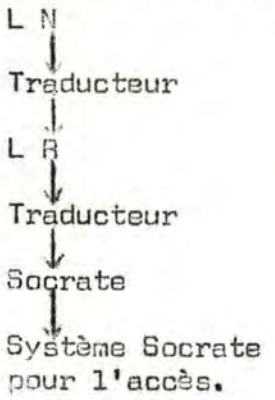
Nous avons choisi Socrate comme langage rigide, choix dicté par le fait qu'un stage effectué à Grenoble nous a permis d'étudier ce dernier. Il est certain que cette analyse resterait valable si un autre langage terminal avait été choisi. Seul l'aspect accès serait à modifier dans notre structure intermédiaire.

Il nous faut encore signaler que cette structure est l'élément capital de l'ensemble LR - structure intermédiaire. Le langage LR pourrait très bien ne pas exister. Nous l'avons défini ici, car il peut fournir un outil de programmation, mais il n'est pas indispensable pour le problème qui nous intéresse. Nous emploierons souvent LR, par abus de langage, pour désigner la structure intermédiaire à laquelle il est associé.

L'application choisie pour exposer ce travail sera une gestion de bibliothèques.

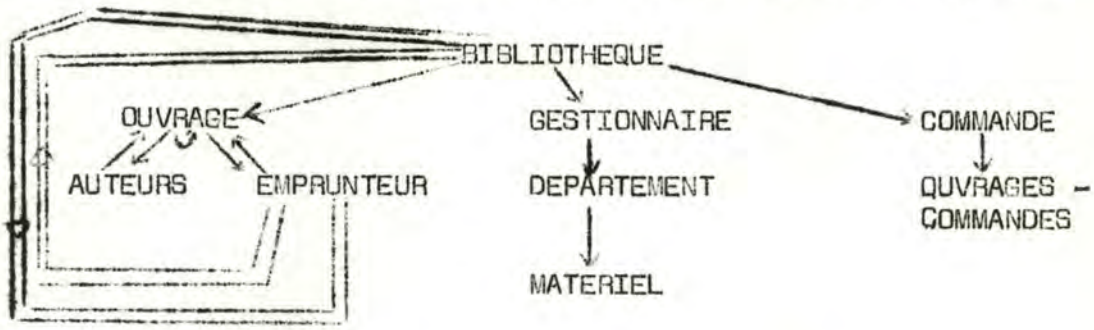
Dans l'état actuel du travail, nous n'avons pas eu la possibilité de passer au stade de la réalisation. C'est pour cette raison que nous resterons au niveau analyse du problème sans donner des codes aux éléments que nous utiliserons, sans parler de longueur ni d'organisation des tables employées.

Nous pouvons illustrer notre problème par le schéma suivant:



I.2 Description de l'application choisie.

L'application peut se représenter par le schéma suivant:

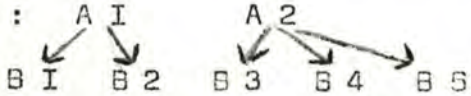


Cette base n'est construite que pour illustrer l'exposé qui va suivre. Il ne faut pas y voir un modèle pour une gestion de bibliothèques.

Dans ce schéma logique, nous avons pris les conventions suivantes:

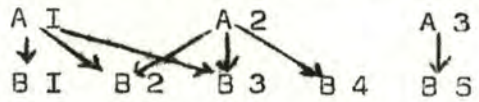
⟶ représente une relation (I N). Si nous avons A ⟶ B, à chaque occurence de A correspond un certain nombre d'occurences de B. Deux occurences distinctes de A sont disjointes relativement aux occurences de B qu'elles déterminent.

Soient Ai et Bi des occurences de A et de B. On aurait par exemple :



⟹ représente une relation (I N*). Si nous avons A ⟹ B, à chaque occurence de A correspond un certain nombre d'occurences de B. Deux occurences distinctes de A n'étant pas nécessairement disjointes relativement aux occurences de B qu'elles

déterminent. Nous aurons par exemple :



Chaque nom représente ce que nous nommerons une ENTITE.
Une entité n'est autre qu'un groupement d'items, groupement pouvant avoir plusieurs occurences ou réalisations.

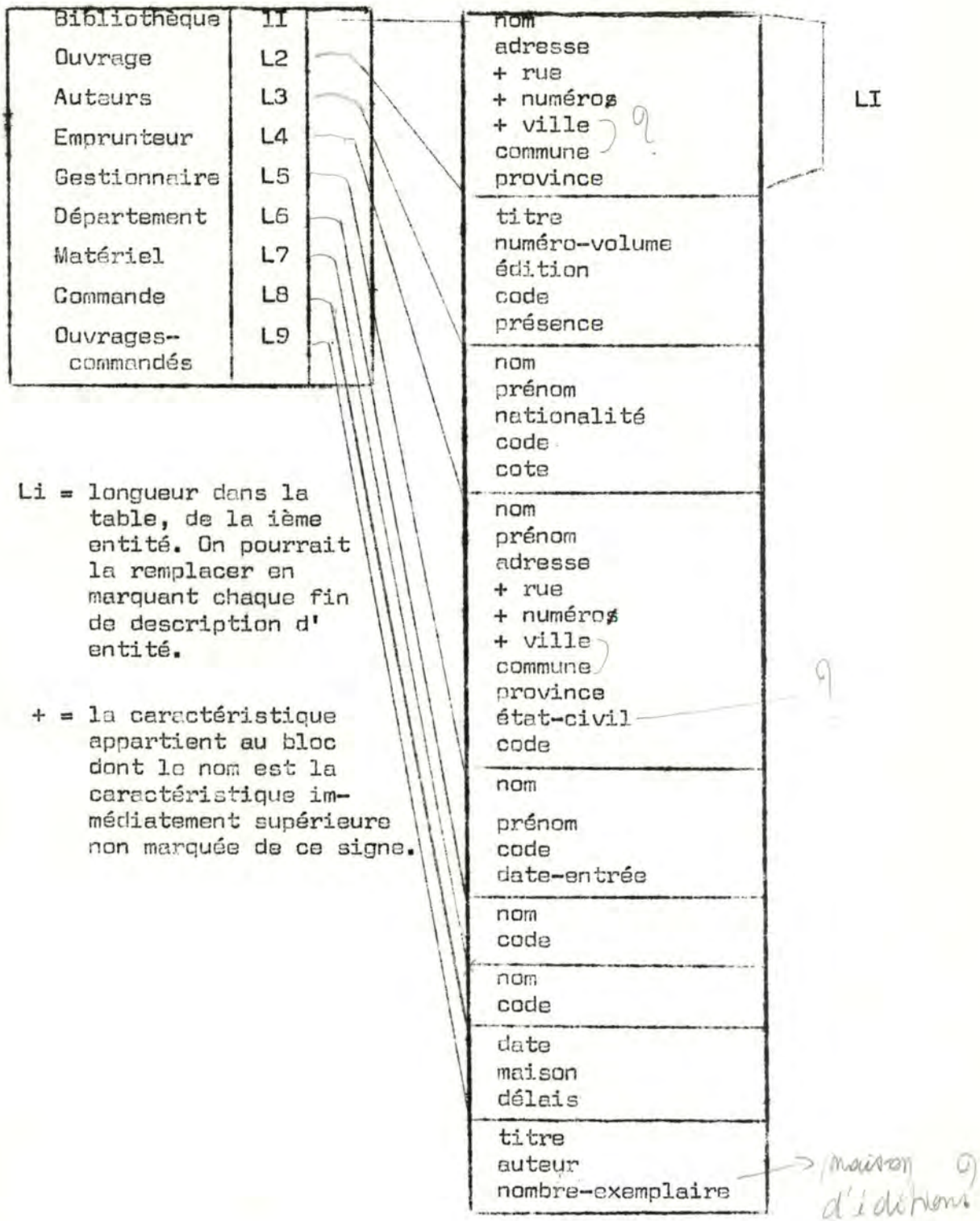
Les éléments qui sont nécessaires pour la suite sont :

- Un tableau descriptif de la base donnant les caractéristiques appartenant aux différentes entités.
- Une table des caractéristiques donnant sans redondance les caractéristiques composant notre data base.
- Une réalisation Socrate de cette data base.

Notons que les deux tables ne sont pas liées au fait que nous avons choisi Socrate comme langage terminal. Elles pourraient convenir pour tout autre langage.

Il nous faut maintenant décrire chacune des entités. Nous allons réaliser cela par ce que nous appelerons :

Tableau descriptif de la base.



On construit une table des caractéristiques. Vu qu'une même caractéristique peut se répéter plusieurs fois, la recherche dans le "Tableau descriptif de la base" peut être longue. La création d'une table contenant ces caractéristiques sans redondance rendra la recherche plus simple.

Table des caractéristiques.

nom
adresse
rue
numéro
ville
commune
province
titre
numéro-volume
édition
code
présence
prénom
nationalité
cote
état-civil
date-entrée
date
maison
délais
auteur
nombre-exemplaire

Ayant choisi Socrate comme langage terminal, nous allons donner la structure correspondant à notre application dans ce langage.Cette structure n'est pas unique et pourrait prendre une forme différente de celle décrite ci-dessous.

```
Début
| Entité Bibliothèque
|   Début   nom   mot
|   |       adresse
|   |         Début   rue   texte
|   |         |       numéro   de I à IOOO
|   |         |       ville   mot
|   |         Fin
|   |       commune   mot
|   |       province   mot
|   |       Entité Ouvrage
|   |         Début   titre   mot
|   |         |       numéro-volume   de I à IO
|   |         |       édition   mot
|   |         |       code  de I à IO 000
|   |         |       présence   ( O I )
|   |         |       Entité Bibliographie
|   |         |         Début   ouvrage   réfère un ouvrage de une
|   |         |         |                 bibliothèque
|   |         |         Fin
|   |         |       auteur   réfère un auteurs
|   |         |       emprunteur   réfère un emprunteur
|   |         Fin
|   |       Entité Gestionnaire
|   |         Début   nom   mot
|   |         |       prénom   mot
|   |         |       code  de I à I5
|   |         |       date-entrée   texte
|   |         |       Entité Département
|   |         |         Début   nom   mot
|   |         |         |       code   de I à 5
|   |         |         |       Entité Matériel
|   |         |         |         Début   nom   mot
|   |         |         |         |       code   de I à 5
|   |         |         |         Fin
|   |         |         Fin
|   |         Fin
|   |       Entité Commande
|   |         Début   date   texte
|   |         |       maison   texte
|   |         |       délais   de I à 25
|   |         |       Entité  Ouvrages-commandés
|   |         |         Début   titre   texte
|   |         |         |       auteur   mot
|   |         |         |       nombre-exemplaire   de I à I5
|   |         |         Fin
|   |         Fin
```

```
            Entité Emprunteur
              Début   emprunteur   réfère un emprunteur
              Fin
    Fin
  Entité Auteurs
    Début   nom   mot
            prénom   mot
            nationalité   mot
            code   de I à I50
            cote   de I à 20
            Entité Ouvrage
              Début   ouvrage   réfère un ouvrage de une bibliothèque
              Fin
    Fin
  Entité Emprunteur
    Début   nom   mot
            prénom   mot
            adresse
              Début   rue   texte
                      numéro   de I à I000
                      ville   mot
              Fin
            commune   mot
            province   mot
            état-civil   ( masc cel mar veuf div )
            code   de I à I000
            Entité Ouvrage
              Début   ouvrage   réfère un ouvrage de une bibliothèque
              Fin
            Entité Bibliothèque
              Début   bibliothèque   réfère une bibliothèque
              Fin
    Fin
Fin
```

2. Définition du langage naturel LN.

2.I. Présentation d'une grammaire.

Après avoir analysé une série de phrases françaises relatives à des questions pouvant être posées à une data base, nous avons pu déterminer quelques éléments fondamentaux "formes" et "mots de commande" revenant souvent. Nous en avons éliminé quelques-uns de façon à construire un ensemble de primitives relatives au langage naturel. La forme de cette grammaire sera voisine des présentations habituelles. Après classement de ces primitives nous obtenons :

a) Commande simple.

⟨func⟩ ::= ⟨fonction⟩ / ⟨item⟩

⟨commande simple⟩ ::=

⟨fonction⟩ [ET ⟨func⟩]* {⟨citation⟩} /

⟨item⟩ {ET ⟨item⟩}* {⟨citation⟩} /

{⟨qualifiant⟩} ⟨entité⟩ {ET ⟨entité⟩}* {⟨citation⟩} /

→ ⟨entité⟩ ::= nom d'entité

Notons que {A}* désigne un nombre quelconque de A.

{A}* désigne un A facultatif, mais s'il existe, il ne peut être présent qu'une seule fois.

b) Commande composée.

⟨commande composée⟩ ::=

QUEL EST ⟨commande simple⟩ /

VERIFIER POUR ⟨citation⟩ SI ⟨commande simple⟩ ⟨oper⟩ ⟨valeur⟩ { ET ⟨commande simple⟩ ⟨oper⟩ ⟨valeur⟩}*

VERIFIER SI ⟨commande simple⟩ ⟨oper⟩ ⟨valeur⟩ /

VERIFIER QUE ⟨commande simple⟩ ⟨oper⟩ ⟨valeur⟩ /

EFFECTUER ⟨fonction⟩ { ET ⟨fonction⟩ }* ⟨citation⟩ /

ETABLIR LISTE ⟨commande simple⟩ {⟨FS⟩} /

ETABLIR TABLEAU ⟨commande simple⟩ {⟨FS⟩} /

RECHERCHER ⟨entité⟩ {ET ⟨entité⟩}* ⟨citation⟩ /

RECHERCHER ⟨item⟩ {ET ⟨item⟩}* ⟨citation⟩ /

IDEM AVEC ⟨identifier I⟩ POUR ⟨identifier I⟩ /

PRECISER ⟨entité⟩

⟨identifier⟩ ::= chaine de caractères

⟨fonction⟩ ::= chaine de caractères reprise dans une table de fonctions.

⟨identifier⟩ ::= ⟨identifier⟩ /
⟨identifier⟩ {* ⟨identifier⟩}*

c) Conditionnel.

⟨Conditionnel⟩ ::=
SI ⟨cond⟩ ALORS ⟨commande⟩ SINON ⟨commande⟩

⟨cond⟩ ::= ⟨cond I⟩ /
⟨cond 2⟩ {ET ⟨cond 2⟩}*

⟨cond 2⟩ ::= ⟨commande simple⟩ ⟨oper⟩ ⟨valeur⟩

⟨cond I⟩ ::= TEST POSITIF /TEST NEGATIF /OUI /NON

⟨commande⟩ ::= ⟨commande composée⟩ /
⟨commande simple⟩

Remarquons que ⟨cond I⟩ est surtout employé après VERIFIER

d) Format de sortie.

Est spécialement utilisé avec ETABLIR

⟨FS⟩ ::= DONNER PAR ⟨item⟩ {ET ⟨item⟩}*
EN DONNANT ⟨item⟩ {ET ⟨item⟩}*

⟨ITEM⟩ ::= toute caractéristique reprise dans le tableau descriptif de la base.

e) Oper.

⟨oper⟩ ::= = / < / > / ≤ / ≥ / ≠

f) Langage naturel.

⟨LN⟩ ::= ⟨commande simple⟩ / ⟨commande composée⟩ /
⟨conditionnel⟩

De plus, nous terminerons toute requête par un point d'interrogation.

Remarques: I - IDEM ne porte que sur le langage avec lequel l'utilisateur travaille, c'est-à-dire, si un utilisateur programme en LN et s'il désire modifier partiellement sa requête, il peut utiliser IDEM. Dans ce cas, c'est une partie de la phrase LN qu'il modifiera et la traduction sera refaite pour cette requête considérée comme nouvelle par le système.

2 - On n'a pas tenu compte des qualifiants pouvant se trouver devant les items. Par définition, un item ne possède q'une seule occurence.

2.2. La citation.

Nous avons introduit dans la formalisation de LN la notion de "citation". C'est également en analysant des requêtes formulées en français que nous avons pu définir une citation, et obtenir le classement que nous allons donner. Mais avant cela, il convient de signaler une option assez importante que nous avons prise : toutes les relations entre les différentes entités de la data base seront nommées. L'utilisateur formulera sa question en tenant compte de ces noms. Ce choix va nous permettre d'accepter des citations non complètement définies, mais que le système pourra compléter grâce au nom de la relation.

Revenons aux citations et donnons-en la classification.

I - Objets qualifiés.

- qualificatif valeur.
  ex. : emprunteur de nom Jules.
- qualifié implicite.
  ex. : emprunteur Jules.
  On ne cite pas la caractéristique (ou item) qui possède la valeur suivant le nom de l'entité.
- qualificatif fonctionnel.
  ex. : nombre total.
  Le qualificatif "total" fait référence à une fonction qui est ici "somme".

- quantificateur.
  ex. : premier, dernier, tout.
- démonstratif.
  Ce démonstratif "Di" est introduit pour permettre à l'utilisateur de se référer à une entité déjà citée. On ne l'a pas repris dans la formalisation de LN. On le considèrera comme appartenant à une entité (entité = démonstratif ou démonstratif + nom d'entité.)

2 - Qualificatif.unique.

Une relation étant nommée, on peut, pour l'entité sommet de cette relation, en omettre le nom ainsi que la caractéristique concernée, en donnant directement une valeur.
ex. : nom de un emprunteur inscrit à BI ?
où "inscrit" est le nom de la relation entre "emprunteur" et "bibliothèque", et où BI est la valeur d'un item de bibliothèque.

3 - Qualificatif par "DE".

C'est ce mot réservé, DE, qui va nous permettre de relier une caractéristique à l'entité qui la possède. Il n'y a aucune ambiguïté à procéder de la sorte vu que toute relation est nommée. En effet, DE ne sera jamais employé pour relier deux entités. On interdit bien sûr l'emploi de DE comme nom de relation.

4 - Relation virtuelle.

On n'envisage pas ce cas dans ce travail, mais nous allons simplement signaler son existence. Il s'agit essentiellement de violer la règle obligeant l'utilisateur à nommer les relations. Un dictionnaire de synonymes nous permettrait de contrôler cela.
ex. :"emprunts de Jules" au lieu de "ouvrages empruntés par Jules".

Il est à remarquer que qualificatif recouvre ici une notion différente de celle habituelle. Par qualificatif on entend tout ce qui peut préciser une entité. Ce peut-être une caractéristique de cette entité, une valeur de cette caractéristique et même un synonyme fonctionnel.

Nous pouvons maintenant donner une définition formalisée de ce que nous appelerons une citation.

⟨qualifiant⟩ ::= ⟨integer⟩ / ⟨qual⟩ / TOUT / DERNIER / UN

⟨qual⟩ ::= ⟨integer⟩ PREMIER / integer DERNIER

⟨ident I⟩ ::= ⟨item⟩ { ⟨oper⟩ } ⟨valeur⟩

⟨rel I⟩ ::= { ⟨nom relation⟩ { ⟨qualifiant⟩ } } ⟨entité⟩ DE ⟨ident I⟩
{ ET ⟨ident I⟩ } $\{\}_1^\infty$ $\{\}_1^\infty$

⟨rel 2⟩ ::= { ⟨nom de relation⟩ ⟨valeur⟩ $\{\}_1^\infty$ /
{ ⟨nom relation⟩ { ⟨qualifiant⟩ } ⟨entité⟩ ⟨valeur⟩ $\}_1^\infty$

⟨rel 3⟩ ::= ⟨rel I⟩ / ⟨rel 2⟩

⟨rel 4⟩ ::= ⟨rel 3⟩ { ⟨rel 3⟩ $\}^*$ / ⟨rel 3⟩ { ET ⟨rel 3⟩ $\}^*$

⟨cit I⟩ ::= {DE} { ⟨qualifiant⟩ { ⟨entité⟩ } DE ⟨ident I⟩ } /
{DE} { ⟨qualifiant⟩ } ⟨entité⟩ DE ⟨ident I⟩ {ET ⟨ident I⟩ $\}^*$

⟨citation⟩ ::= ⟨cit I⟩ { ⟨rel 4⟩ $\}_1^\infty$ / ⟨cit I⟩ ⟨rel 2⟩ /
⟨cit I⟩ ⟨rel I⟩ / ⟨cit I⟩

⟨nom de relation⟩ ::= chaîne de caractères reprise dans une table.

Signification des symboles :

{ } = facultatif.

{ $\}_1^\infty$ = au moins une occurence .

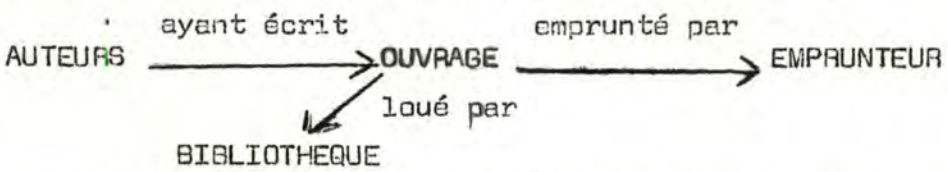
{ $\}^*$ = { $\}_0^\infty$ = un nombre quelconque d'occurences.

Signalons que l'utilisateur doit formuler ses requêtes avec certaines contraintes imposées par cette définition de LN. Mais cela ne diminue que de très peu le caractère naturel du langage LN. Nous le verrons par la suite, les noms de relations seront choisis de façon à ce que toute requête garde un sens vis-à-vis du Français.

Remarquons que "DE" est mis facultatif dans cit I . Cela est nécessaire dans le cas où on utilise Vérifier pour.

Notons que pour la formulation des requêtes, l'utilisateur peut utiliser plusieurs relations en parcourant le graphe de la "structure vue utilisateur" transposée. S'il n'est pas possible de la formuler en suivant le sens des flèches, on peut utiliser " ET".

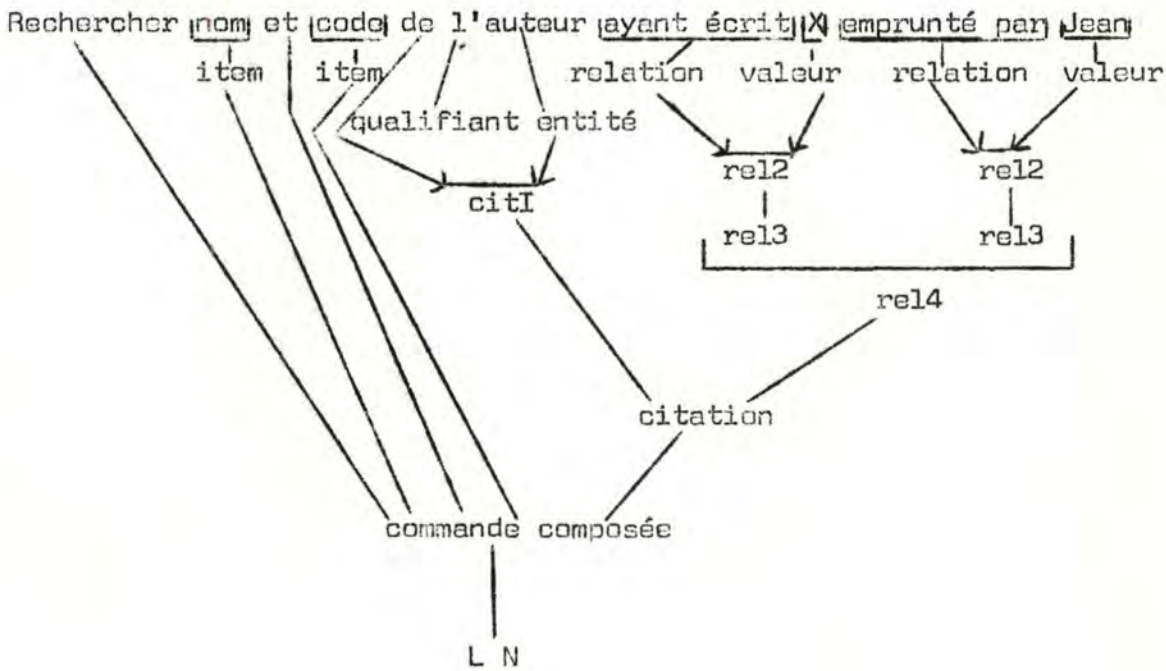
Ex. :

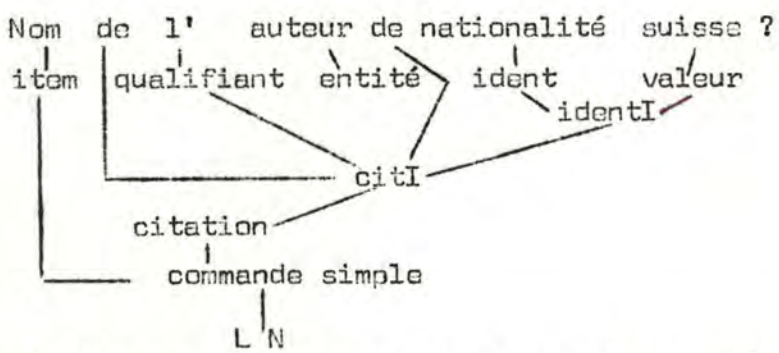


a) Quel est l'auteur ayant écrit "Les Saints vont en enfer" emprunté par Jules?

b) Quel est l'ouvrage emprunté par Jules et loué par la bibliothèque B I ?

2.3. Exemples.

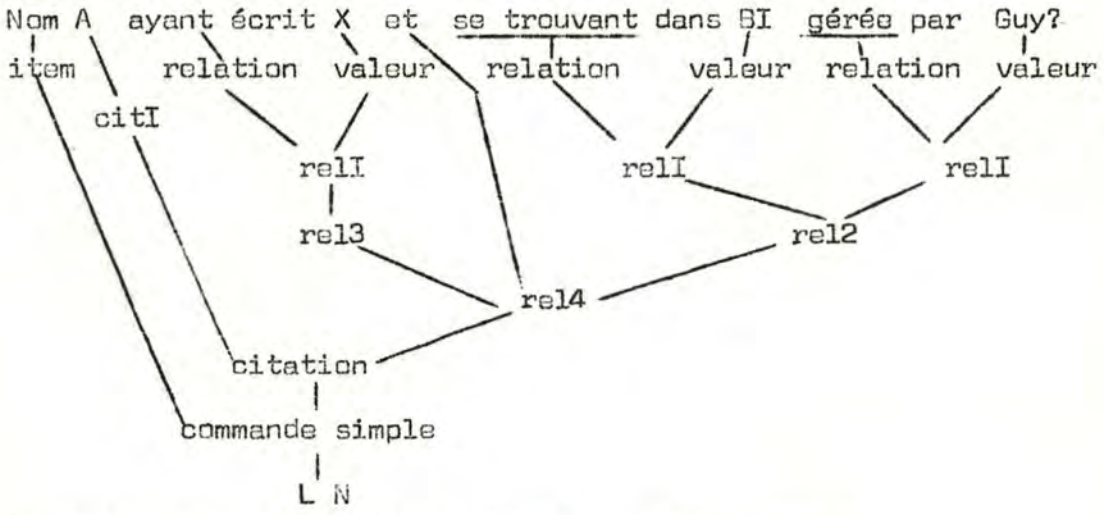


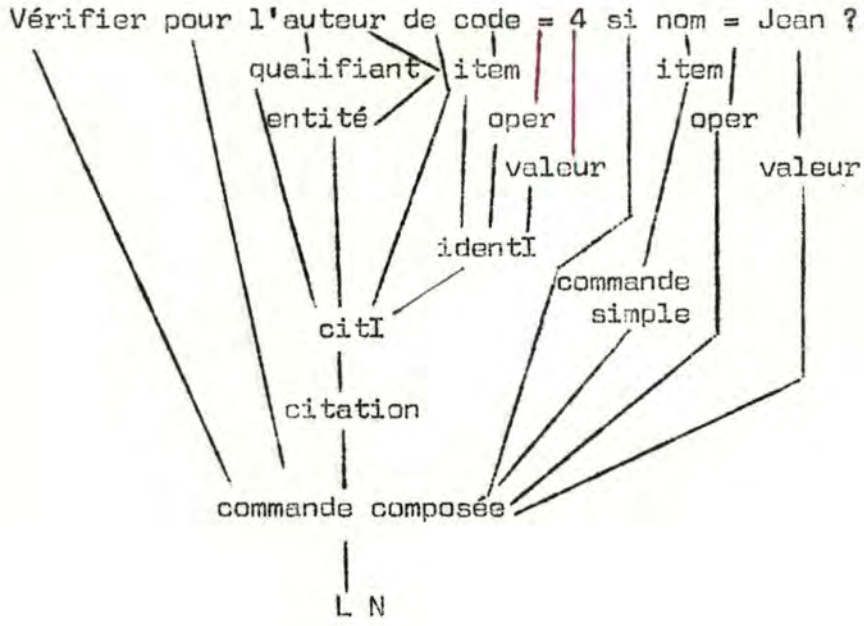


Rem: la "structure vue utilisateur" sera décrite ci-après en § 3.

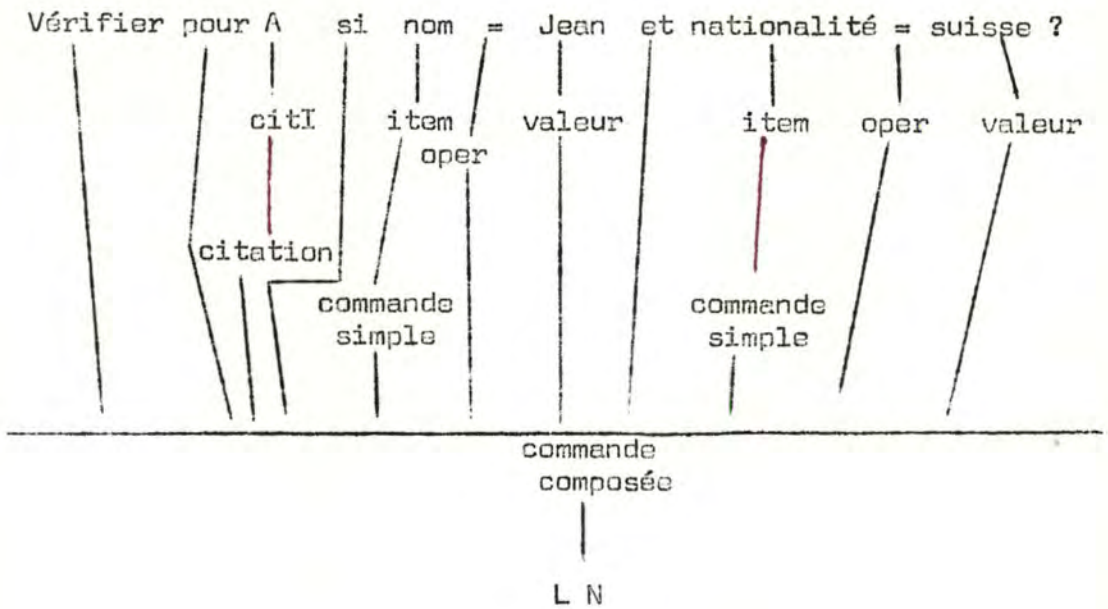
Si nous notons A pour : " de l'auteur de nationalité suisse" et reprenons sa décomposition jusque " citI";nous pourrons représenter la phrase : " nom de l'auteur de nationalité suisse ayant écrit X et se trouvant dans BI gérée par Guy" par " nom A ayant écrit...".

C'est cette phrase que nous allons analyser.





Si nous notons A pour : " l'auteur de code = 4" nous pourrons exprimer " vérifier pour l'auteur de code = 4 si nom = Jean et nationalité = suisse" sous la forme " vérifier pour A si nom = Jean et nationalité = suisse".

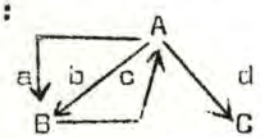


3. Etude d'une structure logique générale.

3.I. Structure vue utilisateur.

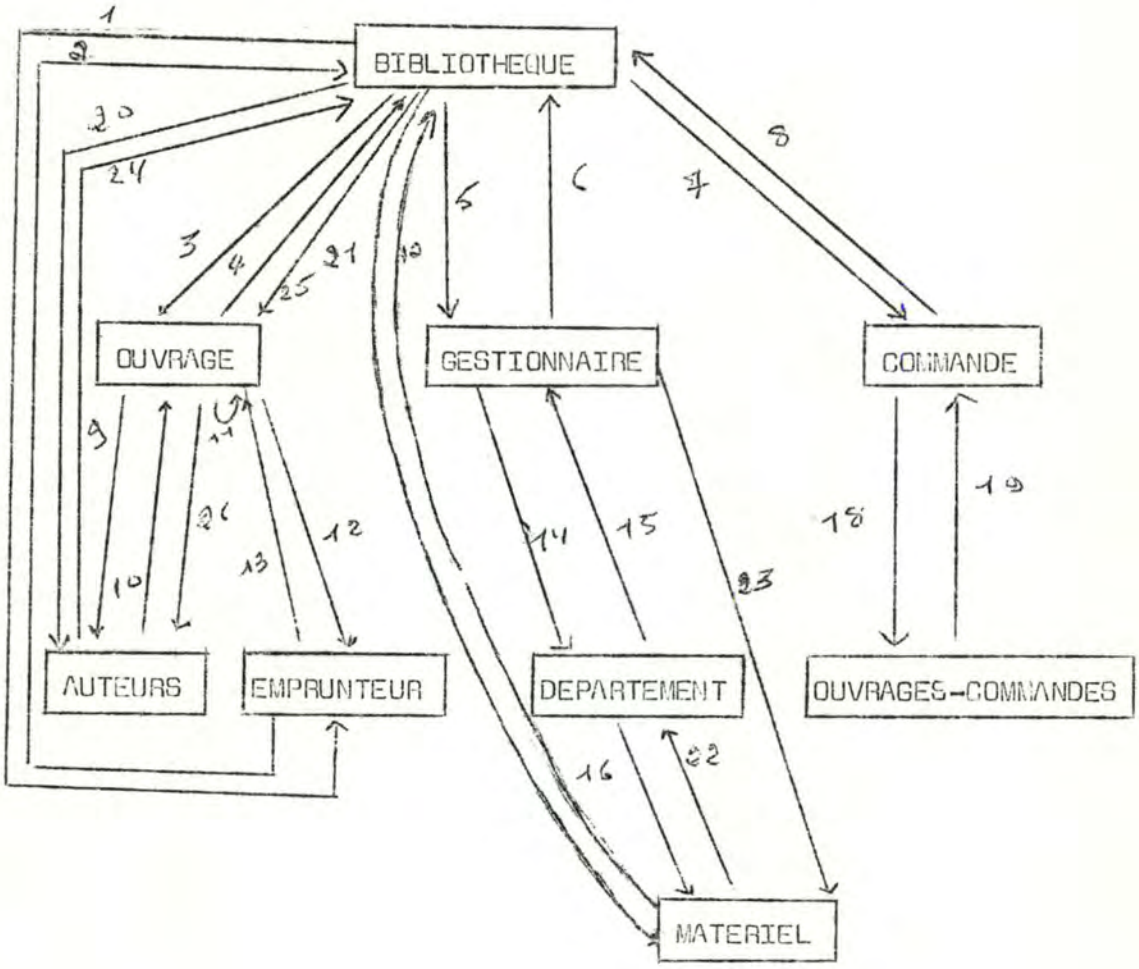
Dans le paragraphe précédent, nous avons choisi de donner un nom à toutes les relations entre entités, et de demander à l'utilisateur d'employer ces noms dans ses requêtes. Nous devons donc définir toutes ces relations, et les présenter à l'utilisateur de façon à ce qu'il puisse facilement rédiger ses questions. On va donc transformer le schéma représentant la data base et donné au paragraphe I. Nous allons créer ce que nous appelerons la "structure vue utilisateur". Elle va contenir toutes les relations que l'on accepte de traiter et qui sont donc offertes à l'utilisateur pour interroger la banque de données.

Certaines relations entre entités peuvent être enrichies d'un ordre , et deux entités pourront être reliées par des relations ayant des structures d'ordre différentes. Dans un tel cas, chacune de ces relations doit apparaître dans la "structure vue utilisateur". On pourra très bien avoir, dans cette "structure", un schéma identique à :



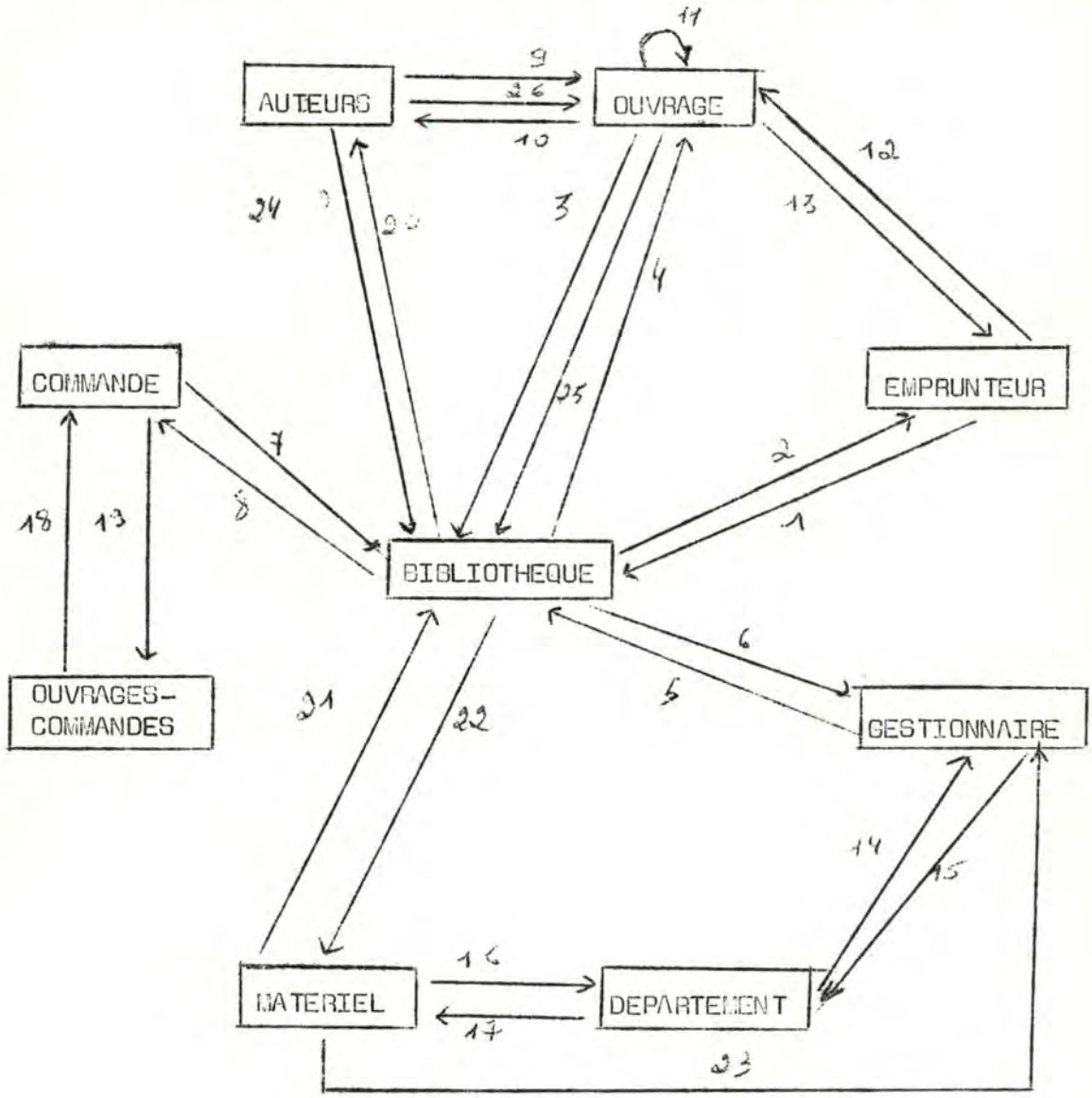
Nous pouvons constater que dans un schéma logique représentant un fichier, le sens des flèches est en général opposé à celui dans lequel il faut prendre leurs extrémités pour exprimer une relation. Comme nous voulons fournir une représentation facile à manipuler par l'utilisateur, il nous a paru intéressant de renverser le sens de ces flèches. Il suffirait alors de suivre le sens des flèches pour exprimer correctement une requête. C'est une telle représentation de la banque de données qui sera donnée à l'utilisateur. C'est une simple transposition de ce que nous avons appelé " structure vue utilisateur ".

Structure vue utilisateur.



Structure "vue utilisateur" transposée pour l'utilisateur.

Dans ce schéma, le numéro indiqué sur chaque flèche nous permet de faire la correspondance avec la "table vue utilisateur" que nous décrivons ci-après. L'utilisateur possédera en réalité un schéma où seront repris les renseignements que la table contient et pas seulement les numéros des relations symbolisées par ces flèches.



Outre les relations représentées dans la "structure vue utilisateur", l'utilisateur a la possibilité d'en créer d'autres à partir de celles qu'on lui a données. Il pourra les créer soit temporairement, c'est-à-dire le temps d'une question, et dans ce cas il ne leur donne pas de nom pour les stocker dans la table, soit définitivement ou du moins pour un certain nombre de questions, auquel cas il pourra leur donner un nom et les enregistrer dans une table. Cette table est la même que celle où seront stockées les relations définies au moment de la création du système.

3. 2. Relations nommées.

Pourquoi nommer toutes les relations? Cela nous permet de restreindre le vocabulaire français en un subset qui définira le langage naturel. Ce n'est pas un handicap pour le langage naturel car nous pouvons constater que les requêtes formulées en prenant ces noms garde une forme correcte vis-à-vis du français, aussi bien du point de vue syntaxique que sémantique.

Cela nous permet aussi de ne pas obliger l'utilisateur à citer toutes les entités qui interviennent dans sa requête. Nous l'avons vu dans les citations, le nom de la relation lève toute ambiguïté relative à l'entité concernée. On aurait pu prendre une autre position : ne pas nommer les relations, mais citer explicitement le nom de l'entité. La première méthode nous semble délivrer un langage plus naturel, et de plus, nous permet une analyse plus simple que la requête. La seconde méthode aurait été, nous semble-t-il, plus lourde dans la formulation des questions.

Prenons un exemple : si la relation "ayant écrit" établit un lien entre les entités "auteurs" et "ouvrage", nous aurons :

- Méthode I : Quel est le nom de l'auteur ayant écrit le Vieil Homme et la Mer ?
- Méthode 2 : Quel est le nom de l'auteur de l'ouvrage de titre Le Vieil Homme et la Mer ?

Si nous prenons des phrases plus complexes, la différence entre les deux méthodes est encore plus marquée.

- Méthode I : Quel est le nom de l'auteur ayant écrit Le Vieil Homme et la Mer emprunté par Jean inscrit à BI ?
- Méthode 2 : Quel est le nom de l'auteur de l'ouvrage Le Vieil Homme et la Mer emprunté par l'emprunteur Jean de la bibliothèque de nom BI ?

Bien que la méthode I est plus souple que l'autre, il y a encore un manque de souplesse dû aux noms des relations. Ces derniers rendent peu souple ce langage, mais il n'est pas exclu de le recouvrir par un ensemble de formes que l'on enregistreraient dans un dictionnaire des synonymes de façon à pouvoir se ramener, dans la phase de traduction, aux relations connues du système par une table que nous décrirons plus tard.

3.3. Table vue utilisateur.

Nous venons de décrire la structure associée au langage rigide que nous développerons dans le paragraphe suivant. Nous allons maintenant mettre cette structure à la disposition de l'ordinateur. Pour cela nous construirons ce que nous appelerons la "table vue utilisateur". Notons que dans cette table les relations seront données sous la forme classique, et non en inversant son sens comme nous l'avons fait pour la " structure vue utilisateur".

Cette table ne contiendra pas seulement les noms des différentes relations et des entités qu'elles relient, mais divers renseignements qui pourront être exploités par le langage rigide terminal qui sera choisi. L'utilité de ces renseignements dépend évidemment du choix de ce langage. Décrivons ceux que nous avons placés dans notre table.

- Un numéro d'ordre qui sera utilisé dans le passage de LN à LR. On désignera toute relation par son numéro d'ordre dans la table de façon à pouvoir la retrouver plus facilement lorsque nous en aurons besoin pour réaliser le passage d'un langage à un autre, spécialement pour retrouver les entités concernées par une relation donnée.

- Une caractéristique d'ordre qui nous indiquera quel est, pour l'entité sommet de cette relation, l'ordre établi entre les occurences de cette entité. Cet ordre dépend bien sûr du travail que l'on veut réaliser sur la data base. Notons que le fait d'avoir choisi Socrate comme langage terminal, nous donne un classement des réalisations d'entités selon l'ordre de création. Le système ne permet pas de préciser un ordre différent, et si ce dernier est nécessaire, c'est à l'utilisateur de le construire. Cela peut se faire au moyen de caractéristiques de type référence, mais le programme Socrate qui devra construire cet ordre ne sera pas très simple à réaliser. Une ligne blanche dans la table signifie que l'ordre est celui d'arrivée des occurences. Dans le cas contraire, l'ordre est celui indiqué.
- Un type est associé à chaque relation. Il s'agit des types (I N) et (I N^*) que nous avons définis dans un paragraphe précédent, et du type (I I) qui est un cas particulier de (I N).

  Ces renseignements ne seront pas exploités par Socrate. Ils pourraient nous donner quelques indications sur l'accès à une entité, mais comme nous prévoyons un renseignement-accès donnant la suite des entités nécessaires pour hiérarchiser la citation Socrate, nous ne nous servirons pas du type pour déterminer l'accès.

Il est nécessaire ici de remarquer que, pour avoir une structure Socrate bien adaptée au problème posé, une étude assez longue de ce problème est nécessaire. Il est même indispensable de prévoir ce qui sera demandé à cette structure après un certain temps de travail. Une structure peut être intéressante pour réaliser des traitements performants puis devenir assez rapidement inadaptée à un nouveau traitement.

Le dernier renseignement nous indique l'accès qui nous permettra de rechercher les éléments demandés. Il est nécessaire de souligner que, contrairement aux autres indications de cette table, l'accès dépend de Socrate, langage terminal choisi. Tout autre langage terminal demanderait une description différente de l'accès, mais le mécanisme que nous allons utiliser pour exploiter cette table reste le même.

Dans cette partie de la table, nous avons utilisé un certain symbolisme que nous allons expliciter.

- devant une entité implique qu'il est nécessaire de parcourir toutes les réalisations de celle-ci pour rédiger en Socrate l'accès que demande une relation.

  ex. : I. Si nous avons une relation BIBLIOTHEQUE ⟶ OUVRAGE et si on désire donner l'accès qui permettrait d'interpréter la requête suivante :
  Quelle est la bibliothèque possédant Le Vieil Homme et la Mer?
  On l'indiquera comme suit: ✳ BIBLIOTHEQUE, OUVRAGE car il faut parcourir toutes les réalisations de l'entité bibliothèque et sortir la valeur de caractéristique NOM de celle satisfaisant la condition sur l'ouvrage.
  On aurait: Pour toute bibliothèque xI
  si existe un ouvrage ayant titre = "Le Vieil Homme et la Mer"; alors I nom de XI
  fin fin ?

  2. Si on désire exprimer la relation: Quel est le titre de l'ouvrage de code 5 acquis par la bibliothèque BI?
  L'accès sera alors: BIBLIOTHEQUE , OUVRAGE
  On aurait en Socrate: Pour une bibliothèque ayant nom = "BI" ;
  Pour un ouvrage ayant code = 5;
  I titre
  fin fin ?

- Entre parenthèses, nous donnons la référence qui, dans l'entité précédent ces parenthèses, permet d'atteindre la caractéristique recherchée.

Si nous supposons une structure simple telle celle que nous allons donner, on pourra illustrer ceci par la requête
Quel est le titre de l'ouvrage emprunté par Jules?

Soit la structure

```
EMPRUNTEUR
     DEBUT ----
           NOM
           OUVRAGE
                DEBUT ----
                      OUVRAGE REFERE UN OUVRAGE DE
                      UNE BIBLIOTHEQUE
                FIN
     FIN
BIBLIOTHEQUE
     DEBUT ----
           ----
           OUVRAGE
                DEBUT ----
                      TITRE
                      ----
                FIN
     FIN
```

L'accès sera indiqué comme suit

EMPRUNTEUR, OUVRAGE (OUVRAGE)
entité référence

et on aura en Socrate : Pour un emprunteur ayant nom = "Jules" ;
I titre de ouvrage de un ouvrage
Fin ?

"Table Vue Utilisateur".

N°	NOM	ORIGINE	SOMMET	TYPE	ORDRE
I	inscrit	bibliothèque	emprunteur	IN*	code(n)
2	prête à	emprunteur	bibliothèque	IN*	
3	louer par	bibliothèque	ouvrage	IN	titre(a)
4	possédant	ouvrage	bibliothèque	II	
5	travaillant	bibliothèque	gestionnaire	IN	
6	géré par	gestionnaire	bibliothèque	II	
7	enrégistré par	bibliothèque	commande	IN	date(n)
8	ayant effectué	commande	bibliothèque	II	
9	ayant écrit	ouvrage	auteurs	II	
I0	écrit par	auteurs	ouvrage	IN	titre(a)
II	renseigné par	ouvrage	ouvrage	IN	titre(a)
I2	ayant emprunté	ouvrage	emprunteur	II	code(n)
I3	emprunté par	emprunteur	ouvrage	IN	titre(a)
I4	dépendre de	gestonnaire	département	IN	
I5	s'occupant	département	gestionnaire	II	
I6	disponible au	département	matériel	IN	
I7	qui dispose	matériel	département	II	
I8	commandé par	commande	ouvrage com.	IN	auteurs(a)
I9	demandant	ouvrage com.	commande	II	
20	se trouvant	bibliothèque	auteurs	9*3	
2I	appartient	bibliothèque	matériel	I6*I4*I5	
22	contient	matériel	bibliothèque	6*I5*I7	
23	comptabilisé par	gestionnaire	matériel	I6*I4	
24	détient	auteurs	bibliothèque	4*I0	
25	acquis	bibliothèque	ouvrage	IN	code(n)
26	ayant rédigé	ouvrage	auteurs	II	

"Table Vue Utilisateur" (suite)

	ACCES
I	bibliothèque, emprunteur (emprunteur)
2	emprunteur, bibliothèque (bibliotyèque)
3	bibliothèque, ouvrage
4	* bibliothèque, ouvrage
5	bibliothèque, gestionnaire
6	* bibliothèque, gestionnaire
7	bibliothèque, commande
8	* bibliothèque, commande
9	bibliothèque, ouvrage (auteurs)
I0	auteurs, ouvrage (ouvrage)
II	bibliothèque, ouvrage, bibliographie (ouvrage)
I2	bibliothèque, ouvrage (emprunteur)
I3	emprunteur, ouvrage (ouvrage)
I4	bibliothèque, gestionnaire, département
I5	* bibliothèque, * gestionnaire, département
I6	bibliothèque, gestionnaire, département, matériel
I7	* bibliothèque, * gestionnaire, * département, matériel
I8	bibliothèque, commande, ouvrages-commandés
I9	* bibliothèque, * commande, ouvrages-commandés
20	* bibliothèque, * ouvrage. (auteurs)
2I	bibliothèque, gestionnaire, département, matériel
22	* bibliothèque, * gestionnaire, * département, matériel
23	bibliothèque, gestionnaire, département, matériel
24	* bibliothèque, * ouvrage (auteurs)
25	bibliothèque, ouvrage
26	* bibliothèque, ouvrage (auteurs)

3.4. Table des liens entre entités.

Pour des relations n'existant pas dans la "Table vue utilisateur" on peut essayer de rechercher un chemin dans la structure pour interpréter cette relation. Pour cela, nous avons besoin d'une table supplémentaire permettant d'accéder à chaque entité. C'est la "table des liens entre entités".

Le symbolisme introduit dans cette table à la signification suivante :

- les parenthèses ont le même sens que dans la "Table vue utilisateur".
- * indique que l'entité concernée est située au niveau le plus externe de la structure, c'est-à-dire accessible directement.

La partie droite de cette table nous indique comment on peut accéder à l'entité située en partie gauche.

Toutes les tables que nous venons de décrire seront utilisées dans les deux passages que nous étudierons plus tard.

Par relation n'existant pas dans la "table vue utilisateur " on désigne :

- les relations formées d'un seul nom d'entité.
- une combinaison des relations de cette table.

Nous expliquerons cela dans le dernier chapitre, le passage à Socrate.

Table des liens entre entités.

bibliothèque	*
	emprunteur,bibliothèque (bibliothèque)
ouvrage	bibliothèque
	auteurs,ouvrage (ouvrage)
	emprunteur,ouvrage (ouvrage)
	bibliothèque,ouvrage,bibliographie (ouvrage)
gestionnaire	bibliothèque
département	bibliothèque,gestionnaire
matériel	bibliothèque,gestionnaire,département
commande	bibliothèque
ouvrages-commandés	bibliothèque,commande
auteurs	*
	bibliothèque,ouvrage (auteur)
emprunteur	*
	bibliothèque,ouvrage (emprunteur)
	bibliothèque,emprunteur (emprunteur)

4. Recherche d'un langage rigide intermédiaire.

4.I. Caractères de ce langage.

Ce langage LR doit être non ambigu et proche de LN, de façon à permettre une programmation facile.

Disons quelques mots sur une programmation directe en LR.

Il est nécessaire, puisque nous voulons permettre à l'utilisateur d'employer LR, de supprimer dans ce langage tout problème d'accès à la data base. Nous pouvons facilement admettre cela vu que le système Socrate va résoudre ce problème d'accès, en le laissant transparent à l'utilisateur.

Ce langage va s'appuyer sur la "structure vue utilisateur" décrite précédemment, et l'utilisateur aura à sa disposition le même schéma représentant la base que celui programmant en LN.

L'utilisateur à ce niveau, est de nouveau tenu d'employer les noms donnés aux relations. Il pourrait très bien donner le numéro d'ordre de la relation, car de toute façon, le nom sera remplacé par ce numéro.

Les noms des entités devront eux aussi être cités dans la forme LR de la requête. Ce langage est tel que nous le voulons complet dans toutes citations, même si le nom de l'entité sommet d'une relation et le nom de cette-dernière sont redondants.

Notons que l'on pourrait permettre à l'utilisateur programmant directement en LR de rédiger des requêtes dans une formulation plus souple que le système se chargerait de compléter.

Par exemple, il pourrait ne citer que les entités appartenant à sa requête, et le système, via le conversationnel, pourrait compléter par le nom des relations.

Nous nous intéressons uniquement à la consultation de la banque de données. Nous n'envisageons pas les problèmes de définition, création et modification de la base.

Dans le cadre de ce travail, nous ne nous intéressons pas à l'aspect programmation directe en LR. Le problème qui nous occupe est essentiellement le passage LN - LR - Socrate. Si nous avons introduit ci-dessus la programmation LR, c'est simplement parce que toute requête LR, après analyse donnera les mêmes tables que celles que nous

allons décrire dans la suite.

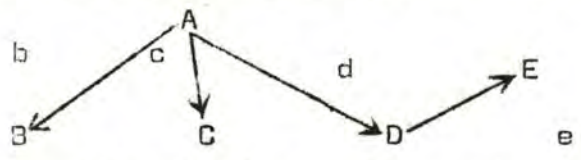
Pour notre problème, la structure vue utilisateur nous permettra de réaliser le passage demandé, mais le langage lui-même n'est pas un élément capital, c'est uniquement un outil supplémentaire que nous pourrions mettre à la disposition de l'utilisateur.

4.2. Le langage lui-même.

Fixons un vocabulaire pour LR. Nous le déterminons en donnant les quelques règles que nous allons utiliser pour générer LR.

- Le lien entre deux entités est réalisé au moyen de la préposition DE.
- Les caractéristiques "filtres" des entités se représentent par des citations reliées par ET, OU et sont placées entre parenthèses.
- Le nom de la relation suit toujours l'entité origine et est introduit par VIA.
- La préposition DE ne peut s'employer que si les relations se suivent dans le schéma que possède l'utilisateur. Dans le cas contraire, ou si plusieurs entités doivent intervenir relativement à la même origine, on utilisera DONT et on reliera ces entités par ET, OU.

Voici quelques exemples de requêtes formulées correctement en LR. Si nous prenons pour schéma donné à l'utilisateur



on pourrait avoir les requêtes suivantes, où x, y, z sont des caractéristiques filtrantes :

A DE D VIA d DE E VIA e ?

A (x ET y) DE C (z) VIA c ?

A DONT (B (x) VIA b ET C (y) VIA c OU D (z) VIA d DE E VIA e)

En LR, le mot de commande sera toujours présent dans une requête.

Ces mots sont : RECHERCHER
PRECISER
IDEM
VERIFIER

Notons que - PRECISER est toujours suivi d'un nom d'entité.
- IDEM n'est pas passé au langage suivant, c'est-à-dire Socrate. La modification impliquée par IDEM porte toujours sur la requête écrite dans le langage avec lequel on travaille, ici, LR, et non sur la forme finale qui serait ici la traduction Socrate.

L'ensemble des mots réservés à LR est le suivant :

ET
DE
VIA
RECHERCHER
PRECISER
VERIFIER
IDEM
AVEC
OU
DONT
POUR
DONNE PAR
EN DONNANT
SI
ALORS
SINON
(
)

4.3. Essai de formulation de LR.

Le symbolisme qui sera employé est le suivant :

{ } signifie facultatif

{ }* représente un nombre quelconque d'occurences

{ }$^{\infty}_{1}$ indique une occurence au moins

⟨ident⟩ ::= chaîne alphanumérique

⟨citation⟩ ::= ⟨ident⟩ { ET ⟨ident⟩ }* DE ⟨design⟩

⟨design⟩ ::= ⟨designI⟩ /
⟨design⟩ DONT (⟨expdont⟩) /
⟨design⟩ { ET ⟨design⟩ }$^{\infty}_{1}$ /
⟨design⟩ { OU ⟨design⟩ }$^{\infty}_{1}$

⟨designI⟩ ::= { ⟨ qualifiant⟩ } ⟨ident⟩ { (⟨expcomp⟩) }
{ VIA ⟨identI⟩ } /
⟨designI⟩ { DE designI }

⟨expcomp⟩ ::= ⟨ident⟩ ⟨oper⟩ ⟨ident⟩ { ET ⟨expcompI⟩ }* /
⟨ident⟩ ⟨oper⟩ ⟨ident⟩ { OU ⟨expcompI⟩ }*

⟨expcompI⟩ ::= ⟨ident⟩ ⟨oper⟩ ⟨ident⟩

⟨expdont⟩ ::= [⟨design⟩ { DONT (⟨design⟩) }$^{\infty}_{1}$ /
⟨design⟩

⟨oper⟩ ::= = / < / ≤ / > / ≥ / ≠

⟨exprech⟩ ::= RECHERCHER ⟨citation⟩ / ⟨citation⟩

⟨exptest⟩ ::= VERIFIER ⟨expver⟩ { ET ⟨expver⟩ }*
DE ⟨design⟩

⟨expver⟩ ::= ⟨ident⟩ ⟨oper⟩ ⟨ident⟩

⟨expidem⟩ ::= IDEM AVEC ⟨ident2⟩ POUR ⟨ident2⟩

⟨ident2⟩ ::= ⟨ident⟩ / ⟨ident⟩ { ⟨ident⟩ }*

⟨ordinal⟩ ::= DERNIER / PREMIER

⟨qualifiant⟩ ::= ⟨integer⟩ / ⟨integer⟩ ⟨ordinal⟩ /
TOUT / UN

⟨integer⟩ ::= ⟨digit⟩ / ⟨integer⟩ ⟨digit⟩

⟨digit⟩ ::= 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 0

⟨exppreciser⟩ ::= PRECISER ⟨nom entité⟩

⟨identI⟩ ::= nom relation / ⟨integer⟩

⟨nom relation⟩ ::= nom appartenant à la "table vue utilisateur"

⟨nom entité⟩ ::= nom repris dans le"tableau descriptif de la base", en partie gauche

⟨expsi⟩ ::= ET ⟨expver⟩ / OU ⟨expver⟩

⟨expcond⟩ ::= SI ⟨expver⟩ { ⟨expsi⟩ }* DE ⟨design⟩ ALORS ⟨expcondI⟩ { SINON ⟨expcondI⟩ }

⟨expcondI⟩ ::= ⟨exprech⟩ / ⟨expidem⟩

⟨LR⟩ ::= ⟨expcondI⟩ / ⟨expcond⟩ /⟨exptest⟩ / ⟨exppreciser⟩

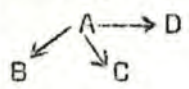
Sans se référer à une structure particulière, nous allons donner quelques formulations possibles en LR. Si nous désignons par A, B, C, D, des noms d'entités nous avons :

A de B de C
A dont (B et C et D)
A dont (B ou C et D)
A dont (B ou C ou D)
A dont (B et C de D)

Ce sont sous ces formes que sera traduite chaque phrase LN. Il est évident que LR permet d'autres formulations, mais nous nous limitons à celles-ci.

Nous allons montrer par quelques exemples que la plupart des formes LR se ramène à ces dernières.

I) A dont (B et C) de D se représente par



et donne A dont (B et C et D)

2) A dont (B de C) se représente par

A ⟶ B ⟶ C

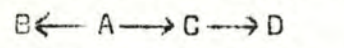
et donne A de B de C

3) A dont (B) de C se représente par

B ⟵ A ⟶ C

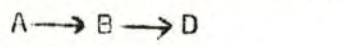
et donne A dont (B et C)

4) A dont (B et C dont D) se représente par



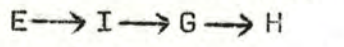
et donne A dont (B et C de D)

5) A dont (B dont(D)) se représente par



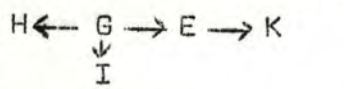
et donne A de B de D

6) E dont (I de G dont(H)) se représente par



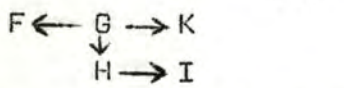
et donne E de I de G de H

7) G dont (H et I) de E dont (K) se représente par



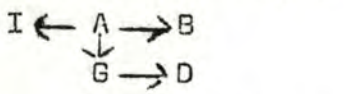
et donne G dont (H et I et E de K)

8) G dont (F et H dont (I) et K) se repr ésente par



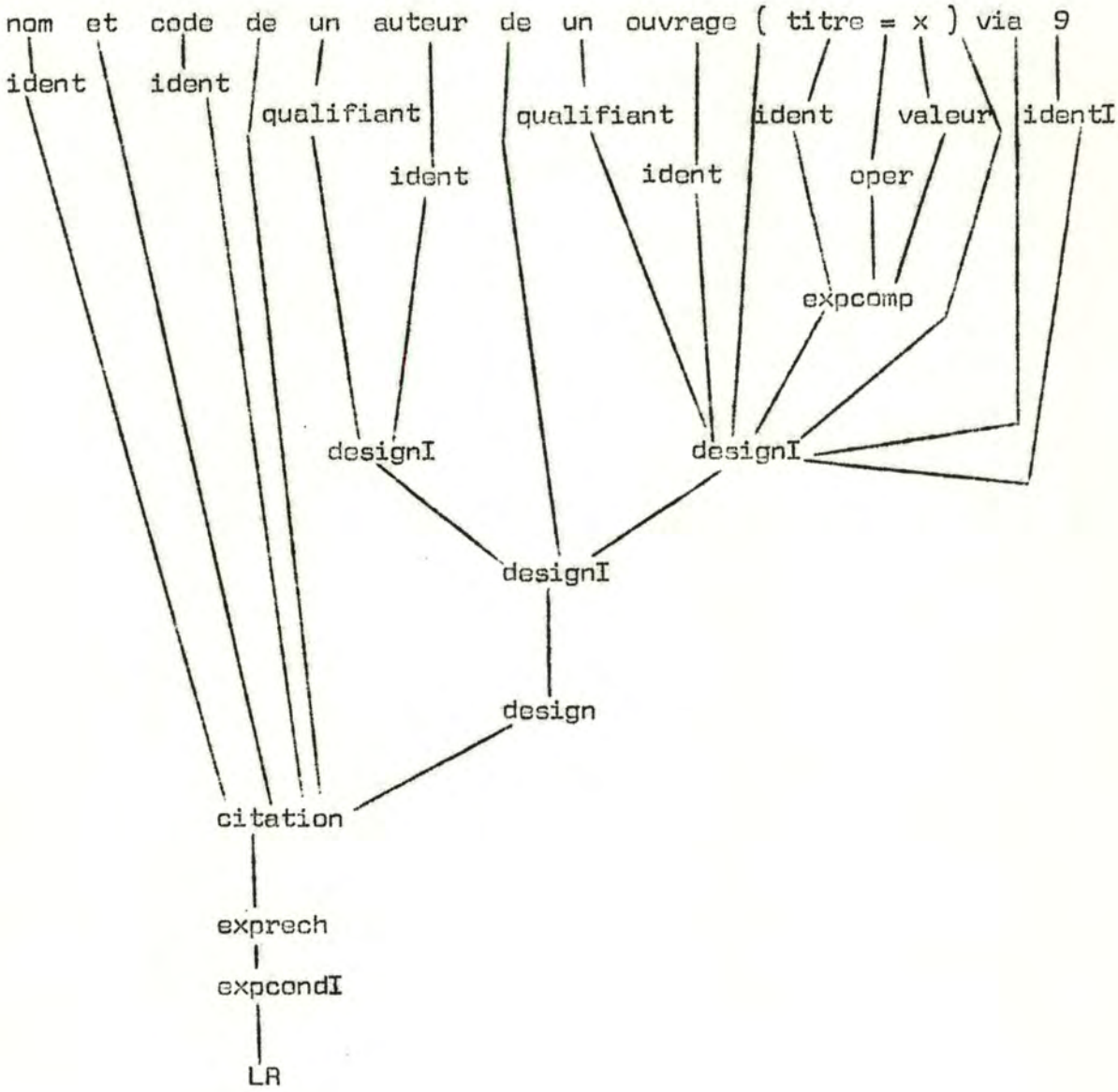
et donne G dont (F et H de I et K)

9) E dont (I et G ou B et G) dont (D) se représente par



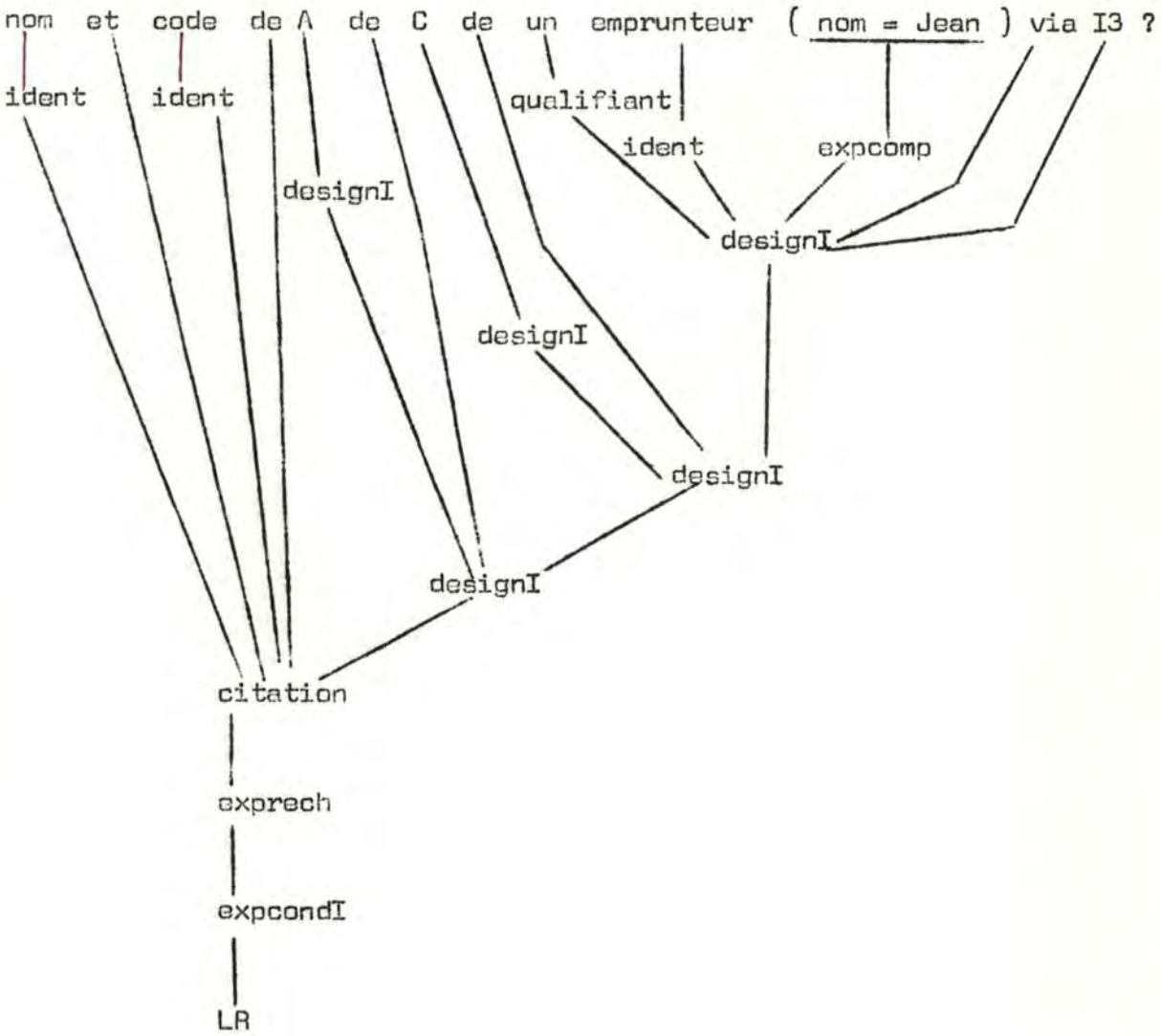
et donne A dont (I et G de D ou B et G de D)

Exemples.


nom et code de un auteur de un ouvrage (titre = x) via 9
ident ident
qualifiant
ident
qualifiant
ident
ident
oper
valeur
identI
expcomp
designI
designI
designI
design
citation
exprech
expcondI
LR

Si nous désignons par A " un auteur " et par B " un ouvrage (titre = X) ",nous povons formuler la requête suivante:
nom et code de un auteur de un ouvrage (titre = X) via 9 de un emprunteur (nom = Jean) via I3 ? par
nom et code de A de B via 9 de un emprunteur (nom = Jean) via I3 ?

Nous venons de voir que A représente designI et que B via 9 que nous noterons C représente aussi designI .La requête peut donc se formuler comme suit:



Nous allons encore analyser deux requêtes que nous simplifions comme suit,d'après les exemples précédents:

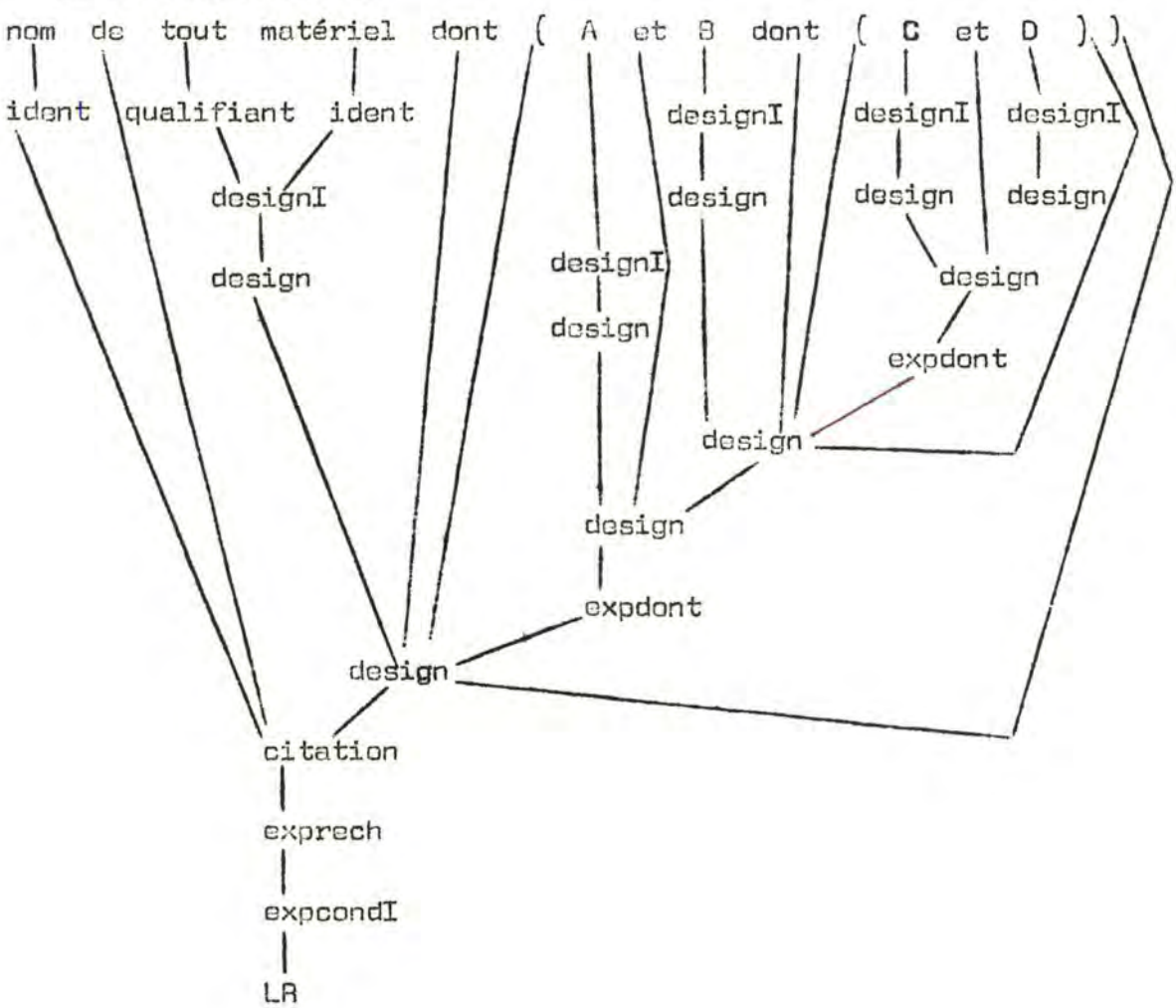
I) Nom de tout matériel dont (un département (code = IO) via I6 et une bibliothèque (nom = BI) via 2I dont (un emprunteur (nom = Jean) via 2 et un gestionnaire (code = 5) via 6)) ?

Convenons d'appeler:

un département (code = IO) via I6 par A
une bibliothèque (nom = BI) via 2I par B
un emprunteur (nom = Jean) via 2 par C
un gestionnaire (code = 5) via 6 par D

A, B, C, D sont des symboles représentant designI

La requête devient:



- Table des caractéristiques virtuelles, qui nous donnera quelques renseignements relatifs aux caractéristiques virtuelles, c'est-à-dire qui ne sont pas reprises dans la data base sous la forme que l'utilisateur défini.

Age	date-naiss	date-jour - date-naiss

- Table des fonctions qui contiendra les différentes fonctions reconnues par le système, ainsi que des renseignements relatifs aux éventuels paramètres.

I	MOYENNE	
2	CERTIFICAT-EMPRUNT	rensei-
3	GRAPHIQUE	gnements
4	TABLEAU	relatifs
5	SOMME	aux
6	DENOMBREMENT	paramètres

- Table des synonymes fonctionnels.

TOTAL	5
MOYENNE DE	I
NOMBRE MOYEN	I

Le numéro apparaissant en 2° colonne est un renvoi à la table des fonctions.

Si on désire enrichir le vocabulaire LN employé pour nommer les relations, on pourra ajouter une table des synonymes relationnels. Cela permettrait d'avoir une plus grande souplesse dans la formulation des requêtes LN.

5.2. Correspondance LN - LR .

Etablissons maintenant une correspondance entre les deux langages qui nous intéressent ici.

L N	L R
Commande simple Quel est	Rechercher, fonctions
Etablir	Rechercher plus éventuellement un format de sortie.
Si	Si
Idem	Pas d'équivalent, puisque travaille au niveau du langage utilisé.
Vérifier	Vérifier
Effectuer	Fonction
Rechercher	Rechercher
Préciser	Pas d'équivalent, est traité de suite, car a la même forme dans chacun des langages.
Si..alors..sinon	Si..alors..sinon

Nous avons donc pour chaque commande de LN un correspondant dans LR. Il faut remarquer que LN n'est pas lié à l'application par ses mots de commande. Il l'est par son vocabulaire, c'est-à-dire par les noms de relations. Cela est normal et il nous semble très difficile d'envisager la création d'un langage naturel qui ne serait pas dédicacé à une application particulière.

Dans le cas où la requête ne fait intervenir qu'une entité et non une relation, c'est le nom de l'entité qui sera stocké dans TL.

ex. : Rechercher le nom de tous les emprunteurs.

TL

emprun-teur

Nous allons donner la représentation de cette table dans chacun des cas fondamentaux de formulation d'une requête LN, cas fondamentaux donnés en LR comme nous l'avons signalé auparavant.

I- A DE B DE C DE D

AB	DE
BC	DE
CD	DE

2- A DONT (B ET C ET D)

AB	DONT
AC	ET
AD	ET

3- A DONT (B OU C OU D)

AB	DONT
AC	OU
AD	OU

4- A DONT (B OU C ET D)

AB	DONT
AC	OU
AD	ET

5- A DONT (B ET C DE D)

AB	DONT
AC	ET
CD	DE

A, B, C, D représentent des entités.

AB est la relation, du moins son nom, qui unit B et A, c'est-à-dire B ⟵ A dans la "structure vue utilisateur".

5.4. Restrictions.

Nous avons exclu de cet exposé toute structure de parenthétisation qui pourrait intervenir dans LN. Ce serait le cas pour des requêtes qui donneraient en LR le résultat suivant :

A DONT (B ET C DONT (D ET E))

ou A DONT (B OU (C OU D) ET B)

Dans de tels cas, il serait nécessaire d'ajouter une colonne supplémentaire à TL, on en stockerait la portée des parenthèses.

Remarquons que dans les exemples cités ci-dessus, nous n'avons pas repris le numéro de la relation. Nous allons les reformuler en donnant ces numéros et les tables TL nécessaires pour en réaliser une interprétation correcte.

I- A DONT (B VIA I ET C VIA 2 DONT (D VIA 3 ET E VIA 4))

TL :

I		DONT
2		ET
3	2	DONT
4		ET

2- A DONT (B VIA I OU (C VIA 2 OU D VIA 3) ET E VIA 4)

TL :

I		DONT
2	2	OU
3		OU
4		ET

Le nombre situé en colonne deux donne la portée des parenthèses.

L'exemple deux intervient lorsque l'on veut modifier la priorité des opérateurs ET, OU.

L'exemple UN revient à dire que nous travaillons essentiellement sur la première entité d'une requête. Les autres entités qui interviennent ne seront pas filtrées par d'autres relations, mais uniquement par des caractéristiques.

Les modifications qu'il faudrait apporter pour admettre des requêtes telles que les deux exemples ci-dessus ne changent rien au principe de cette méthode d'analyse de LN. Il suffirait d'ajouter aux organigrammes décrits ci-après, une routine pour traiter les parenthèses.

Nous admettons ici que l'utilisateur construit sa question en tenant compte de la facilité qui lui est offerte par le schéma de la base. Il lui suffit de suivre le sens des flèches et au cas où cela serait impossible parce qu'il doit employer plusieurs relations non successives, il lui suffit de les relier par ET ou OU.

Ne sont pas prévues des requêtes donnant comme filtre plusieurs valeurs d'une même caractéristique, sauf s'il s'agit d'un ou.

Quels gestionnaires gèrent DI ou D2 ?

Cela sera accepté.

Quels gestionnaires gèrent les départements DI et D2? où DI et D2 sont les noms des départements concernés.

De telles phrases n'ont pas été prévues dans l'analyse.

Pour admettre une telle requête, il faudrait vérifier si les valeurs reliées par et définissent la même caractéristique, auquel cas on générerait les telles différemment de façon à rappeler chaque fois le nom de l'entité.

5.5. Aspect conversationnel du système.

Dans ce paragraphe,nous allons rassembler les quelques ambiguïtés que le conversationnel a permis de lever.

C'est essentiellement dans le passage LN - LR que nous avons besoin de ce mode.Nous y avons quatre modules.

ConversI: Connaissant une requête utilisateur,le système constate que le mot de commande est absent.Il demande alors à l'utilisateur si c'est bien une recherche qu'il faut lancer.En effet, ce mot de commande est accepté dans un tel cas.Si l'utilisateur répond "non",il lui est demandé de reformuler sa requête.

Convers2: Connaissant une relation et une valeur,le système va proposer à l'utilisateur l'entité concernée par la relation,et lui demander si c'est cette dernière qui est sous-entendue.Si l'utilisateur répond " oui ",le système lui donne alors les caractéristiques que comprend cette entité,en lui demandant de préciser laquelle possède la valeur donnée par la requête.

Convers3: Dans sa requête,l'utilisateur a donné le nom d'une entité et une valeur.Le système va alors lui donner la liste des caractéristiques qui composent cette entité et demander à l'utilisateur d'indiquer celle qui possède la valeur renseignée par la requête.

Convers4: L'utilisateur a formulé une requête dans laquelle il donne le nom de l'entité concernée par la requête.Le système lui donne alors les caractéristiques qui appartiennent à cette entité et lui demande de préciser lesquelles font partie de la demande.Le système les place dans TD et termine la liste par un astérisque.

5.6. Analyseur de requêtes LN.

Décrivons tout d'abord les différents modules de cet analyseur avant de présenter quelques organigrammes.

ANALN : Son rôle est d'aiguiller l'analyse vers un module ad hoc.Pour cela,il se base sur les premiers mots de la requête LN.Ces différents modules sont:

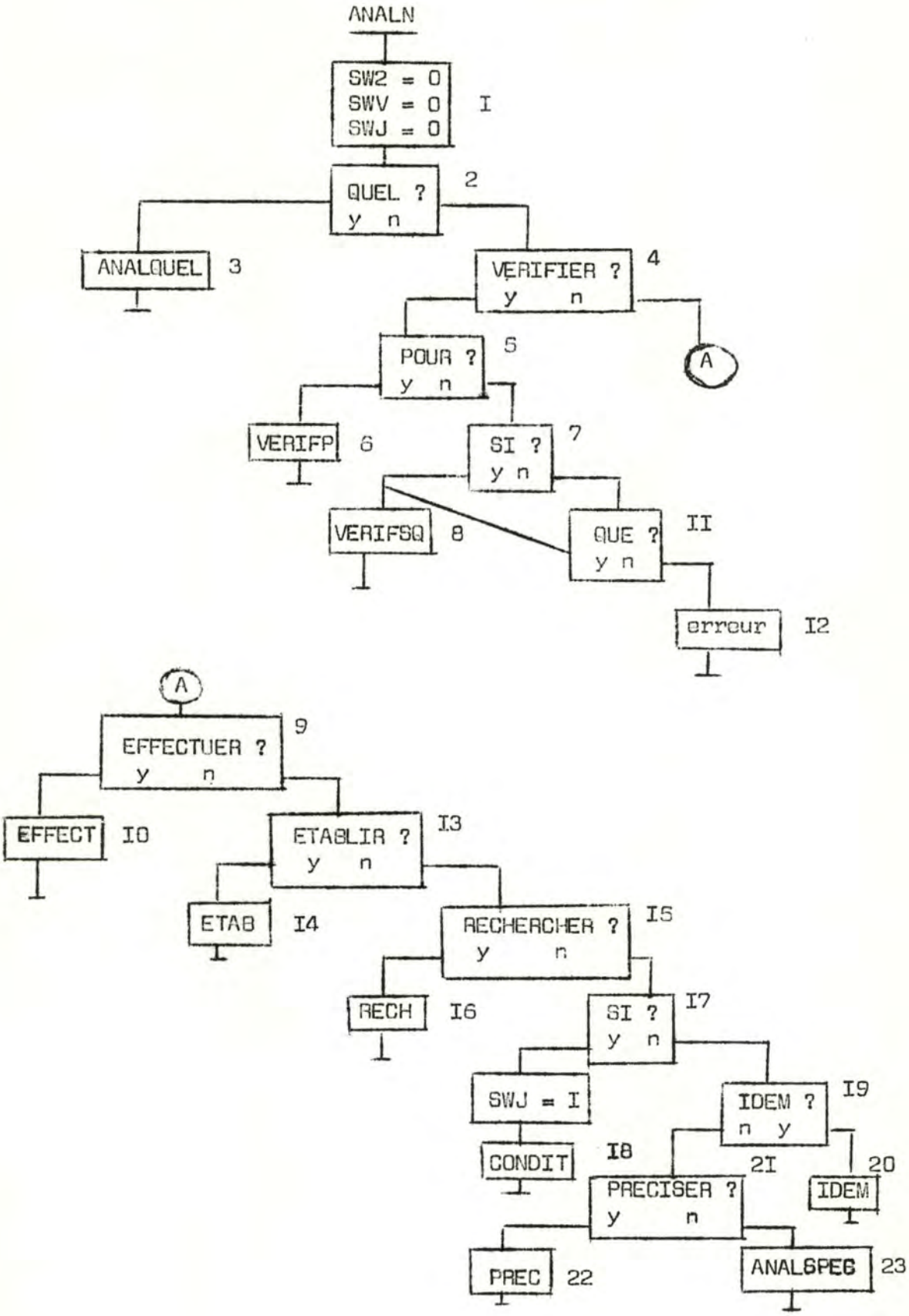
- ANALQUEL: qui traitera toute requête du genre QUEL EST ...
- VERIFP: qui s'occupe des requêtes à plusieurs vérifications,c'est-à-dire du genre VERIFIER POUR ...
- VERIFSQ: traite des vérifications de la forme VERIFIER SI... ou VERIFIER QUE...
- EFFECT: analyse toute requête qui commence par EFFECTUER.
- ETAB: traite des requêtes commençant par ETABLIR.
- RECH: analyse toute requête débutant par RECHERCHER.
- CONDIT: traite les requêtes conditionnelles.
- IDEM: exécute la commande IDEM,qui consiste à remplacer une chaîne de caractères par une autre.
- PREC: analyse la requête PRECISER qui permet à l'utilisateur de demander les caractéristiques composant une entité.
- ANALSPEC: traite les requêtes ne commençant pas par un mot de commande.Il permet de vérifier que la commande sous-entendue est bien RECHERCHER qui est la seule admise dans ce cas.

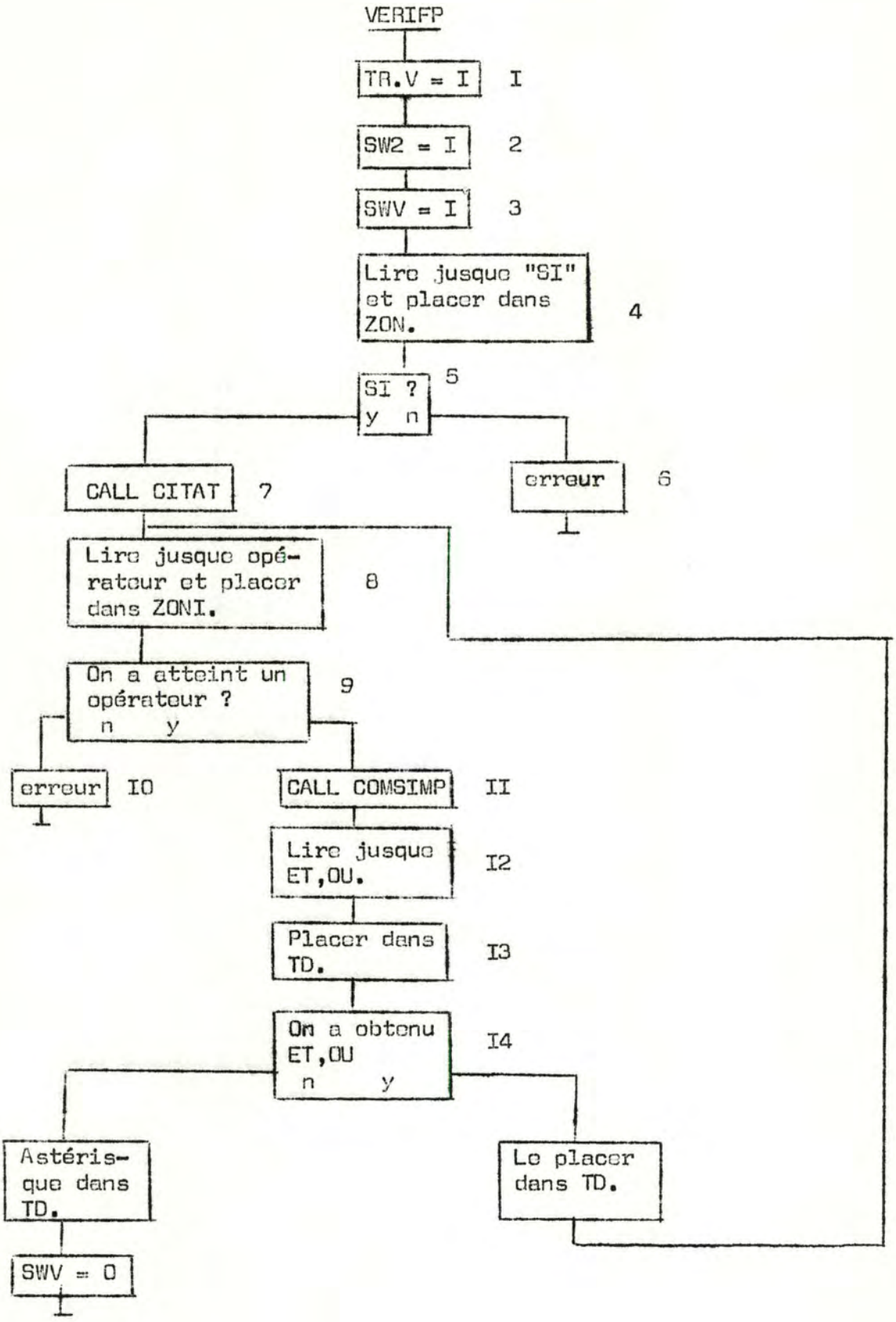
Pour analyser les requêtes LN,ces modules vont en appeler d'autres qui réaliseront le traitement de la requête.

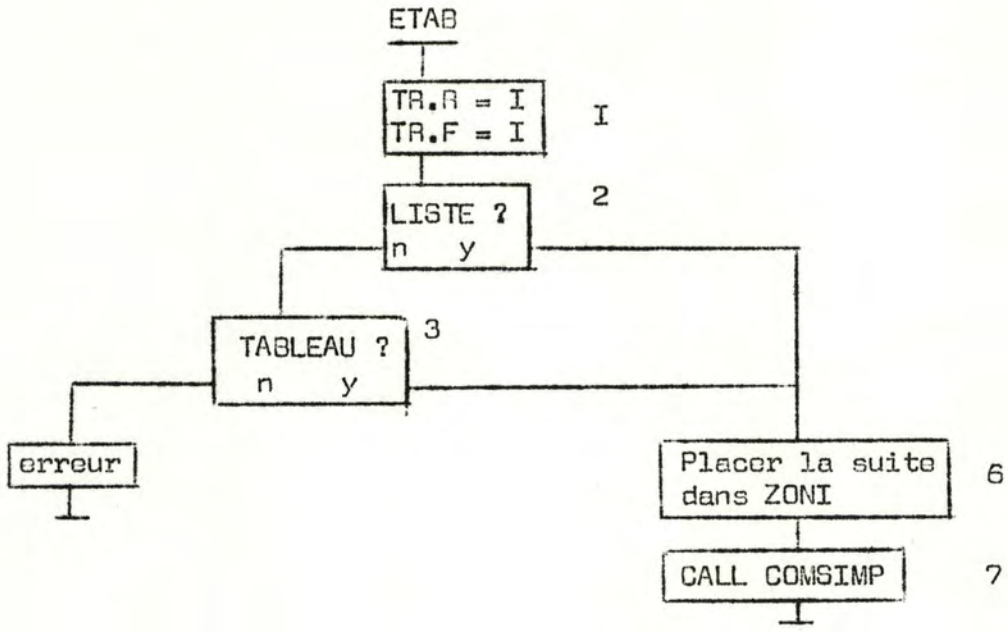
Il s'agit des modules suivants:

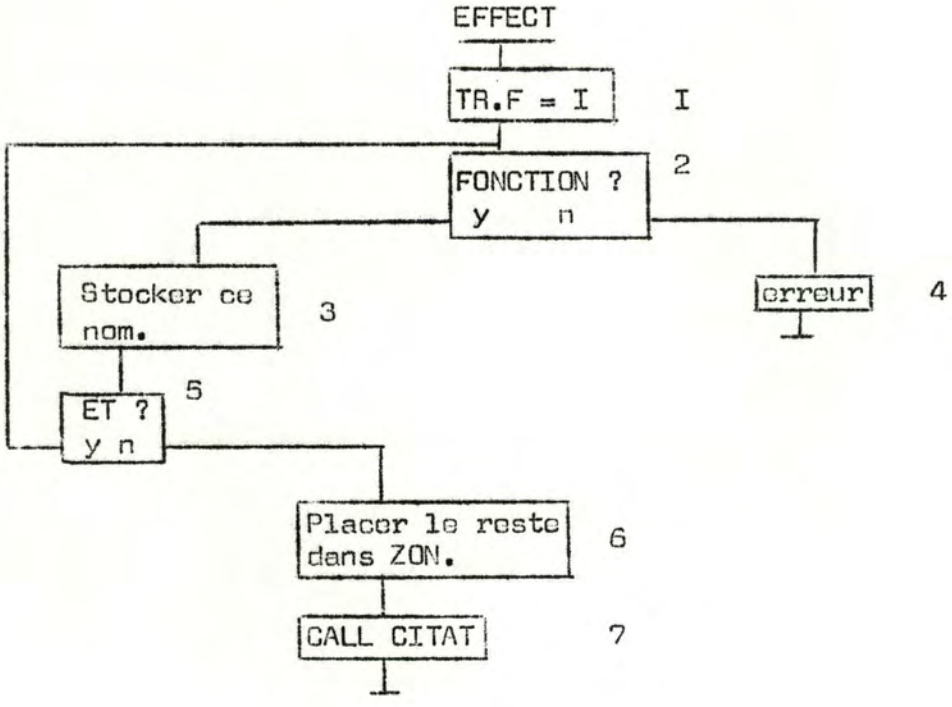
- COMSIMP: qui analyse ce que nous avons appelé commande simple.
- CITAT: est le module qui va analyser les citations telles qu'elles ont été définies dans la grammaire.
  Nous l'avons décomposé en plusieurs sous-modules:
  - CITATI: analyse les critères de sélection relatifs à la première entité d'une citation.
  - REL4: qui est chargé de garnir la table TL avec ET,DE,DONT.Il fait appel à deux autres modules qui analysent les relations proprement dites.Il s'agit de:
    - RELI: traite les relations du type nom de relation + nom d'entité + DE + item
    - REL2: traite les relations du genre nom relation + nom d'entité + valeur
      ou nom de relation + valeur.

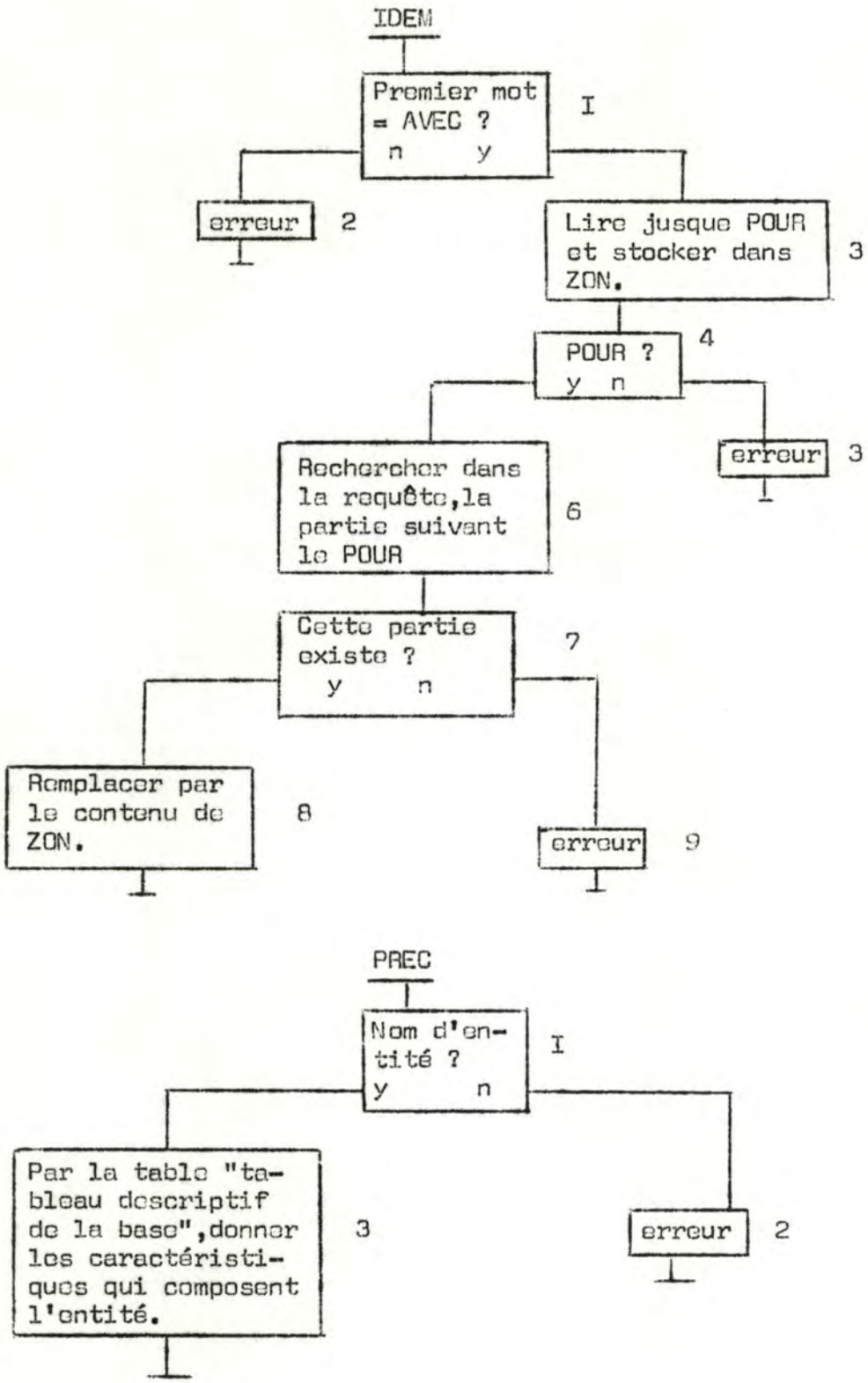
Le module CONDIT fait appel à COND qui se charge de transformer la requête pour qu'elle puisse être analysée par VERIFSQ.

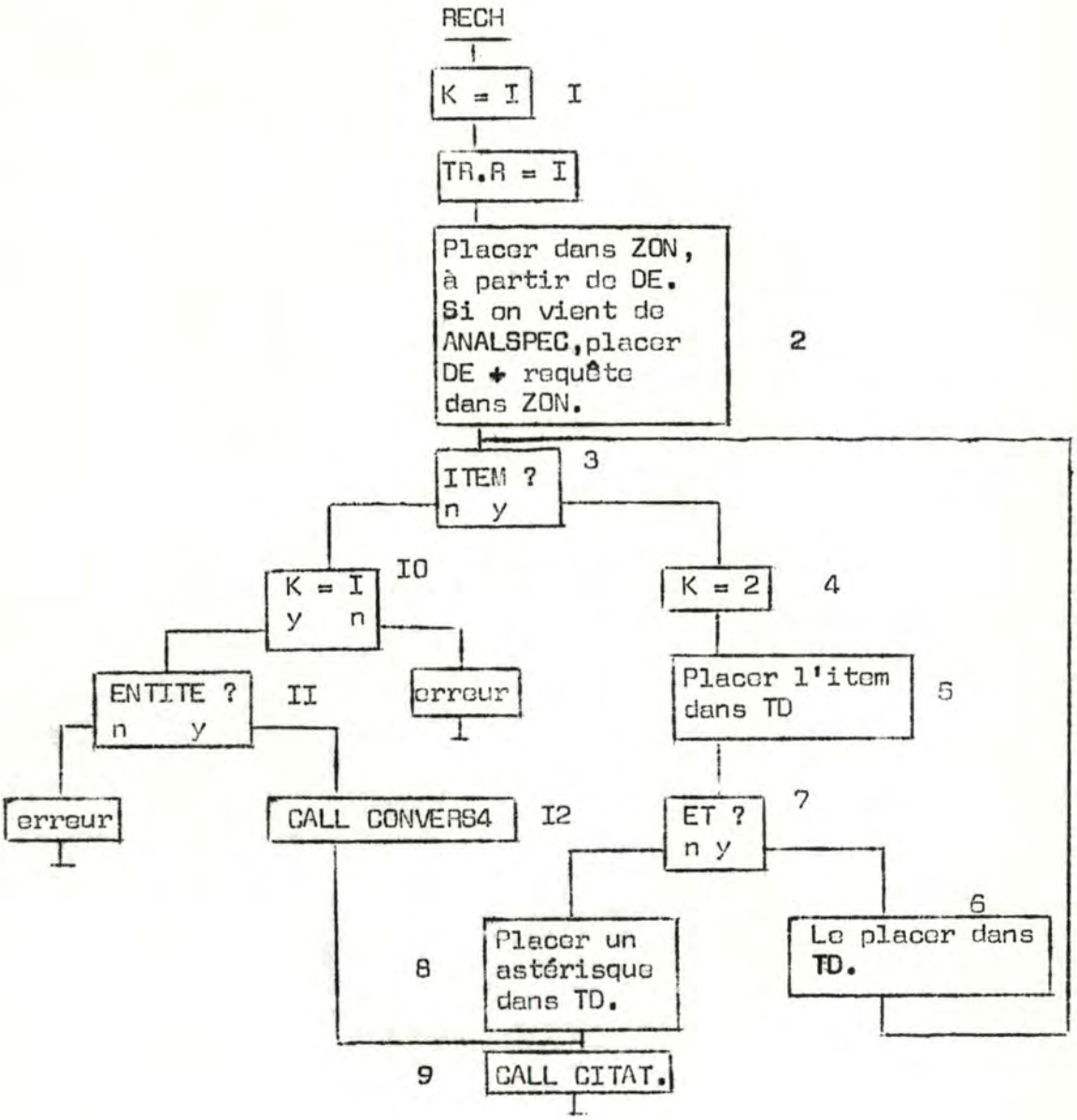

ANALN
SW2 = 0
SWV = 0
SWJ = 0
I
QUEL ?
y n
2
ANALQUEL
3
VERIFIER ?
y n
4
POUR ?
y n
5
A
VERIFP
6
SI ?
y n
7
VERIFSQ
8
QUE ?
y n
II
erreur
I2
A
EFFECTUER ?
y n
9
EFFECT
I0
ETABLIR ?
y n
I3
ETAB
I4
RECHERCHER ?
y n
I5
RECH
I6
SI ?
y n
I7
SWJ = I
CONDIT
I8
IDEM ?
n y
I9
2I
PRECISER ?
y n
20
IDEM
PREC
22
ANALSPEC
23

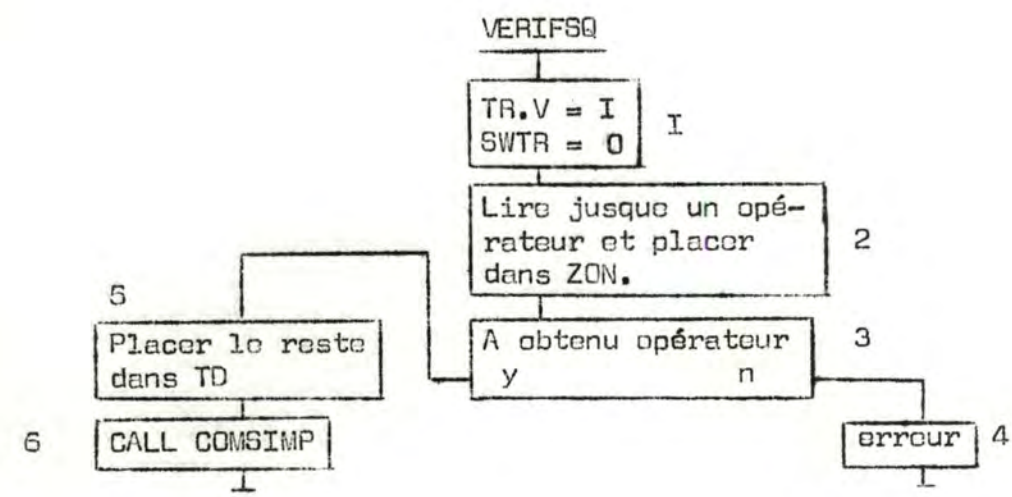

VERIFP
TR.V = I
I
SW2 = I
2
SWV = I
3
Lire jusque "SI" et placer dans ZON.
4
SI ?
y n
5
CALL CITAT
7
erreur
6
Lire jusque opérateur et placer dans ZONI.
8
On a atteint un opérateur ?
n y
9
erreur
IO
CALL COMSIMP
II
Lire jusque ET,OU.
I2
Placer dans TD.
I3
On a obtenu ET,OU
n y
I4
Astérisque dans TD.
SWV = 0
Le placer dans TD.

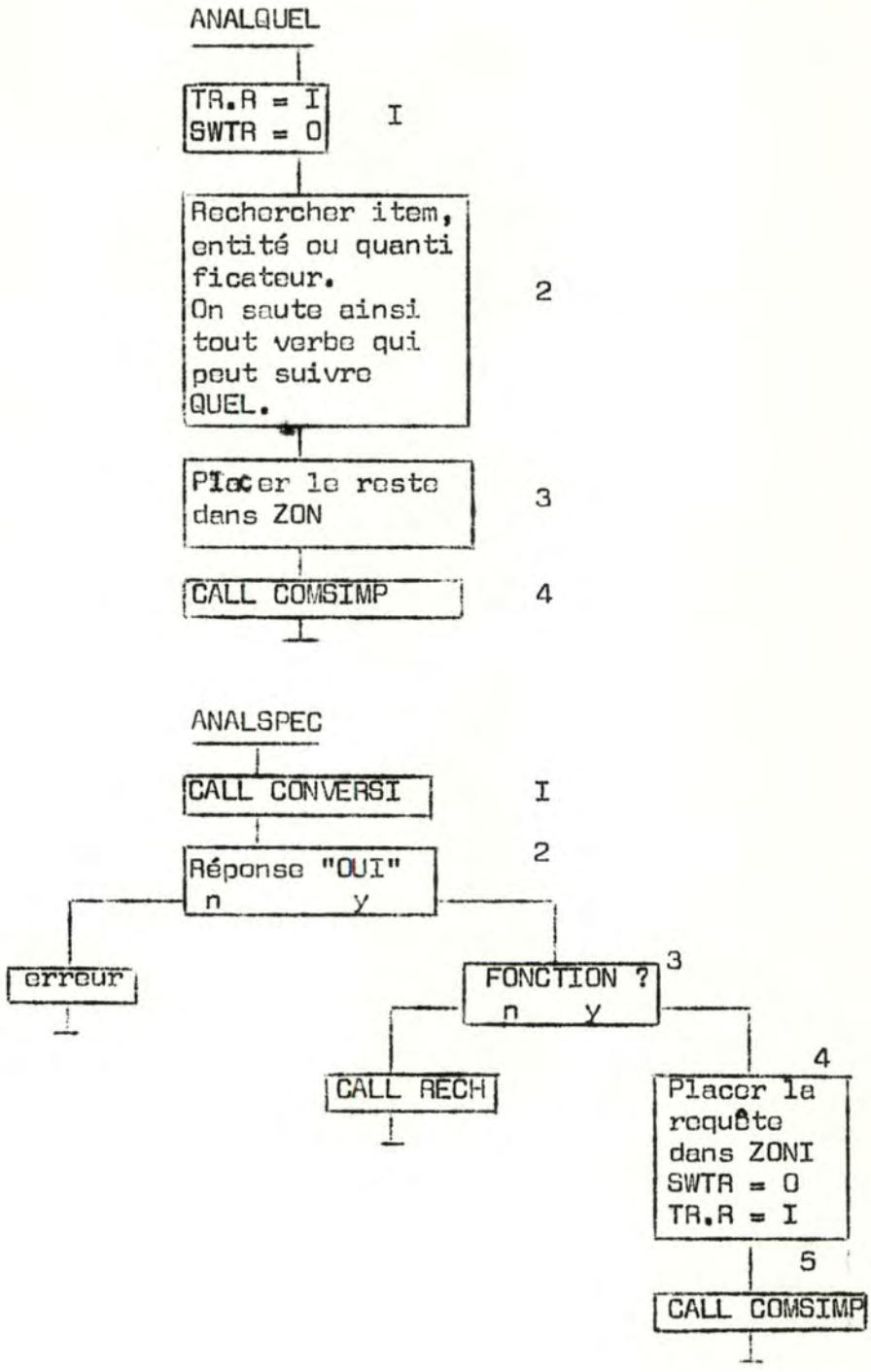

ETAB
TR.R = I
TR.F = I
I
LISTE ?
n y
2
TABLEAU ?
n y
3
erreur
Placer la suite
dans ZONI
6
CALL COMSIMP
7

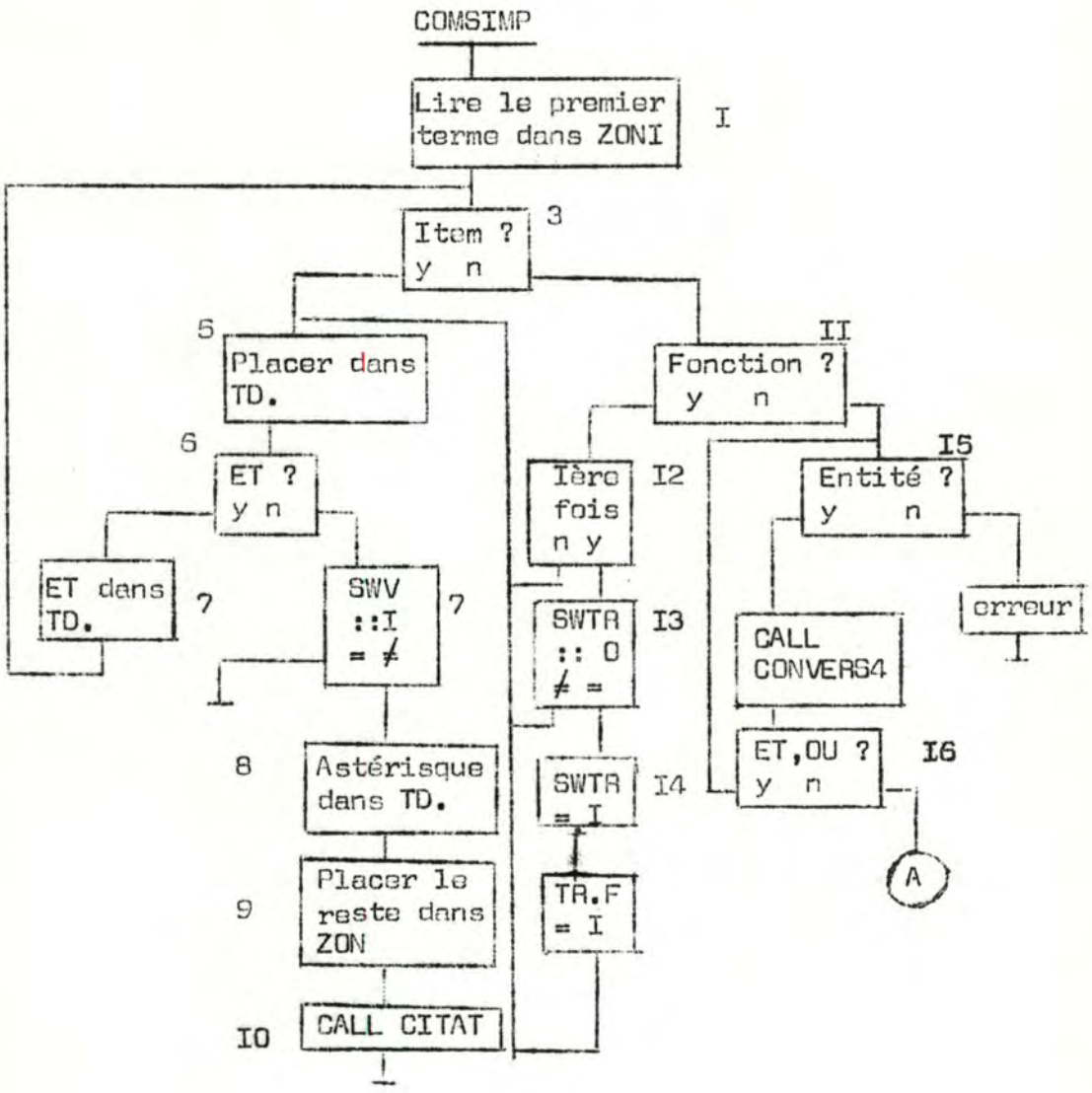

EFFECT
TR.F = I
I
FONCTION ?
y n
2
Stocker ce
nom.
3
erreur
4
ET ?
y n
5
Placer le reste
dans ZON.
6
CALL CITAT
7

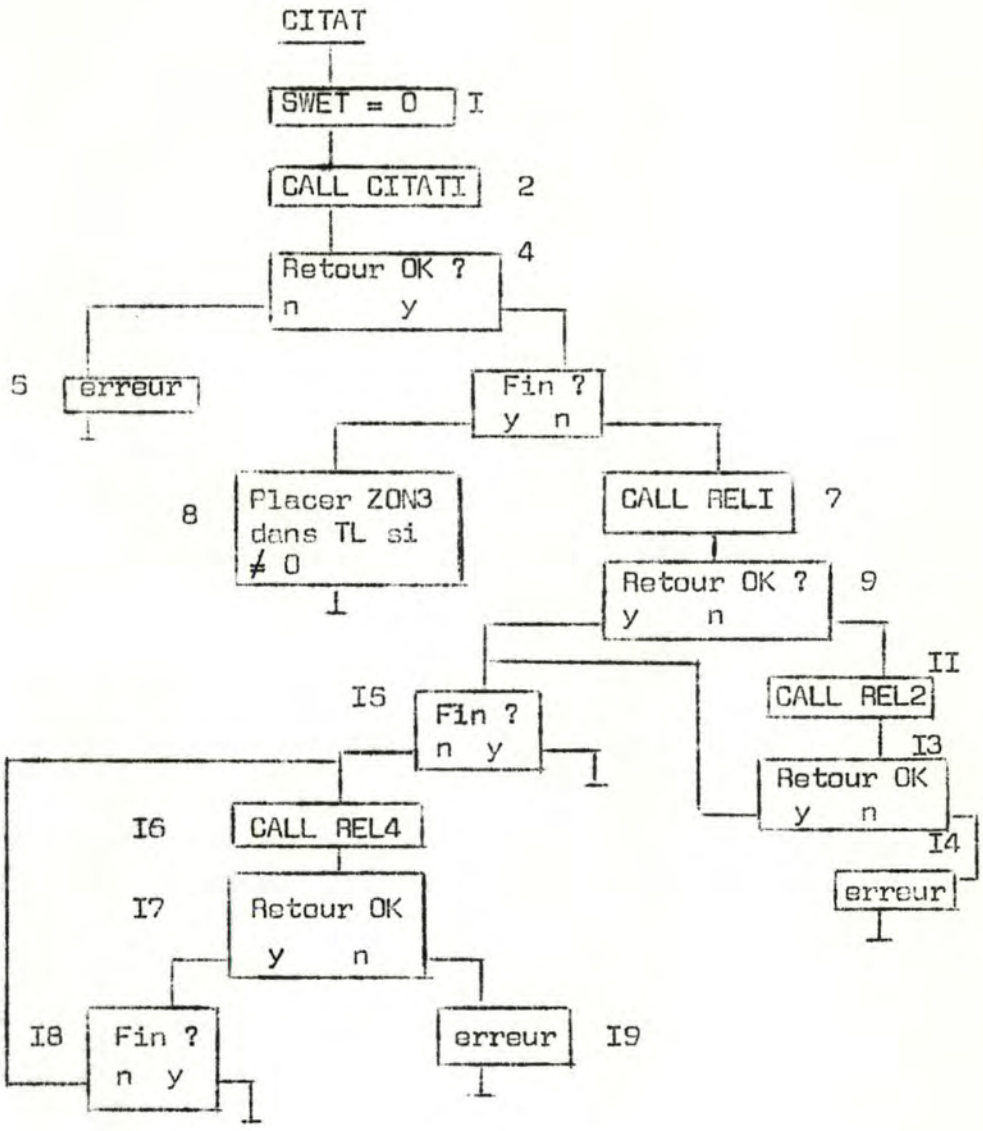

IDEM
Premier mot = AVEC ?
n y
I
erreur
2
Lire jusque POUR et stocker dans ZON.
3
POUR ?
y n
4
erreur
3
Rechercher dans la requête, la partie suivant le POUR
6
Cette partie existe ?
y n
7
Remplacer par le contenu de ZON.
8
erreur
9
PREC
Nom d'entité ?
y n
I
Par la table "tableau descriptif de la base", donner les caractéristiques qui composent l'entité.
3
erreur
2

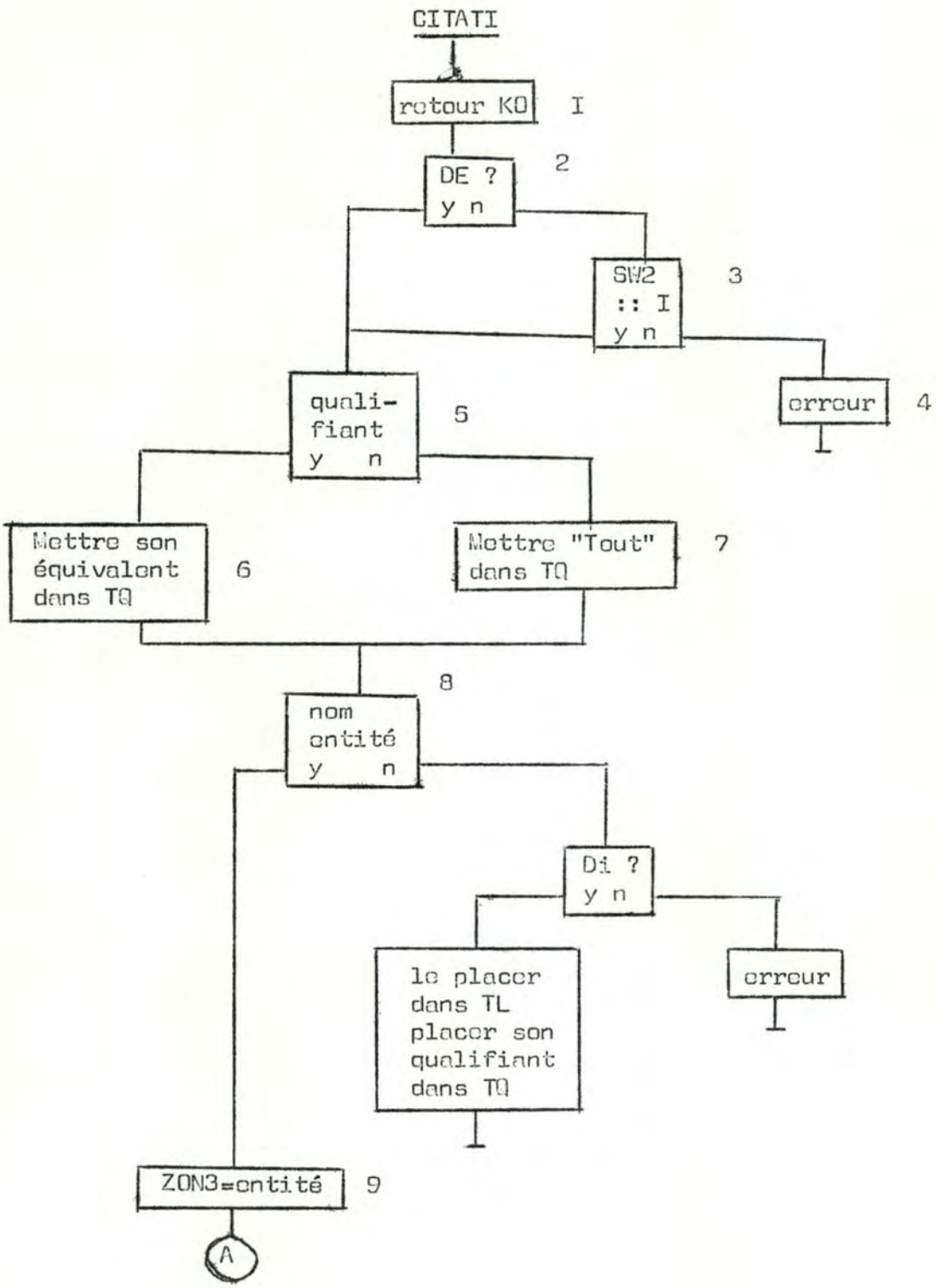

RECH
K = I
I
TR.R = I
Placer dans ZON, à partir de DE. Si on vient de ANALSPEC, placer DE + requête dans ZON.
2
ITEM ?
n y
3
K = I
y n
I0
K = 2
4
ENTITE ?
n y
II
erreur
Placer l'item dans TD
5
erreur
CALL CONVERS4
I2
ET ?
n y
7
6
Le placer dans TD.
8
Placer un astérisque dans TD.
9
CALL CITAT.

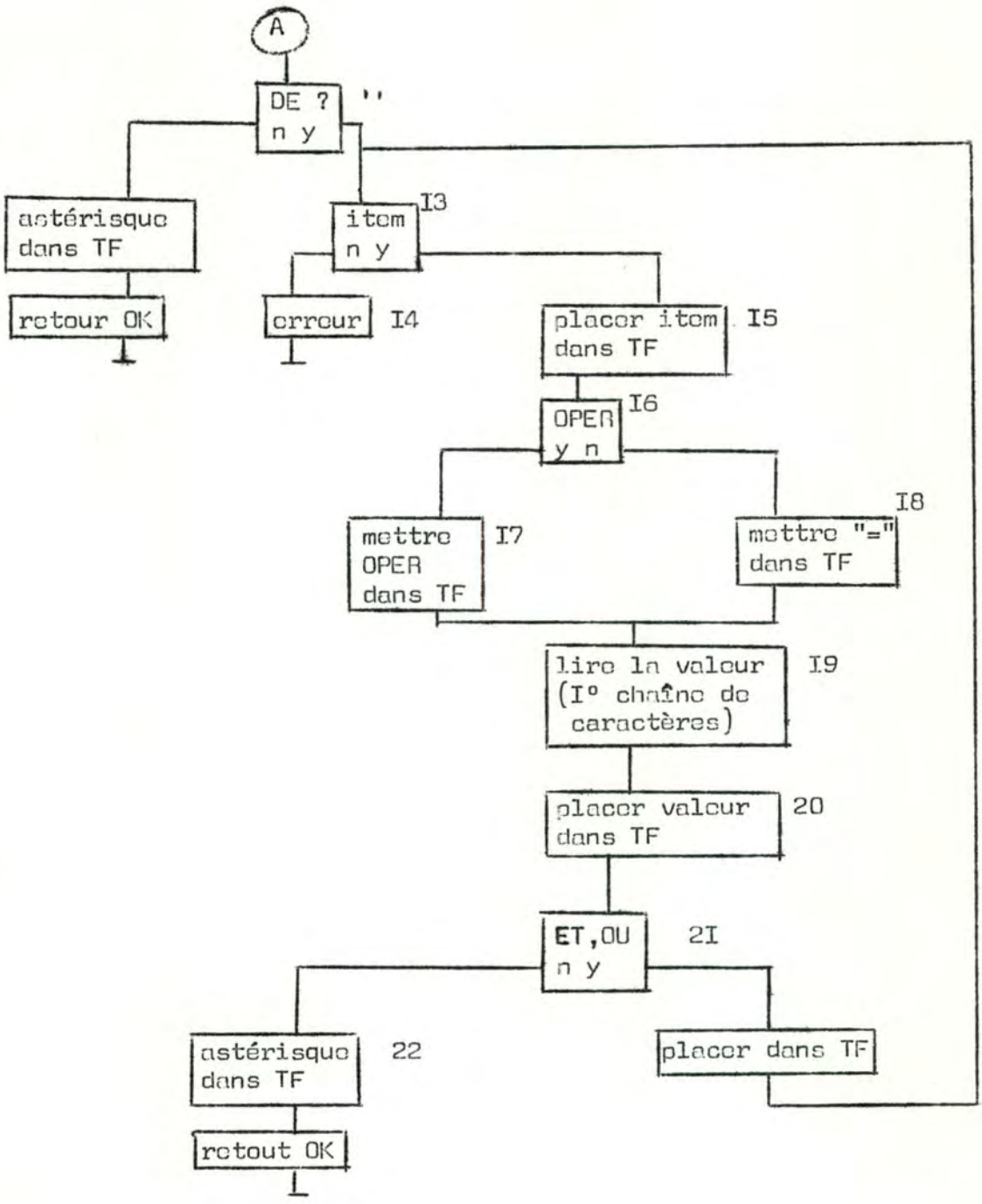

VERIFSQ
TR.V = I
SWTR = 0
I
Lire jusque un opérateur et placer dans ZON.
2
5
Placer le reste dans TD
A obtenu opérateur
y n
3
6
CALL COMSIMP
erreur
4

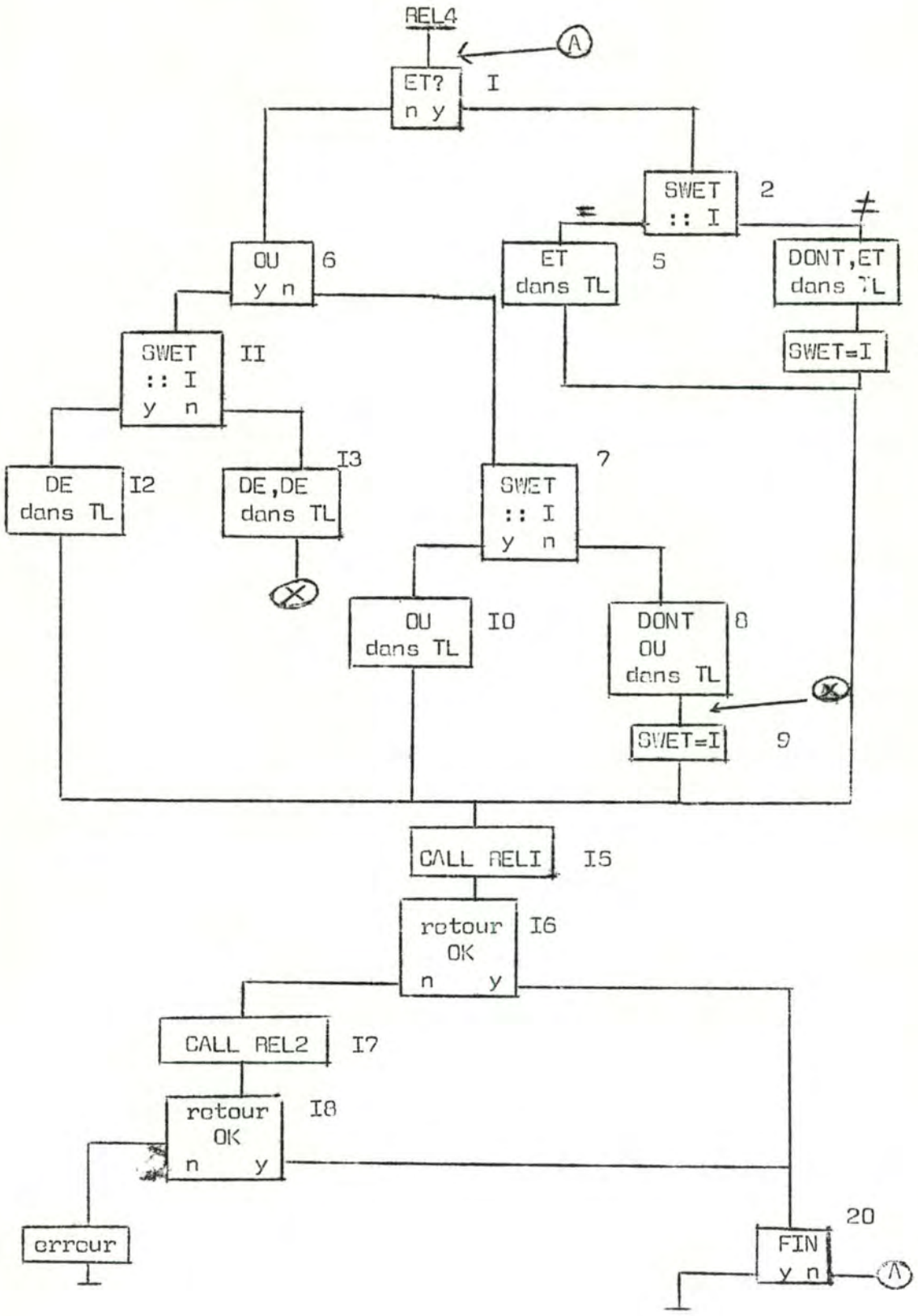

ANALQUEL
TR.R = I
SWTR = 0
I
Rechercher item, entité ou quantificateur. On saute ainsi tout verbe qui peut suivre QUEL.
2
Placer le reste dans ZON
3
CALL COMSIMP
4
ANALSPEC
CALL CONVERSI
I
Réponse "OUI"
n
y
2
erreur
FONCTION ?
n
y
3
CALL RECH
Placer la requête dans ZONI
SWTR = 0
TR.R = I
4
5
CALL COMSIMP

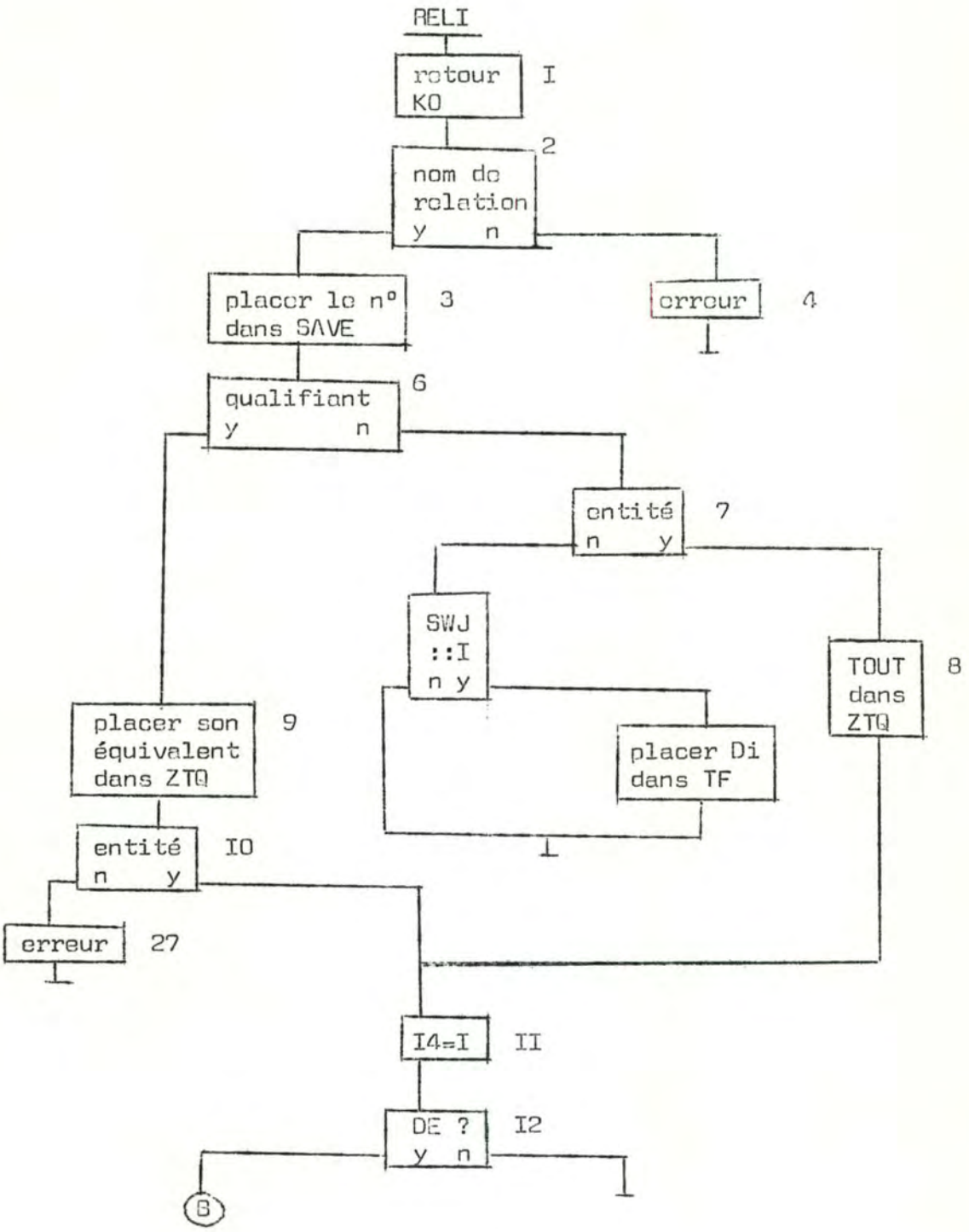

COMSIMP
Lire le premier terme dans ZONI
I
Item ?
y n
3
5
Placer dans TD.
6
ET ?
y n
ET dans TD.
7
SWV
::I
= ≠
7
8
Astérisque dans TD.
9
Placer le reste dans ZON
I0
CALL CITAT
II
Fonction ?
y n
Ièro fois
n y
I2
SWTR
:: 0
≠ =
I3
SWTR
= I
I4
TR.F
= I
I5
Entité ?
y n
erreur
CALL CONVERS4
ET,OU ?
y n
I6
A

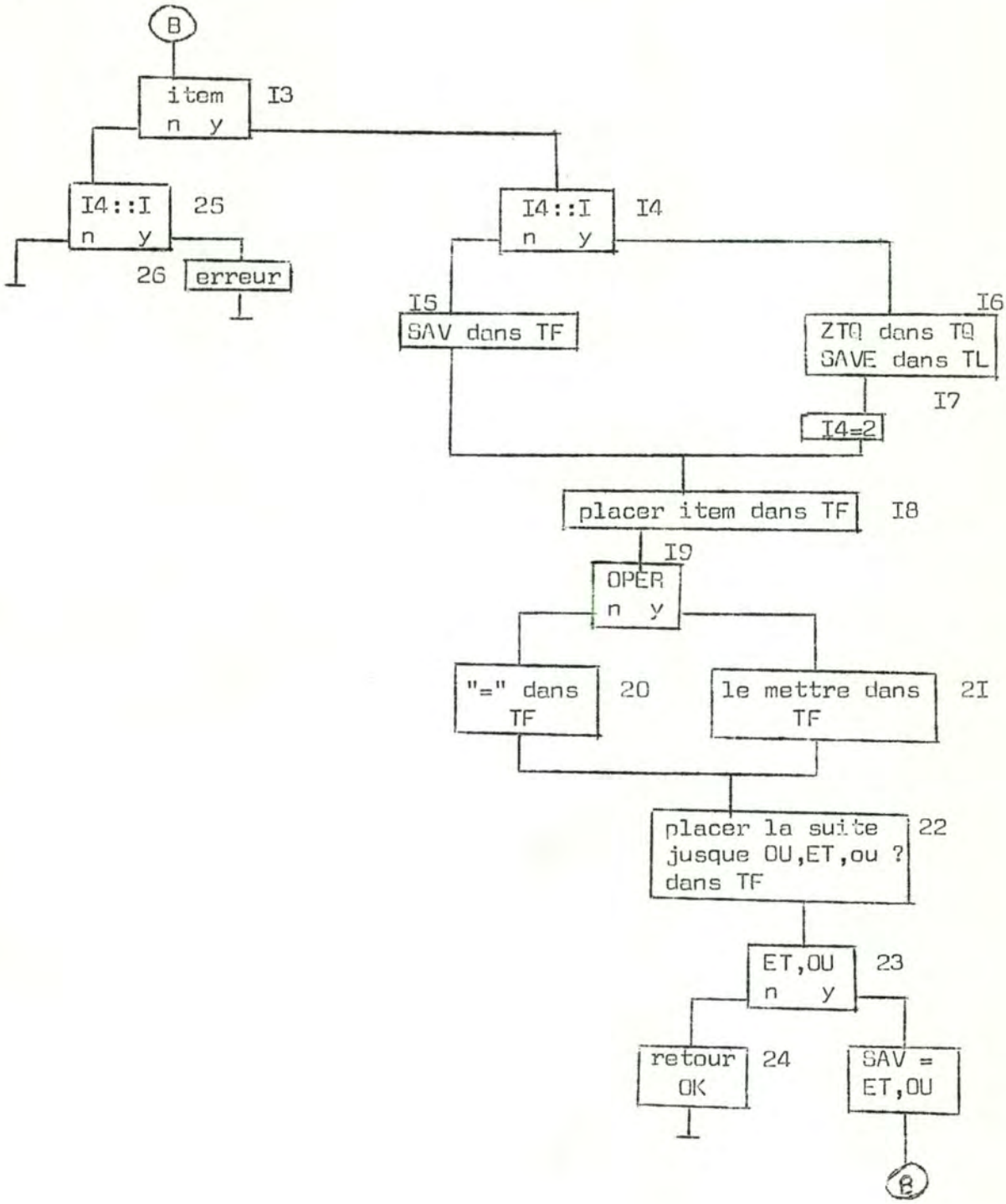

CITAT
SWET = 0
I
CALL CITATI
2
Retour OK ?
n y
4
5
erreur
Fin ?
y n
8
Placer ZON3
dans TL si
≠ 0
CALL RELI
7
Retour OK ?
y n
9
II
CALL REL2
I3
Retour OK
y n
I4
erreur
I5
Fin ?
n y
I6
CALL REL4
I7
Retour OK
y n
I8
Fin ?
n y
erreur
I9

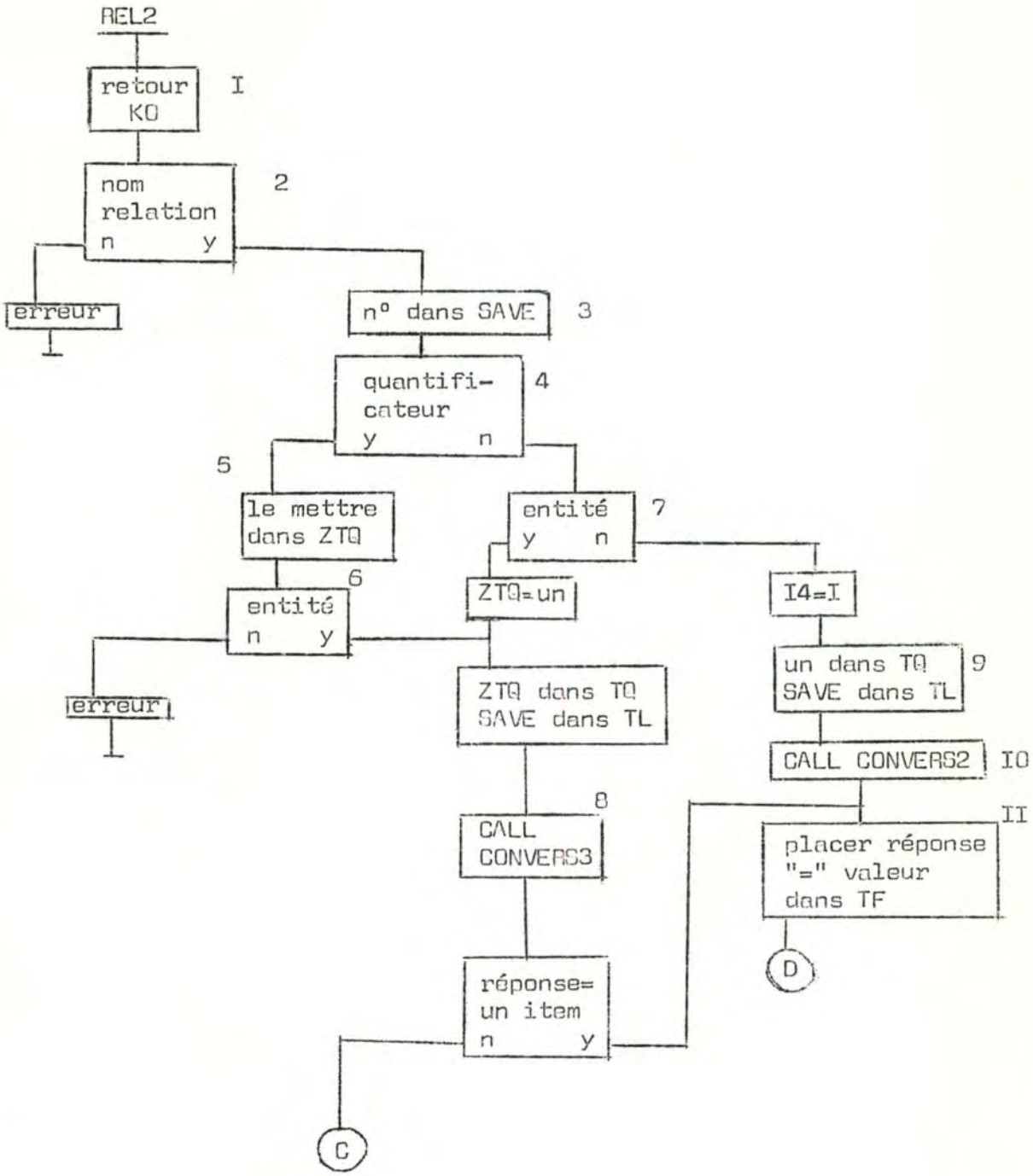

CITATI
retour KO
I
DE ?
y n
2
SW2
:: I
y n
3
erreur
4
qualifiant
y n
5
Mettre son équivalent dans TQ
6
Mettre "Tout" dans TQ
7
nom entité
y n
8
Di ?
y n
le placer dans TL placer son qualifiant dans TQ
erreur
ZON3=entité
9
A

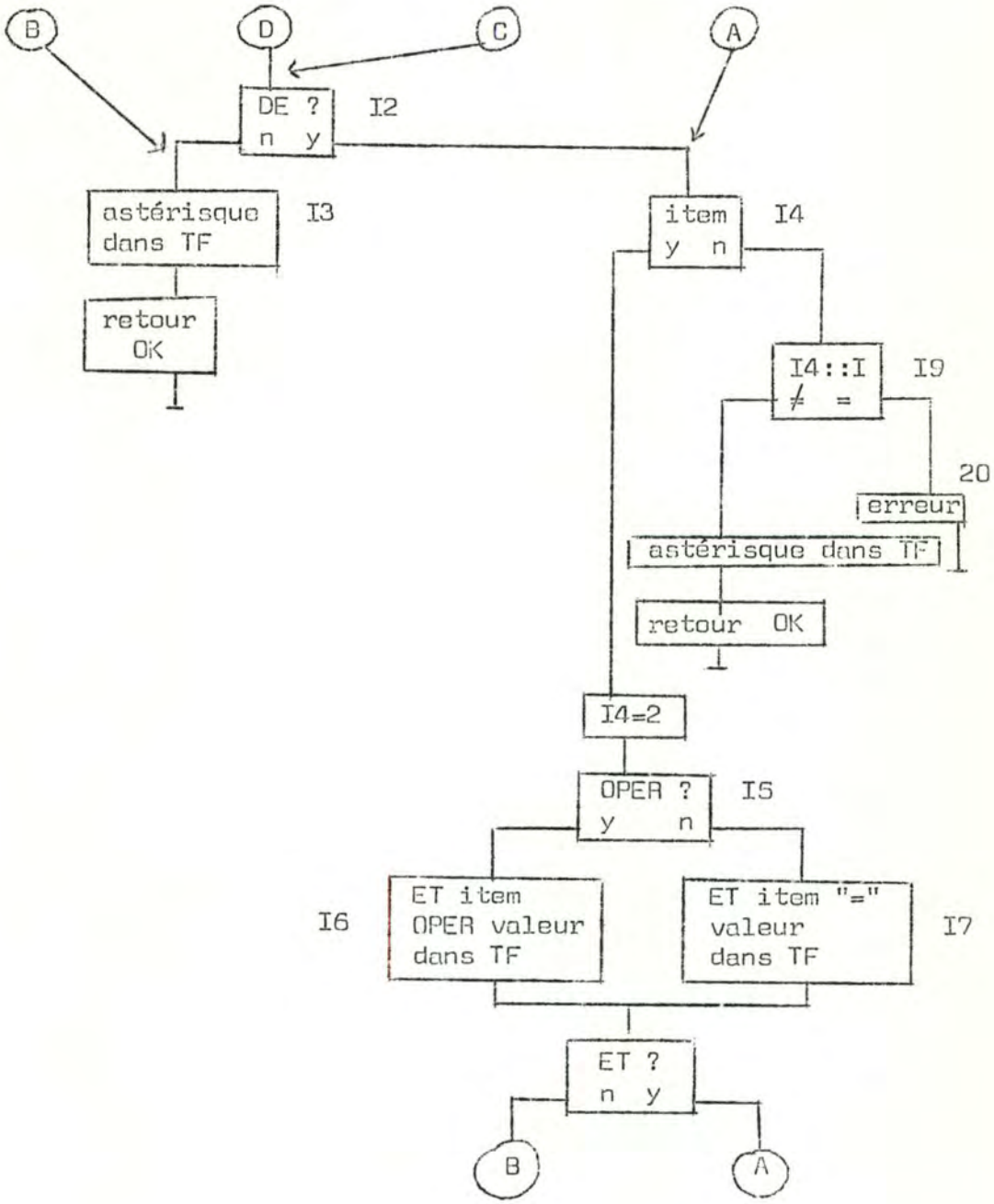

A
DE ?
n y
''
astérisque
dans TF
retour OK
item
n y
I3
erreur
I4
placer item
dans TF
I5
OPER
y n
I6
mettre
OPER
dans TF
I7
mettre "="
dans TF
I8
lire la valeur
(I° chaîne de
caractères)
I9
placer valeur
dans TF
20
ET,OU
n y
2I
astérisque
dans TF
22
retout OK
placer dans TF

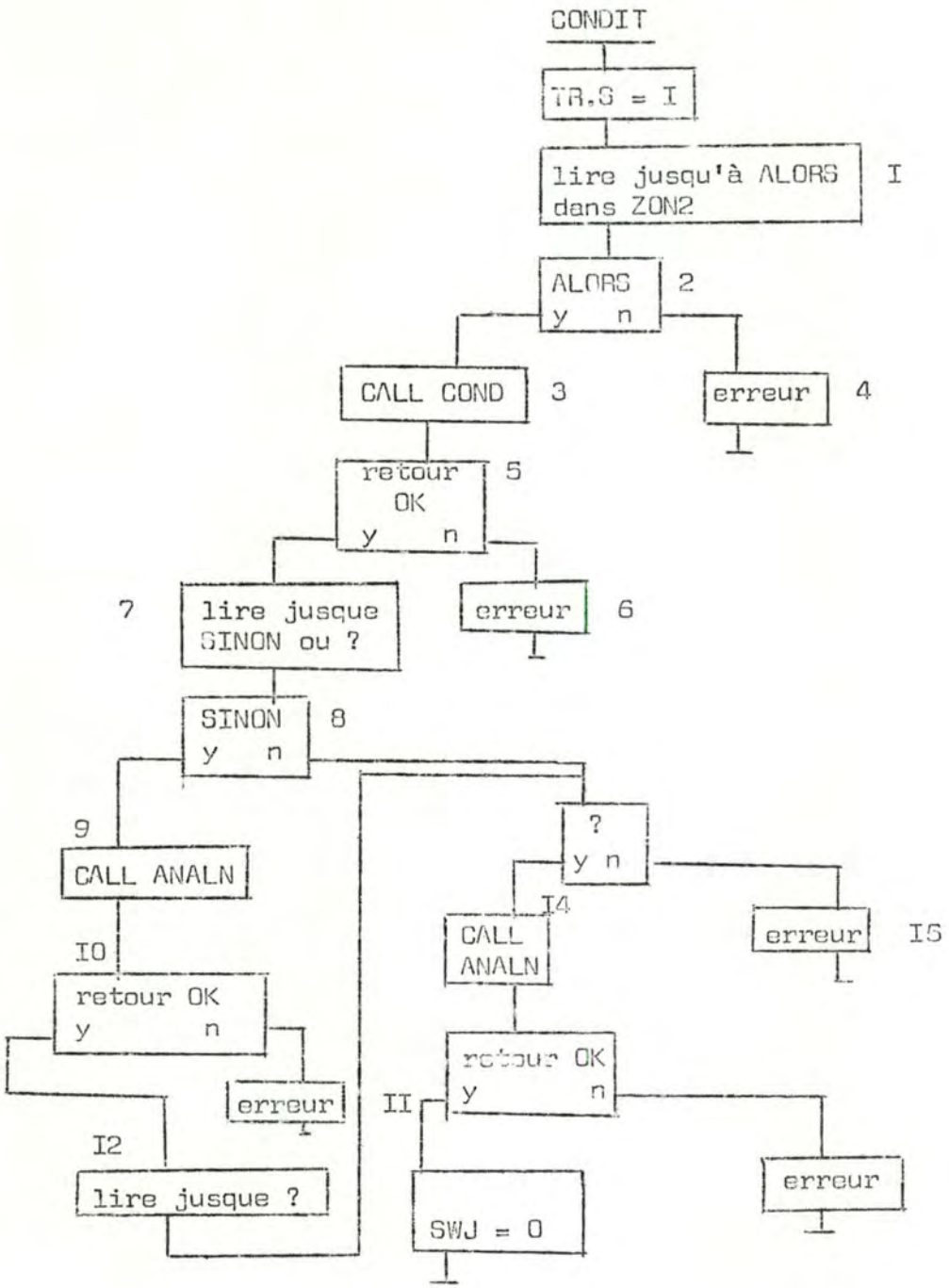

REL4
A
ET?
n y
I
SWET
:: I
2
=
≠
OU
y n
6
ET
dans TL
5
DONT,ET
dans TL
3
SWET
:: I
y n
II
SWET=I
4
DE
dans TL
I2
DE,DE
dans TL
I3
SWET
:: I
y n
7
OU
dans TL
I0
DONT
OU
dans TL
8
SWET=I
9
CALL RELI
I5
retour
OK
n y
I6
CALL REL2
I7
retour
OK
n y
I8
erreur
FIN
y n
20
A

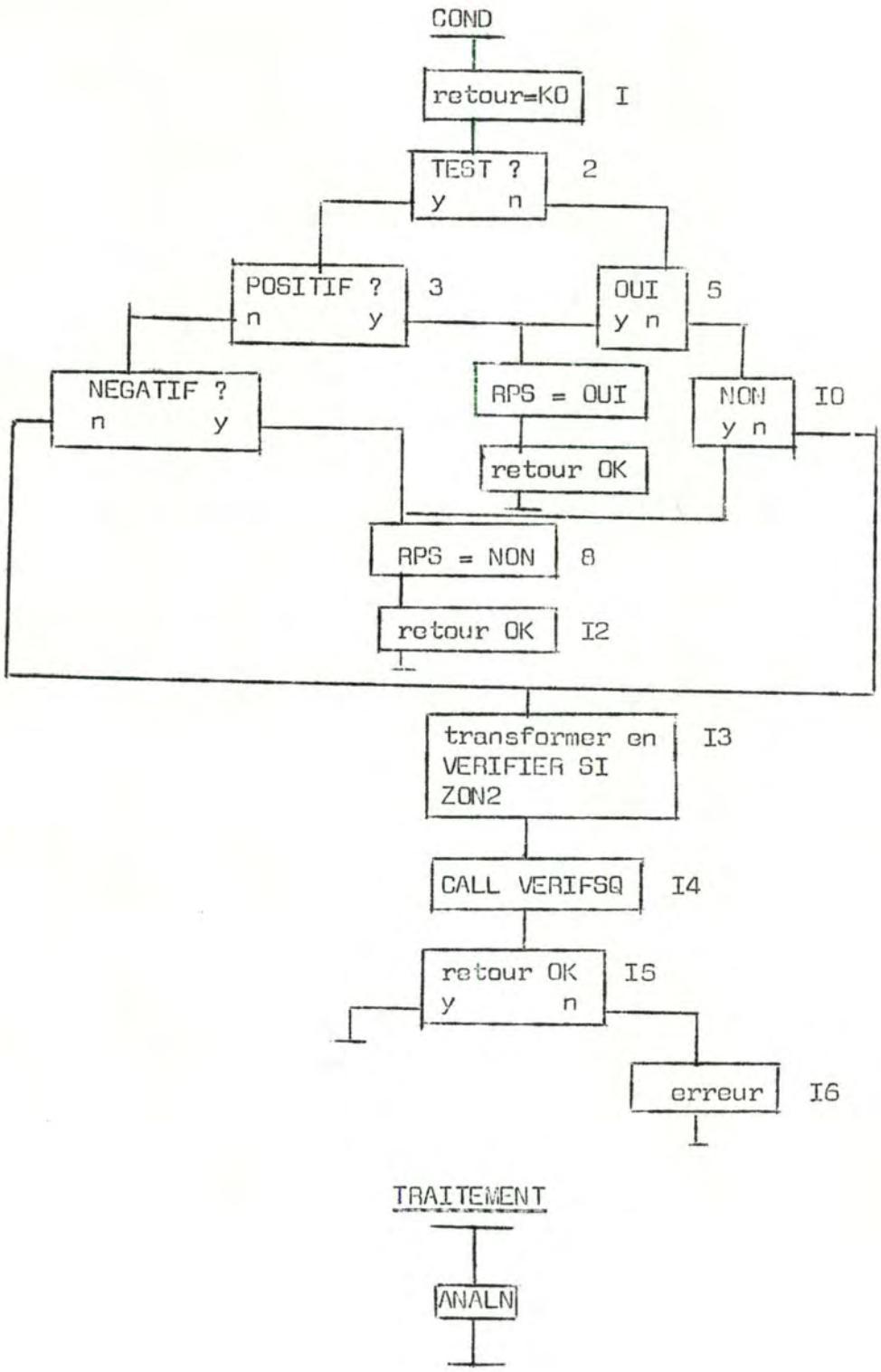

RELI
retour KO
I
2
nom de relation
y n
placer le n° dans SAVE
3
erreur
4
qualifiant
y n
6
entité
n y
7
SWJ ::I
n y
TOUT dans ZTQ
8
placer Di dans TF
placer son équivalent dans ZTQ
9
entité
n y
IO
erreur
27
I4=I
II
DE ?
y n
I2
B


B
item
n y
I3
I4::I
n y
25
26
erreur
I4::I
n y
I4
I5
SAV dans TF
I6
ZTQ dans TQ
SAVE dans TL
I7
I4=2
placer item dans TF
I8
I9
OPER
n y
"=" dans
TF
20
le mettre dans
TF
21
placer la suite
jusque OU,ET,ou ?
dans TF
22
ET,OU
n y
23
retour
OK
24
SAV =
ET,OU
R


REL2
retour KO
I
nom relation
n y
2
erreur
n° dans SAVE
3
quantifi-cateur
y n
4
5
le mettre dans ZTQ
entité
y n
7
6
entité
n y
ZTQ=un
I4=I
erreur
un dans TQ SAVE dans TL
9
ZTQ dans TQ SAVE dans TL
CALL CONVERS2
IO
8
CALL CONVERS3
II
placer réponse "=" valeur dans TF
D
réponse= un item
n y
C


B
D
C
A
DE ?
n y
I2
astérisque
dans TF
I3
retour
OK
item
y n
I4
I4::I
≠ =
I9
20
erreur
astérisque dans TF
retour OK
I4=2
OPER ?
y n
I5
I6
ET item
OPER valeur
dans TF
ET item "="
valeur
dans TF
I7
ET ?
n y
B
A


CONDIT
TR.S = I
lire jusqu'à ALORS dans ZON2
I
ALORS
y n
2
CALL COND
3
erreur
4
retour OK
y n
5
7
lire jusque SINON ou ?
erreur
6
SINON
y n
8
9
CALL ANALN
?
y n
I4
CALL ANALN
erreur
I5
I0
retour OK
y n
erreur
II
retour OK
y n
I2
lire jusque ?
SWJ = 0
erreur


COND
retour=KO
I
TEST ?
y n
2
POSITIF ?
n y
3
OUI
y n
5
NEGATIF ?
n y
RPS = OUI
NON
y n
IO
retour OK
RPS = NON
8
retour OK
I2
transformer en
VERIFIER SI
ZON2
I3
CALL VERIFSQ
I4
retour OK
y n
I5
erreur
I6
TRAITEMENT
ANALN

Quelques remarques sont à formuler relativement à ces organigrammes. - aucun switch ne comporte la lettre i comme symbole,sauf si elle est associée à un digit.De même,le chiffre un n'apparaît jamais dans une configuration symbolique.Cela provient des caractères machine à écrire.Une affectation telle que I4 = I signifie que le switch " i4 " prend la valeur " un ".Des relations de ce genre reviennent assez souvent dans les organigrammes.

- SWJ nous permet de reconstruire les tables de travail lorsque l'on vient de " SI ".Il est supposé testé au début de chacun des modules suivants : ETAB,EFFECT,RECH,VERIFQS.

- SW2,SWV indiquent que nous sommes occupés à analyser une requête du genre VERIFIER POUR .

- TR.R désigne la zone R de la table TR.Il en va de même pour V,F,S.

5.? Exemples.

I) Vérifier si le nom de l'auteur de cote 4 = Jean ?

```
ANALN
  I SW2 = 0      SWV = 0      SWJ = 0
  2 NON
  4 OUI
  5 NON
  7 OUI
   → VERIFSQ
       I  SWTR = I          | R | 0 |
                            | V | I |
                            | F | 0 |
                            | S | 0 |
       2 ZON I = " NOM DE L'AUTEUR DE COTE 4 "
       3 OUI
       5 TD
         | = Jean |
       6 → COM SIMP
             I NON
             3 OUI
             5      TD
                | NOM = JEAN |
             6 NON
             7 ≠
             8 * dans TD   | NOM = JEAN * |
             9 ZON = " DE L'AUTEUR DE COTE 4 "
            IO → CITAT
                  I  SWET = 0
                  2→ CITAT I
                       2  OUI
                       5  OUI
                       6  TD
                          | UN |
                       8 OUI
                       9 ZON 3 = "AUTEUR"
                      II OUI
                      I3 OUI
                      I5   TF
                           |
                      20 | COTE = 4 |
```

```
2I NON
22  TF
  | COTE = 4 * |

    retour OK à CITAT

4  OUI
6  OUI
8  TL
  | AUTEURS |

   fin
```

TR	TD	TF	TQ	TL
R O V I F O S O	NOM = JEAN*	COTE = 4*	UN	AUTEURS

En LR, nous avons

VERIFIER NOM = JEAN DE UN AUTEURS(COTE = 4) ?

2) Nom et code de l'auteur ayant écrit X emprunté par Jean ?

ANALN

I. SW2 = 0 SWV = 0 SWJ = 0
2. NON
4. NON
9. NON
I3. NON
I5. NON
I7. NON
I9. NON
2I. NON

ANALSPEC
→ Rechercher
→ Rech

I K = I

TR

R	I
V	0
F	0
S	0

3 OUI
4 K = 2
5 "NOM" dans TD
7 OUI
6 " ET" dans TD
3 OUI
4 K = 2
5 " CODE " dans TD
7 NON
8 "* " dans TD
9 → CITAT

TD

NOM	ET
CODE	*

I SWET = 0
2 CITAT I

2 OUI
5 OUI
6 TQ

UN

8 OUI
9 ZON3 = " AUTEUR"
II NON

TF

*

retour OK à CITAT

4 OUI
6 NON

```
7 --> RELI
        2 OUI
        3 SAVE = 9
        6 NON
        7 NON
          retour à CITAT

 9 NON
II --> REL2
        2 OUI
        3 SAVE = 9
        4 NON
        7 NON
          I 4 = I
        9  TQ        TL
          +----+    +---+
          | UN |    | 9 |
          | UN |    +---+
          +----+
       I0 résultat : TITRE
       II   TF
          +-------------+
          |      *      |
          | TITRE ← X   |
          +-------------+
       I2 NON
       I3   TF
          +---------------+
          |      *        |
          | TITRE = X *   |
          +---------------+
          retour OK CITAT

 I3 OUI
 I5 NON
 I6 --> REL4
         I NON
         6 NON
        II ≠
        I3    TL
             +-------+
             | 9  DE |
             |    DE |
             +-------+
        I4 SWET = I
        I5 --> RELI
                 2 OUI
                 3 SAVE = I3
                 6 NON
                 7 NON
                   retour KO à REL4

        I6 NON
```

I7 → REL2
2 OUI
3 SAVE = I3
4 NON
7 NON
I 4 = I
9 TQ TL

TQ
UN
UN
UN

TL	
9	DE
I3	DE

I0 résultat : NON
II TF

TF
TITRE = X *
NOM = JEAN

I2 NON
I3 TF

TF
TITRE = X *
NOM = JEAN *

retour OK à REL4

I8 OUI
20 OUI

TR

R	I
V	0
F	0
S	0

TD

NOM ET
CODE *

TF

*
TITRE = X
NOM = JEAN *

TL

9	DE
I3	DE

TQ

UN
UN
UN

Nous avons en LR

RECHERCHER NOM ET CODE DE UN AUTEUR$DE UN OUVRAGE (TITRE = X)
VIA 9 DE UN EMPRUNTEUR (NOM = JEAN) VIA I3 ?

3) Rechercher titre de l'ouvrage de présence 0 acquis par BI de commune = Namur et province = Namur et écrit par l'auteur Jean?

ANALN

```
 I SW2 = 0        SWV = 0        SWJ = 0
 2 NON
 4 NON
 9 NON
I3 NON
I5 OUI
I6 --> RECH
        I K = I
        2  TR
       +---+---+
       | R | I |
       | V | 0 |
       | F | 0 |
       | S | 0 |
       +---+---+
        3 OUI
        4 K = 2
        5  TD
       +-------+
       | TITRE |
       +-------+
        7 NON
        8  TD
       +--------+
       | TITRE* |
       +--------+
        9 --> CITAT
                I SWET = 0
                2   CITAT I
                     2 OUI
                     3 NON
                     5 OUI
                     6 TQ
                      +----+
                      | UN |
                      +----+
                     8 OUI
                     9 ZON3 = " OUVRAGE "
                    II OUI
                    I3 OUI
                    I5  TF
                    +-------------+
                    | PRESENCE = 0 |
                    +-------------+
                    20
                    2I NON
                    22  TF
                    +--------------+
                    | PRESENCE = 0 * |
                    +--------------+
                      retour OK à CITAT
```

4 OUI
6 NON
7 → RELI

2 OUI
3 SAVE = 25
6 NON
7 NON

retour KO à CITAT

9 NON
II → REL2

2 OUI
3 SAVE = 25
4 NON
7 NON
I 4 = I
9 TQ TL

UN	25
UN	

I0 réponse : NON
II TF

NOM = BI

I2 OUI
I4 OUI
I 4 = 2
I5 TF

PRESENCE = 0 *
NOM = BI ET
COMMUNE = NAMUR

I7

I8 OUI
I4
I7
I8 OUI
I4 NON
I9 ≠
* de TF

TF

PRESENCE = 0 *
NOM = BI ET
COMMUNE = NAMUR ET
PROVINCE = NAMUR *

retour OK à CITAT

I3 OUI
I5 NON
I6 → REL4

I OUI
2 ≠
3 TL

25 DONT ET

4 SWET = I
I5 → RELI

2 OUI
3 SAVE = IO
6 OUI
9 ZTQ = UN
IO OUI
II I 4 = I
I2 NON

retour à REL4

I6 NON
I7 → REL2

2 OUI
3 SAVE = IO
4 OUI
5 ZTQ = UN
6 OUI

TQ

UN
UN
UN

TL

25 DONT
IO ET

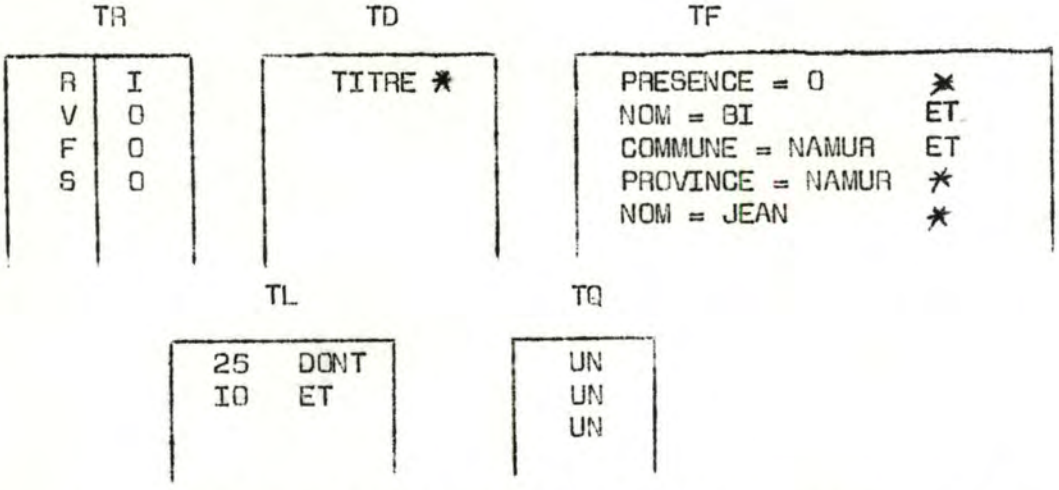
8 résultat : NON
II
I2 NON
I3

TF

PRESENCE = 0 *
NOM = BI ET
COMMUNE = NAMUR ET
PROVINCE = NAMUR *
NOM = JEAN *

retour OK à REL4

I8 OUI
20 OUI



on LR, RECHERCHER TITRE DE UN OUVRAGE (PRESENCE = O) DONT (UN BIBLIOTHEQUE (NOM = BI ET COMMUNE = NAMUR ET PROVINCE = NAMUR) VIA 25 ET UN AUTEUR$(NOM = JEAN) VIA IO) ?

6. Description du passage du langage rigide LR à Socrate.

Rappelons que nous nous sommes limités, pour le passage LN-LR, à des relations d'une des formes suivantes :

- A DE B DE C
- A DONT (B ET C ET D)
- A DONT (B ET C OU D)
- A DONT (B OU C OU D)
- A DONT (B ET C DE D OU F)

C'est donc à des requêtes de ce style que nous allons nous intéresser maintenant. Pour réaliser le passage LN - LR, nous avons construit une table. C'est sur cette dernière que nous allons travailler pour générer l'équivalent Socrate de la requête LN. Ce passage à Socrate va se réaliser par l'intermédiaire de la 'table vue utilisateur'et de la 'table des liens entre entités'. Ces deux tables ont été décrites précédemment.

A chaque relation élémentaire de la "table des liens entre entités", nous allons associer un certain poids qui nous permettra de choisir un chemin d'accès lorsque on devra le construire à partir de cette table. Nous ne faisons que signaler cette possibilité, mais nous ne nous en servirons pas dans le cadre de ce travail. Remarquons que les chemins d'accès représentés dans la "table vue utilisateur" ne sont pas concernés par cette notion de poids. En effet, les relations qui y sont ont été demandées et sont indispensables pour le travail que l'on veut réaliser sur la data base. Dans certains cas, il s'agira de relations que l'utilisateur aura stockées, et donc qu'il considère comme nécessaire pour ses travaux. Il est à remarquer que nous ne construirons un chemin d'accès par la "table des liens entre entités" que dans certains cas où il ne sera pas possible de procéder autrement. Si la structure de la base est bien définie, ces cas devraient être peu fréquents.

Bien que nous n'utiliserons pas la notion de poids, nous allons cependant nous y attarder un peu, car c'est une des premières optimisations à considérer pour un tel système.

6.I. Notion de poids.

Comment attribuer un poids à un chemin d'accès? Deux éléments vont intervenir pour le calculer :

- le nombre de réalisations a atteindre pour sélectionner celle qui est demandée.
- un coéfficient dépendant de la méthode d'accès, et qui sera déterminé après une étude statistique de la banque de données.

Soient Xi = nombre d'occurences de la ième entité

Li = fonction du nombre de filtres permettant de sélectionner une entité i. Li est toujours inférieur à I

Ti = un coéfficient dépendant de la méthode d'accès pour l'entité i. Il sera stocké dans une table.

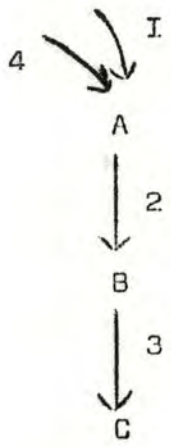
Une étude statistique nous permettra de déterminer le coéfficient Li qui est fonction de la caractéristique filtrante, de sa valeur et de l'opérateur attaché à cette valeur.

Notons que Xi dépend des occurences de l'entité mère de l'entité i, qui est sélectionnée. Si nous avons les deux entités A_{j-I} et A_j, Xj sera le nombre moyen d'occurences de A_j par occurences de A_{j-I}.

On construit alors la formule suivante :

$$TI.\ XI + LI\left[T2.X2 + L2\left[T3.X3 + \ \ldots\right.\right.$$

$$\left.\left. + L_{k-I}\left[Tk.Xk\right]\ldots\right]\right]$$



Pour sélectionner les bonnes occurences de A, nous accédons aux XI réalisation. On va y sélectionner un certain nombre d'occurences que nous estimons par li. Pour chacune de ces sélections, nous allons accéder aux X2 réalisations de B où nous referons la même opération.

Pour la dernière entité sélectionnée, soit C, nous ne faisons pas intervenir le nombre de filtres puisque nous devrons accéder à toutes ses réalisations pour en sélectionner les bonnes, et l'opération est terminée.

Signalons que nous n'avons pas différencié les quantificateurs UN et TOUT. Dans certains cas, il faudra pour UN comme pour TOUT accéder à toutes les occurences d'une entité, parfois, il y aura une assez forte différence entre les deux quantificateurs. Le paramètre Li doit donc aussi tenir compte de ceci.

Nous choisirons pour la phrase Socrate la solution la plus simple à générer. Cela nous permettra de traduire LR en Socrate d'une manière relativement simple, ce qui ne veut pas dire que ce passage ne pose aucun problème. Nous verrons qu'il y a quelques difficultés dues à la hiérarchisation des citations Socrate.

Il est certain que c'est le fait de vouloir générer Socrate automatiquement qui rend le passage difficile. Le travail de traduction ne serait pas tellement compliqué si on le réalisait manuellement, et il serait certainement plus performant. Dans le cadre de cet exposé, nous nous attachons à montrer qu'un passage est possible, et nous donnons une méthode pour le réaliser, et non la méthode idéale. Il est cependant possible de générer Socrate avec un analyseur plus performant; mais ce dernier serait très compliqué et demanderait un temps de réalisation non négligeable.

6.2. Description de ce passage.

I. Les ordres IDEM et PRECISER travaillant au niveau même du langage LN, nous n'avons pas à en prévoir la traduction. Nous n'envisageons pas la programmation directe en Socrate, mais il est utile de signaler que ce langage ne possède pas d'ordres équivalents aux deux cités ci-dessus.

2. Nous nous sommes intéressés dans ce travail au seul aspect consultation de la data base. Nous emploierons souvent dans le généré Socrate l'ordre "I" qui symbolise une recherche et la sortie du résultat. Si ce qui suit cet ordre est entre guillemets, cela signifie qu'il faut l'imprimer tel quel.

3. Les opérations de vérifications seront réalisées grâce à la commande Socrate SI...ALORS...SINON...

4. Les fonctions seront considérées comme des macros préprogrammées auxquelles on passera les paramètres nécessaires. Nous reviendrons plus loin sur ce problème de fonctions.

5. Rappelons que nous n'avons pas envisagé la transformation d'une expression booléenne LN en son équivalent Socrate. Nous avons signalé que la priorité des opérateurs est différente entre ces deux langages.

Nous allons maintenant établir une correspondance entre LR et Socrate pour les commandes prises en considération.

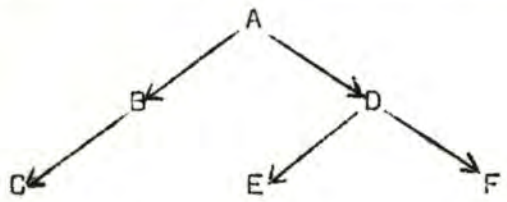
RECHERCHER	Interrogation
SI...ALORS...SINON	SI...ALORS...SINON
VERIFIER	Interrogation + réponse " oui ", " non "
Fonction	Macro
POUR...VERIFIER	= vérifier

Nous pouvons constater qu'en Socrate, tout se ramène à un ordre de consultation: "I". C'est la table TR qui nous permettra de connaître l'équivalent LN.

De façon à garder une forme générale à la génération de Socrate, nous traduirons toute requête LR au moyen de la boucle POUR. Cela n'est pas un handicap pour le résultat Socrate, car dans l'interprétation d'une requête, le système commence toujours par l'entité la plus externe, ce qui équivaut à transformer la requête en une série de "POUR".

Donnons un exemple simple de présentation du résultat d'une traduction de LR.

Soit une structure représentée comme suit :



Si on s'intéresse à une occurence de l'entité C, on formulerait la requête suivante en Socrate.

I d de C ayant a; de B ayant b; de A ayant c; ?

où A, B, C sont des entités, et a, b, c sont des filtres, c'est-à-dire des critères de sélection, et d est la caractéristique demandée.

Nous pourrions aussi avoir la représentation sous forme de boucle POUR (ou forme préfixée) :

```
Pour A ayant c;
    Pour B ayant b;
        Pour C ayant a;
            I d
        fin
    fin
fin ?
```

C'est cette forme que nous choisirons.

Il est aussi nécessaire de souligner que le nom des entités employées par l'utilisateur peuvent être légèrement différents de ceux définis. Par exemple, l'entité AUTEURS est définie avec "S" alors que la requête LN contiendra souvent " AUTEUR ". Comme nous ne parlons pas du mécanisme d'analyse des mots, cela n'est pas très important. Il serait cependant intéressant de permettre des écarts de vocabulaire.

6.3. Requête à une entité.

Pour de telles requêtes la table TL contient un seul nom d'entité, donc TQ n'a qu'un seul quantifiant.

Exemple : Rechercher le nom de l'auteur de code 4 et de cote 5 ?

L'analyse du passage LN - LR que nous venons de faire, nous donne la table suivante :

TR		TD	TF	TL	TQ
R	I	nom *	code = 4 et cote = 5 *	Auteurs	UN
V	0				
F	0				
S	0				

Comment allons nous transcrire cela en Socrate? Le seul problème qu'il nous faut résoudre ici est celui de le hiérarchisation de la requête. Il nous suffira pour cela de passer dans une table TQS dans laquelle nous pourrons inscrire les entités nécessaires, lorsque l'entité appartenant à la requête n'est pas située à l'extérieur dans la structure Socrate.

Le processus général sera :

I) Analyser TR de façon à déterminer le mot de commande.

2) Regarder dans la "Table des liens entre entités" quel chemin nous permet d'accéder à l'entité contenue dans la requête. Le premier chemin trouvé sera sélectionné car ils sont classés de telle façon que celui-ci soit le plus simple.
Dans la partie gauche de cette table nous trouverons l'entité concernée, et dans la partie droite le chemin d'accès à cette dernière.

3) Grâce à la table TQS et TV nous pourrons construire la requête Socrate. Rappelons que TV est l'ensemble TR, TD, TF, TL, TQ.

a) Pour l'exemple ci-dessus nous aurons :

I) Rechercher; car TR a son switch R mis à I

2) Auteur est accèssible directement, car un astérisque est repris dans la "Table des liens entre entités". Donc, TQS est vide.

3) On traduit directement en Socrate à partir de TV, vu que TQS est vide.

```
Pour un auteur ayant code=4 et cote=5 ;
     (I)   (2)                (3)
     I     nom
     (4)    (5)
Fin
```

(I) : TQ
(2) : TL
(3) : TF
(4) : TR
(5) : TD

b) Prenons un exemple où l'entité connue n'est pas accessible directement. Soit : Code du département de nom = X ?
Nous obtenons toujours selon la technique vue dans le passage LN - LR, la table TV suivante :

TR		TD	TF	TL	TQ
R	I	code*	nom =*	Département	un
V	0				
F	0				
S	0				

I) Rechercher sera le mot de commande car TR(R) = I

2) Nous obtenons l'accès suivant Bibliothèque,Gestionnaire

TQS

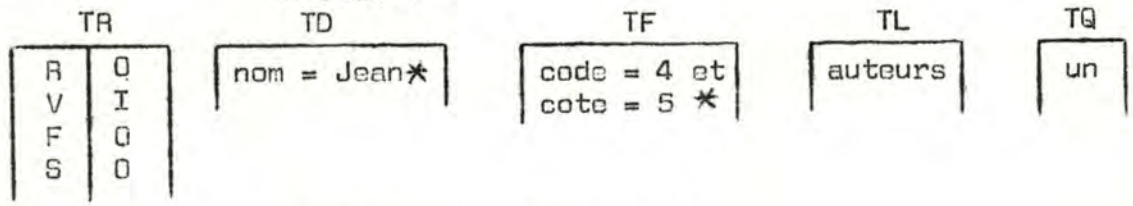
tout Bibliothèque
tout gestionnaire

3) On va ainsi **obtenir**

```
Pour tout Bibliothèque
 Pour tout Gestionnaire
  Pour un auteur ayant nom = X ;
       I code
  fin
fin
fin ?
```

Un <u>compteur de "pour"</u> nous permet de générer le nombre de "fin" nécessaire.

c) " Vérifier si le nom de l'auteur de code 4 et de cote 5 = Jean ? "

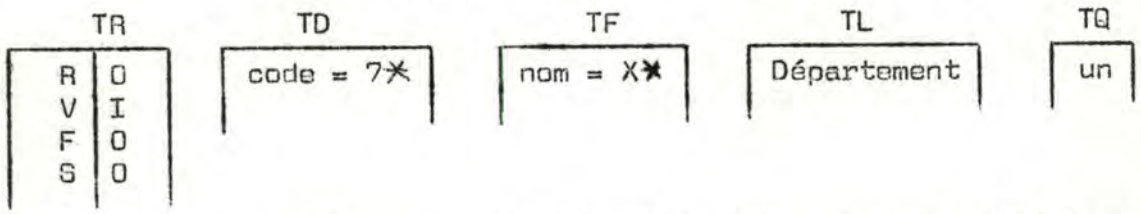


Nous aurons en Socrate :

```
Pour un auteur ayant code = 4 et cote = 5 ;
   si nom = Jean
      alors I oui
      sinon I non
   fin
fin ?
```

d) Vérifier que le code du département de nom X = 7 ?

Nous obtenons toujours la table TV suivante :



Au moyen de la "Table des liens entre entités" nous déterminons l'accès BIBLIOTHEQUE, GESTIONNAIRE

Nous construisons alors

TQS

toute bibliothèque tout gestionnaire

et nous générons alors

```
Pour toute bibliothèque
  Pour tout gestionnaire
    Pour tout département ayant nom = X ;
      si code = 7 alors I OUI
                        SORT
                   sinon I non
      fin
    fin
  fin
fin ?
```

Par contre, si on s'intéresse à faire cette vérification pour TOUT département, nous aurons comme modification

Nous aurons en Socrate

```
MYI = 0
Pour toute bibliothèque
  Pour tout gestionnaire
    Pour tout département ayant nom = X ;
              si code = 7 alors MYI = I
                          sinon MYI = 0
                                SORT
              fin
    fin
  fin
fin
Si YI = I alors I " OUI "
          sinon I " NON "
fin ?
```

SORT nous permet de sortir de toutes les boucles. Cet ordre n'est pas prévu dans Socrate. Nous allons encore envisager le cas où la requête contient une relation avant de donner les organigrammes correspondants.

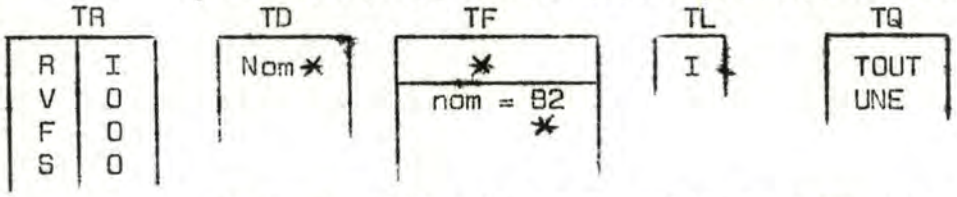
6.4. Requête à une relation.

Comme dans le paragraphe précédent, nous allons aussi décrire succinctement le mécanisme puis nous passerons aux organigrammes.

Nous allons construire une table "TTR" qui comportera des colonnes référence, *, qualifiant, entité, filtre, demande. Si l'accès se fait via des références, nous noterons ces dernières dans la colonne référence. La colonne * contiendra "I" si l'entité correspondante est marquée de ce même signe dans l'accès, sinon elle contiendra "O" .

Prenons quelques exemples pour exposer ceci. La table TV sera toujours obtenue comme vu dans le passage LN - LR.

a) Rechercher le nom de tout emprunteur inscrit à B2 ?



Notons qu'il est assez facile de retrouver pour les quantificateurs TOUT et UN, les entités correspondantes.

En effet,

le 2° élément de la relation correspond au I quantificateur.

le I° élément de la relation correspond au 2 quantificateur.

Si nous regardons, dans la "Table vue utilisateur" nous voyons que la relation "I"relie Bibliothèque et Emprunteur

donc nous avons $\Longrightarrow$ Tout emprunteur
Un Bibliothèque

Construisons TTR

ref	X	qualifiant	entité	filtre	demande
	O	une	bibliothèque	nom = B2	
emprunteur	O	tout	emprunteur		nom

L'accès donné par "Table vue utilisateur" est Bibliothèque, emprunteur (emprunteur), c'est-à-dire la Iere ligne de cette table vu que la relation a le numéro "I".
Nous n'avons pas d'astérisque, dons la colonne✱ sera à 0.
Regardons bibliothèque : le qualifiant donné est un ; TF donne le filtre qui est nom = B2.
Pour emprunteur : par convention, la caractéristique entre parenthèses est une référence et nous la placerons dans la colonne ref. TQ indique que le qualifiant est tout et TFque le filtre n'existe pas puisqu'elle contient un astérisque. TD nous donne la caractéristique demandée.
Nous pouvons construire la requête Socrate en parcourant cette table de haut en bas.

```
Pour une bibliothèque ayant nom = B2 ;
    Pour tout emprunteur
        I nom de emprunteur
    fin
fin ?
```

b) Quel est le nom de la bibliothèque qui possède TI ?

TR	
R	I
V	0
F	0
S	0

TD
nom ✱

TF
✱
Titre = TI✱

TL
4

TQ
UN
UN

La ligne 4 de la "Table vue utilisateur" nous donne le chemin d'accès:

✱bibliothèque, ouvrage

Nous construisons la table TTR comme dans l'exemple précédent avec en plus un "I" dans la colonne en face de bibliothèque vu qu'il y a un astérisque dans le chemin d'accès devant cette entité.

TTR

ref	✱	qualifiant	entité	filtre	domande
	I	une	bibliothèque		nom
	0	un	ouvrage	titre=TI	

Nous pouvons constater que la demande est relative à la première entité de TTR et non plus à la seconde. Cela est reconnaissable grâce à l'astérisque ("I" dans la colonne *)

```
Pour toute bibliothèque XI
    si existe un ouvrage ayant titre = TI ;
       alors I nom de XI
            SORT
    fin
fin ?
```

Nous remarquerons aussi que le quantifiant relatif à bibliothèque est toute et non une comme indiqué dans TTR. Cela peut aussi se déterminer par le "I" dans la colonne * . Nous voyons ici que le qualifiant un peut parfois devenir tout pour exprimer la requête correctement en Socrate. Il serait intéressant de voir ce que donnerait la même requête si nous avons toute bibliothèque.

c) Quels sont les noms des bibliothèques qui possèdent TI ?
La table TTR est la même que dans l'exemple b, seuf en ce qui concerne le qualifiant de bibliothèque.

ref	*	qualifiant	entité	filtre	demande
	I	toute	bibliothèque		nom
	O	un	ouvrage	titre=TI	

Nous aurons en Socrate:

```
Pour toute bibliothèque XI
   si existe un ouvrage ayant titre = TI ;
      alors I nom de X I
   fin
fin ?
```

Ce qui différencie ce cas du précédent, c'est l'absence de la commande **SORT** dans le Si dès que l'on a trouvé un ouvrage.

d) Prenons un exemple où la requête doit être hiérarchisée.

Quels sont les départements qui disposent de MI ?

TR

R	I
V	O
F	O
S	O

TD

nom*

TF

*
nom=MI*

TL

I7

TQ

tout
un

I7 a pour accès * bibliothèque, * gestionnaire

* département, matériel

La table TTR est la suivante :

ref	*	qualifiant	entité	filtre	demande
	I	tout	bibliothèque		
	I	tout	gestionnaire		
	I	tout	département		nom
	O	un	matériel	nom=MI	

Pour toutes les entités de hiérarchisation nous prenons le quantificateur TOUT. Il est facile de savoir quelles sont ces entités, car ce sont celles qui appartiennent à l'accès et non à la relation citée dans la requête.

Exprimons cette question en Socrate.

```
Pour tout bibliothèque
    Pour tout gestionnaire
        Pour tout département XI
            si existe un matériel ayant nom = MI ;
               alors I nom de XI
            fin
        fin
    fin
fin  ?
```

Nous venons de voir une série d'exemples. Dans ceux-ci nous avons eu besoin d'un démonstratif XI. Dans certains cas, il nous faudra plusieurs de ces démonstratifs, spécialement quand il nous faudra, dans une boucle pour imbriquée dans d'autres, demander une valeur de caractéristique se trouvant dans une entité de cette série de pour.

Pour éviter des retours en arrière nous générerons toujours les Xi de façon à pouvoir s'en servie dès que nécessaire.

e) Quel gestionnaire comptabilise MI ?

Nous ajoutons pour cet exemple la relation

27 : comptabilise : Matériel → Gestionnaire

accès ✱ Bibliothèque ✱ gestionnaire

département, matériel

TR		TD	TF	TL	TQ
R I V O F O S O		nom ✱	✱ nom= MI✱	27	un un

Construisons TTR

ref	✱	qualifiant	entité	filtre	demande
	I	tout	bibliothèque		
	I	un	gestionnaire		nom
	I	tout	département		
	O	un	matériel	nom=MI	

La solution Socrate est

```
Pour toute bibliothèque XI
  Pour tout gestionnaire X2
    Pour tout département
        Si existe un matériel ayant nom = MI ;
        alors I nom de X2 de XI de XO
              SORT
        fin
    fin
  fin
fin ?
```

Si nous désirons :

Quels gestionnaires comptabilisent MI ?

Nous aurons le même schéma de tables et l'équivalent Socrate sera :

```
Pour toute bibliothèque XI
  Pour tout gestionnaire X2
    Pour tout département
        Si existe un matériel ayant nom = MI ;
        alors I nom de X2 de XI de XO
           sortie ( I )
        fin
    fin
  fin
fin  ?
```

La commande sortie (n) n'existant pas en Socrate, est ajoutée pour permettre de sortir de n boucles.

6.5. Expression SI...ALORS...SINON.

Soit la question Si A alors B sinon C.

Une requête de cette forme n'apporte rien de neuf quant à sa génération en Socrate. En effet, A, B, C sont des requêtes identiques à celles que nous venons d'étudier.

Le seul problème est celui de la modification à apporter à la routine SRECH ou SSRECH dans le cas d'un SI. Le module destiné à traiter l'expression SI...ALORS...SINON... ne sera en fait qu'un ensemble d'instructions destinées à modifier les générations obtenues dans le cas d'une vérification.

Nous allons simplement décrire la suite des opérations exigées par une telle requête sans entrer dans les détails d'un organigramme.

Nous avons choisi un exemple simple mais la technique est aussi valable pour toutes les formes de relation que nous venons d'étudier.

Soit l'exemple SI COTE DE UN AUTEUR DI DE CODE = 4
ALORS RECHERCHER LE NOM DE DI
SINON RECHERCHER LA COTE DE DI ?

DI est un démonstratif qui sera stocké dans une table pour permettre au système de rattacher une occurence d'application à son occurence de définition.

Notons que ce démonstratif joue à peu près le même rôle que celui utilisé en Socrate. Nous supposons que ce démonstratif garde sa valeur même quand la question est traitée, cela permettra à l'utilisateur de scinder une question en plusieurs sous-questions, et évitera par le fait même, des questions trop complexes. En conversationnel, il est plus intéressant de poser des requêtes relativement simples dans leur formulation, et il nous semble que c'est ce qui se passe dans la plupart des cas.

Pour la requête donnée en exemple, nous aurions les tables suivantes :

TR

R	0
V	0
F	0
S	I

I

TR		TD	TF	TL	TQ	
R	0	cote = 4*	code = 50*	auteur	un	2
V	I					
F	0					
S	0					
R	I	nom*	*	DI	un	3
V	0					
F	0					
S	0					
R	I	cote*	*	DI	un	4
V	0					
F	0					
S	0					

A DI sera associée une entrée dans la table des démonstratifs.

Auteur	DI	XI

on y mettra XI démonstratif utilisé par Socrate, on pourrait avoir une autre table pour générer les démonstratifs Socrate.

I se brancher au module Si.

2 nous donne Vérifier si la cote de un auteur DI de code 50 = 4 ?

ce qui donne en Socrate

pour un auteur XI ayant code = 50 ;
si cote = 4 alors

3 nous donne Rechercher le nom de XI ?

ce qui donne en Socrate

I nom de XI

Nous avons donc en Socrate

Pour un auteur XI ayant code = 50 ;
si cote = 4 alors
I nom de XI sinon

4 nous donne Rechercher la cote de DI ?

ce qui donne en Socrate

I cote de XI

Nous avons donc, pour la génération totale en Socrate :

Pour un auteur XI ayant code = 50 ;
Si cote = 4 alors I nom de XI
sinon I cote de XI
fin
fin ?

Les modifications que nous devons apporter à la routine traitant les vérifications, consistent à empêcher la génération des réponses OUI ou NON lorsque l'on vérifie une citation.

Module SI.

Décrivons succinctement le but de ce module.

- Analyser 2.

  Vérifier demande un branchement au module de vérification.
  On y génèrera la bonne séquence grâce à un swicht introduit dans le module général qui indiquera que l'on vient de l'analyse de l'expression SI...ALORS...SINON
  On va générer ce qui est prévu par le module de vérification en se limitant à ALORS.

- Analyser 3.

  On peut constater par la table TR qu'il faut brancher au module traitant RECHERCHER.
  Ensuite on placera la requête ainsi générée après ALORS placé par la génération précédente.
  On génère ce que prévoit le module VERIF, module de vérification, jusque SINON.

- Analyser 4.

  La table TR nous indique qu'un branchement au module traitant RECHERCHER est nécessaire.
  On placera la requête ainsi trouvée à la suite de SINON générée précédemment.
  On continuera de générer ce qui est prévu par le module VERIF de façon à obtenir le nombre de " FIN " nécessaire à la traduction Socrate.

6.6. Les fonctions.

Nous avons déjà introduit cette notion précédemment, et nous allons développer un peu ce que signifie ce terme. Nous savons déjà qu'elles sont reprises dans une table que le système consultera en commençant l'analyse d'une requête. Il y a un problème pour l'analyse de ces fonctions, car nous les considérons comme de macros Socrate. Or une fonction travaille sur une citation tandis qu'une macro travaille avec des paramètres bien déterminés. Le macro-générateur de Socrate effectue simplement un remplacement de chaînes de caractères.

Nous pourrions procéder comme suit. La citation sera générée en Socrate comme nous l'avons vu jusqu'à présent, et elle sera considérée comme paramètre de la macro représentant la fonction.

Ex. : Effectuer un dénombrement de tous les emprunteurs de Namur inscrits à BI.

TR		TD	TF	TQ	TL
R	O	DENOMBREMENT*	province = Namur*	tout	I
V	O		nom = BI*	un	
F	I				
S	O				

SFUNCT fera appel à des modules de UNENTITE ou UNEREL pour interpréter cette requête. Ici aussi une modification devra être apportée à l'organigramme décrit ci-avant. Il s'agirait comme dans le cas du SI, de modifier la séquence finale de la génération. Au lieu de générer " I <commande> ", on ne **gênèrera** que le nom de la fonction. Nous ne travaillerons pas de cette façon, mais nous avons cité un tel cas car l'appel de la fonction y est naturel.

Une telle procédure demanderait une complète modification du macro-**gênérateur** Socrate. Nous allons essayer de trouver une méthode plus proche de ce qui peut se traiter par Socrate.

Nous commencerons cependant par étendre un peu la notion de paramètre par rapport à Socrate. Dans ce langage, un paramètre est une chaîne de caractères et ils sont séparés par un ou plusieurs blancs. Nous donnerons au paramètre un sens plus large, en lui permettant de contenir des blancs. On pourra ainsi passer comme paramètre une partie de la citation qui accompagne la fonction. Nous verrons plus loin comment choisir cette partie. De façon à pouvoir distinguer deux paramètres, nous introduirons un signe de ponctuation.

Ainsi, le rôle de SFUNCT se limite à une vérification du nom de la fonction et des paramètres en fonction de renseignements situés dans la "table des fonctions". Il est évident que maintenant l'appel d'une macro ne sera plus à caractère naturel, il sera plus ou moins rigide. De plus, il va dépendre de la façon dont la macro va être programmée en Socrate.

Notons que si la fonction peut se programmer avec un appel classique de paramètre, cette solution sera choisie. Soit, par exemple, une fonction destinée à calculer le nombre de réalisation d'une entité. Soit DENENT son nom. Dans ce cas, le paramètre est tout simplement le nom de l'entité concernée, du moins au niveau LN. En fait, la véritable macro aura pour nom SDENENT et DENENT sera simplement chargée de faire un appel à cette dernière après avoir modifié le paramètre. Cette modification concernera essentiellement la hiérarchisation de la "citation" paramètre.

Supposons que l'utilisateur demande

DENENT gestionnaire ?

Le rôle de cette macro sera de hiérarchiser le paramètre de façon à le rendre présentable pour l'appel réel. La macro connaîtra son rôle grâce aux renseignements stockés dans la "table des fonctions".

La hiérarchisation sera réalisée par l'intermédiaire de la "table des liens entre entités".

Dans notre exemple, la macro DENENT génèrera

SDENENT gestionnaire de toute bibliothèque ?

et la macro SDENENT sera simplement ceci :

```
! defmac      ,SDENENT !
! exp         M Y2 = 0
              Pour tout ! I !
              M Y2 = Y2 + I
              Fin
              I Y2
! fdef !  ?
```

L'appel de cette macro donnera

```
M Y2 = 0
Pour tout gestionnaire de toute bibliothèque
M Y2 = Y2 + I
FIN
I Y2 ?
```

L'appel d'une macro peut être plus complexe que celui donné dans cet exemple. Si nous avons une requête telle que MOYENNE DES COTES DES AUTEURS INSCRITS A BI ?, comment déterminer les paramètres? Rappelons que ces derniers ne sont pas déterminés selon la requête; mais en fonction de la programmation Socrate de la requête.

Il faut d'abord connaître le nombre de paramètres qu'une macro va accepter. Cela aussi sera stocké dans la table des fonctions. Il serait cependant intéressant de permettre un nombre variable de paramètres. Cela est possible, puisque l'appel LN ou LR d'une macro ne nous branche pas directement sur la séquence d'instructions réalisant l'opération demandée, mais sur un intermédiaire qui jouerait le rôle d'un petit générateur. Il est évident que le travail de la "macro intermédiaire" sera plus simple si le nombre de paramètres est fixé. Elle effectuera alors un simple contrôle.

Une autre méthode consisterait à stocker les résultats fournis par la requête, et les passer comme paramètres avec les caractéristiques concernées.

Prenons un exemple.

Soit Effectuer la moyenne des cotes des auteurs de nom Jean inscrits à BI ?

On pourrait alors donner la commande

Rechercher les auteurs de nom Jean inscrits à BI ?

Un ordre de stockage intermédiaire permettrait de ranger le résultat dans une zone quelconque dont on passerait aussi le nom comme paramètre, ou dans une zone connue du système. Le résultat dont nous parlons ici serait l'adresse virtuelle des occurences de l'entité auteur qui ont été sélectionnées.

Ensuite, on ferait l'appel

SMOYENNE COTE ?

Toute l'opération intermédiaire serait réalisée par un module général de traitement de macro auquel on passerait la main dès que l'on détecte un nom de macro.

6.7..Analyseur de requêtes L_R .

Signification des différentes variables:

SWN : retient que l'on a une demande dans TTR .

SW* : mémorise qu'il y a un astérisque dans l'accès.

SW*T : mémorise le fait que l'on a modifié UN en TOUT après avoir rencontré un astérisque dans TTR .

LD : mémorise la ligne TTR où on a une demande .

K2 : retient que l'on a une demande dans une ligne de TTR autre que l'avant-dernière .

J : mémorise le nombre de lignes de TTR .

I2 : numéro de la ligne de TTR en cours de traitement .

KI : sert d'indice pour générer une suite de Xi dans la requête dans sa partie demande .

K : sert d'indice pour générer les Xi de la formulation Socrate .

Signification des différents modules :

SOCGEN : module général de génération en Socrate .

UNEREL : analyse le cas où la requête ne contient q'une relation .

UNENTITE : analyse les requêtes exprimées par une seule entité .

MULTIREL : analyse les combinaisons de relations .Nous n'en donnons qu'une description .

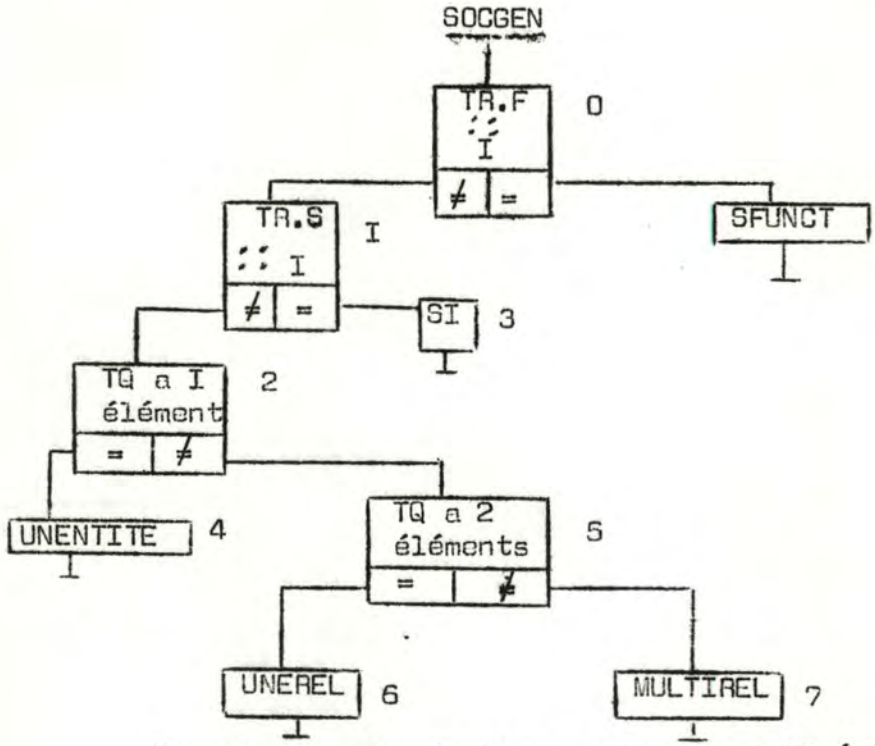
SI : analyse les requêtes du genre SI - ALORS - SINON .Nous n'en donnons qu'une description .

SFUNCT : analyse les fonctions .Nous nous sommes limités à une description .

SRECH : est appelé par UNENTITE et est chargé de générer la requête en Socrate .

SSRECH : est appelé par UNEREL et est chargé de générer la requête en Socrate.

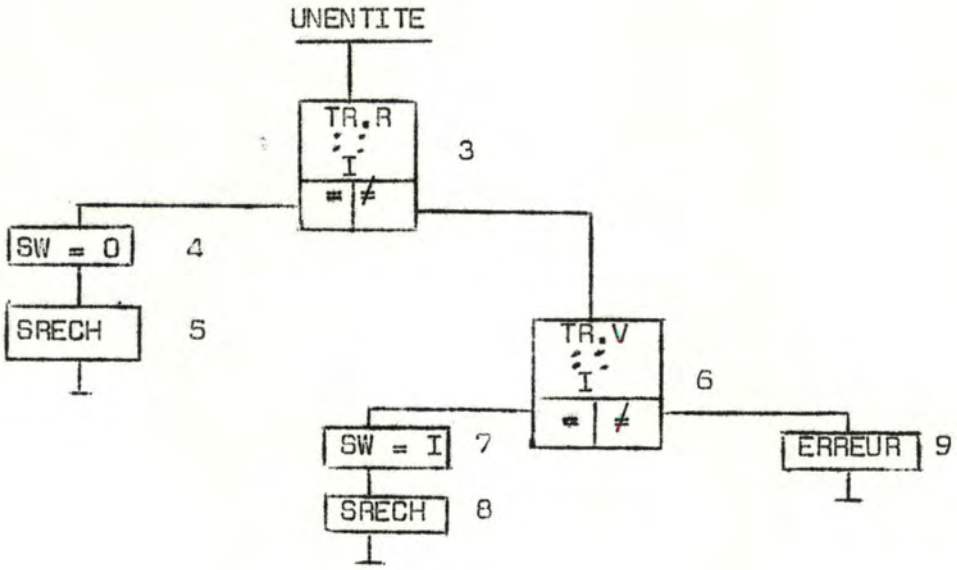
Organigrammes .



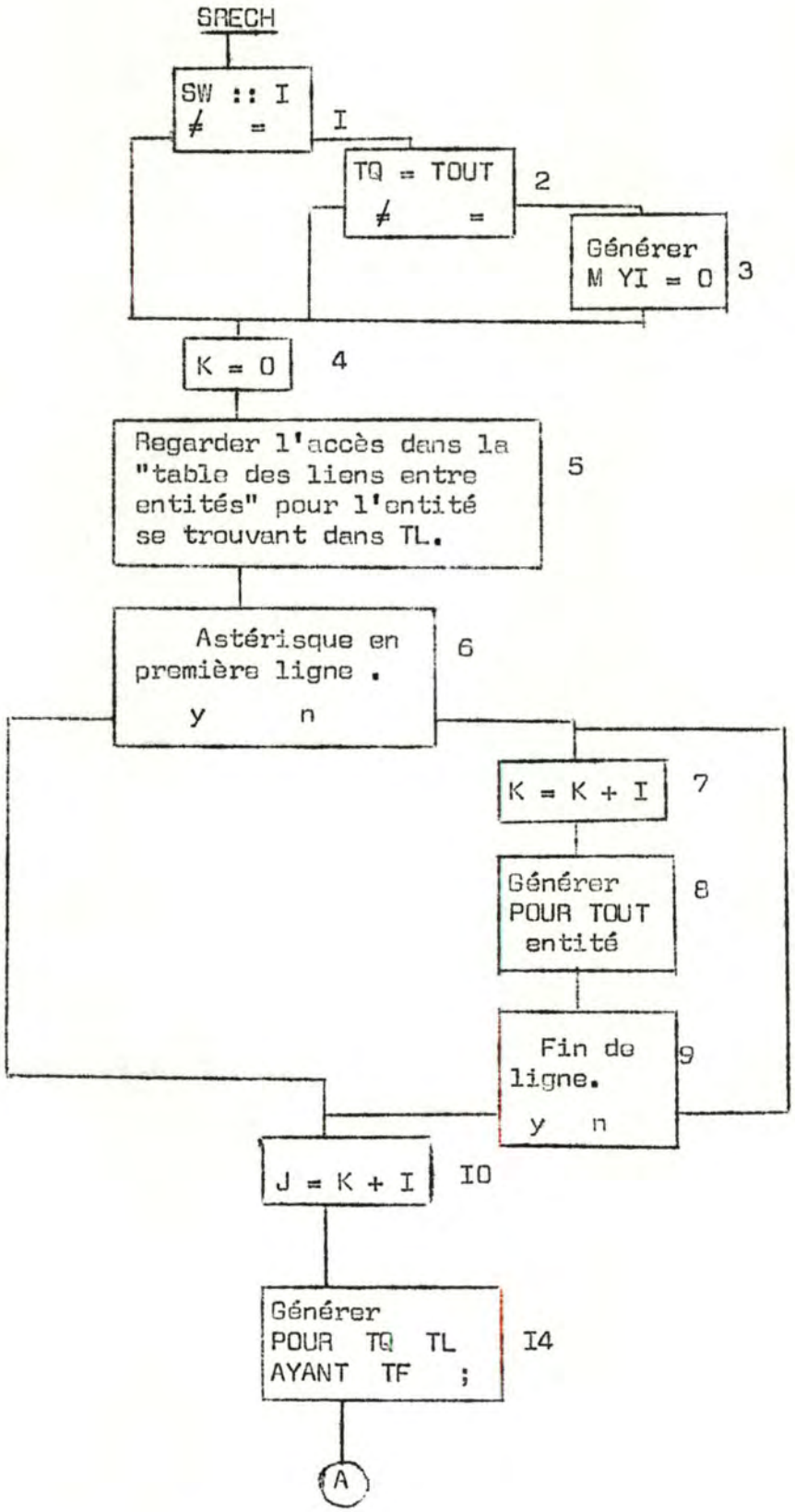
Nous venons de décrire succinctement SI (§ 6.5) et SFUNCT (§ 6.6) sans en donner d'organigrammes.Nous ferons la même démarche pour MULTIREL (§ 6.9) .

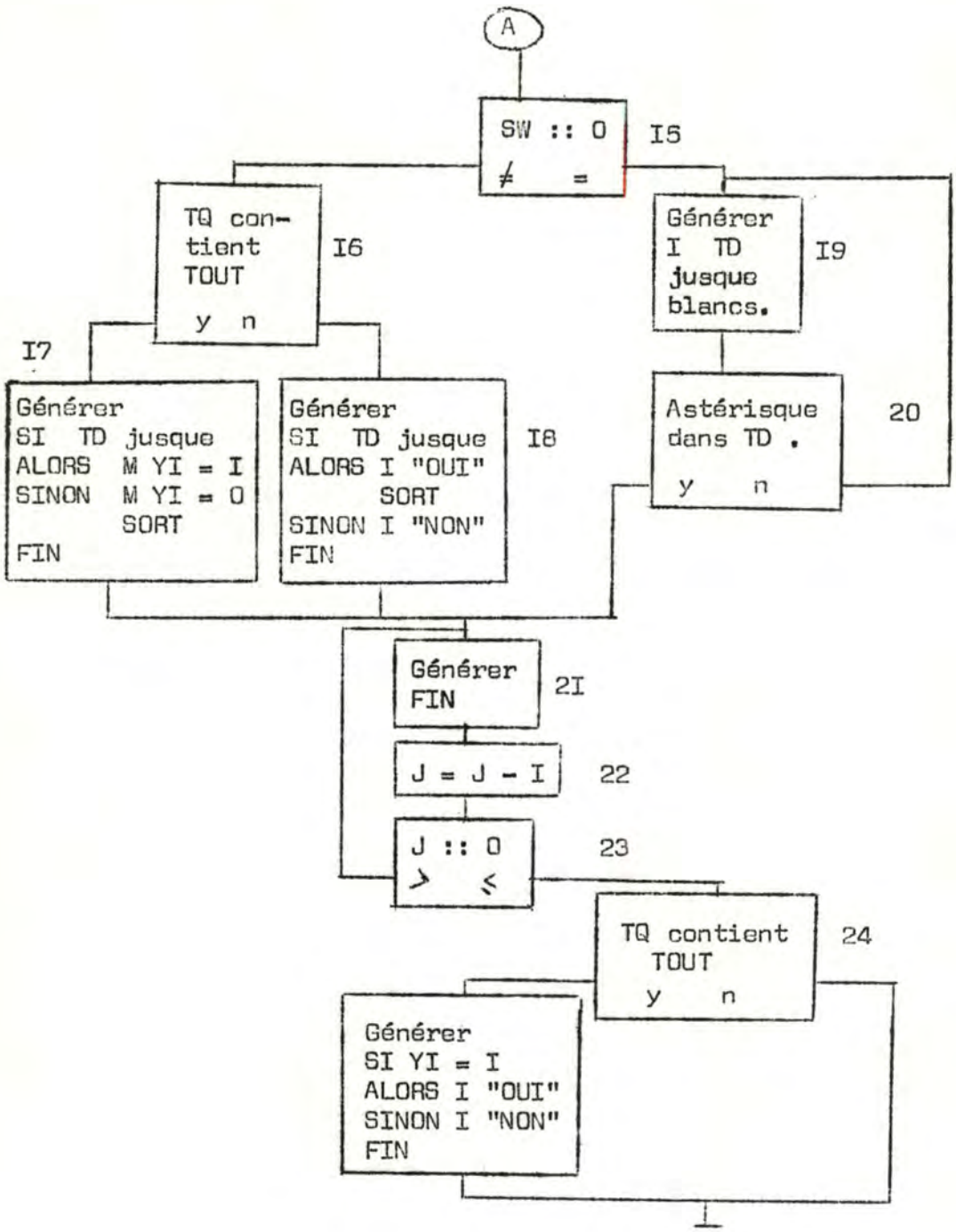
Nous donnons ici les organigrammes de UNEREL et UNENTITE .

TR.F = field F de la table TR de TV .



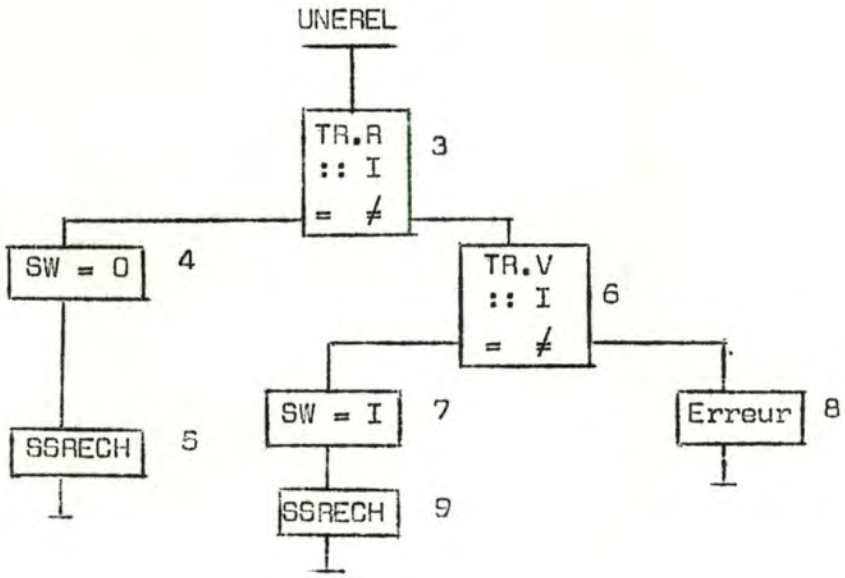
SW va nous permettre de distinguer RECHERCHER de VERIFIER .

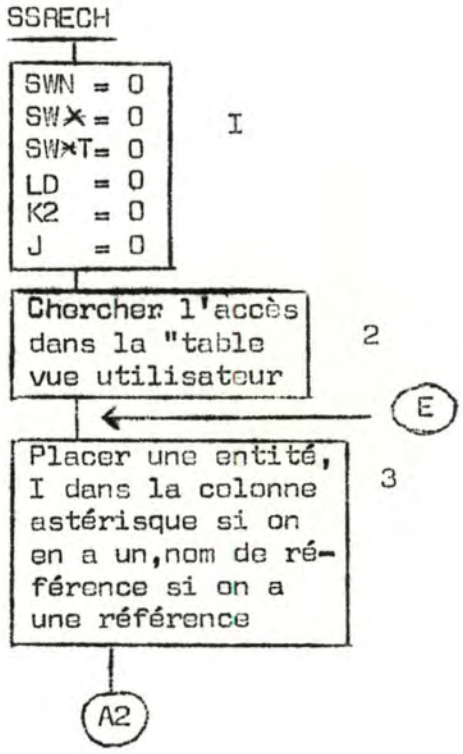

SRECH
SW :: I
≠ =
I
TQ = TOUT
≠ =
2
Générer
M YI = 0
3
K = 0
4
Regarder l'accès dans la "table des liens entre entités" pour l'entité se trouvant dans TL.
5
Astérisque en première ligne .
y n
6
K = K + I
7
Générer
POUR TOUT
entité
8
Fin de ligne.
y n
9
J = K + I
IO
Générer
POUR TQ TL
AYANT TF ;
I4
A

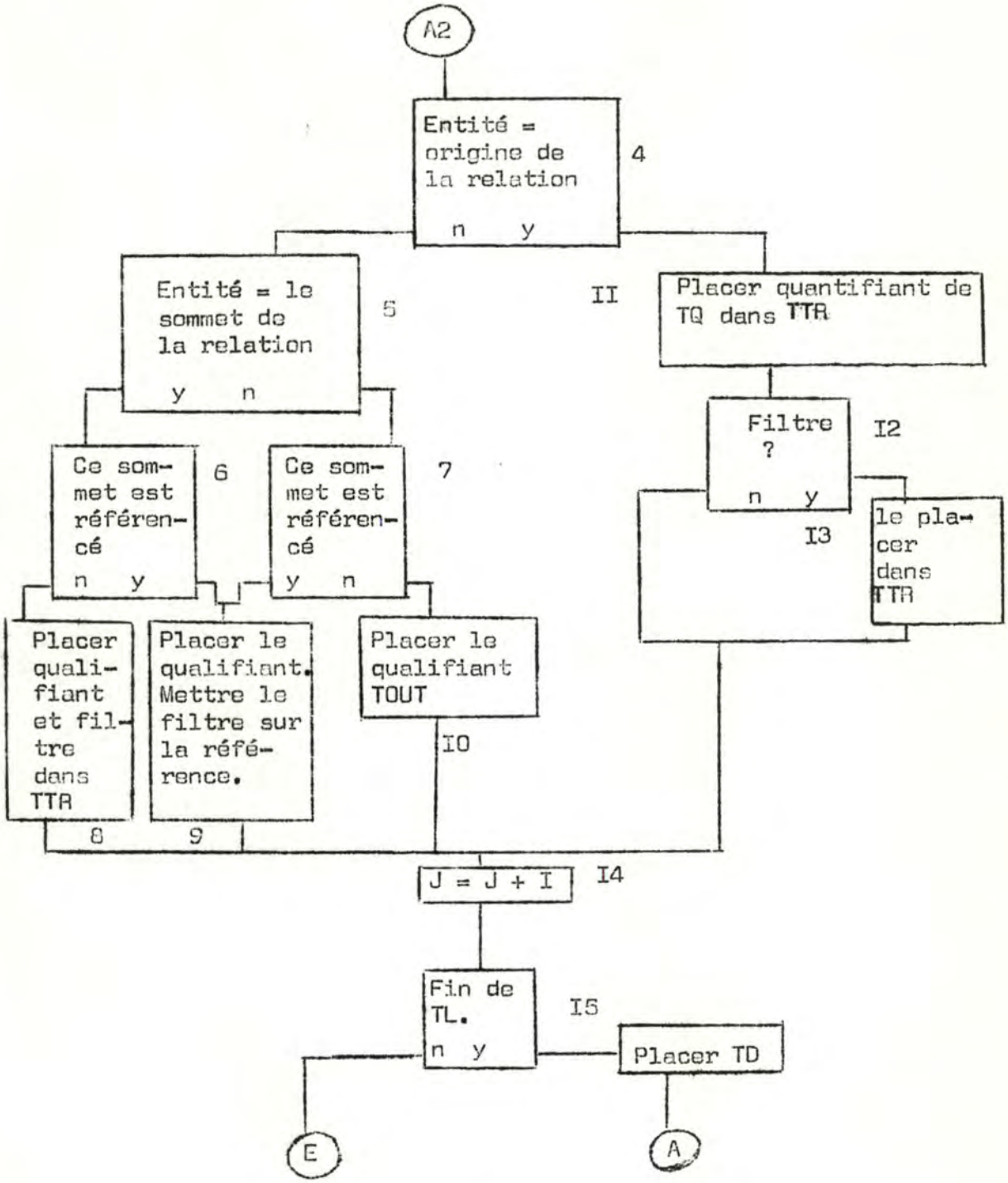


Un astérisque en première ligne signifie qu'on peut accéder directement à l'entité correspondante.

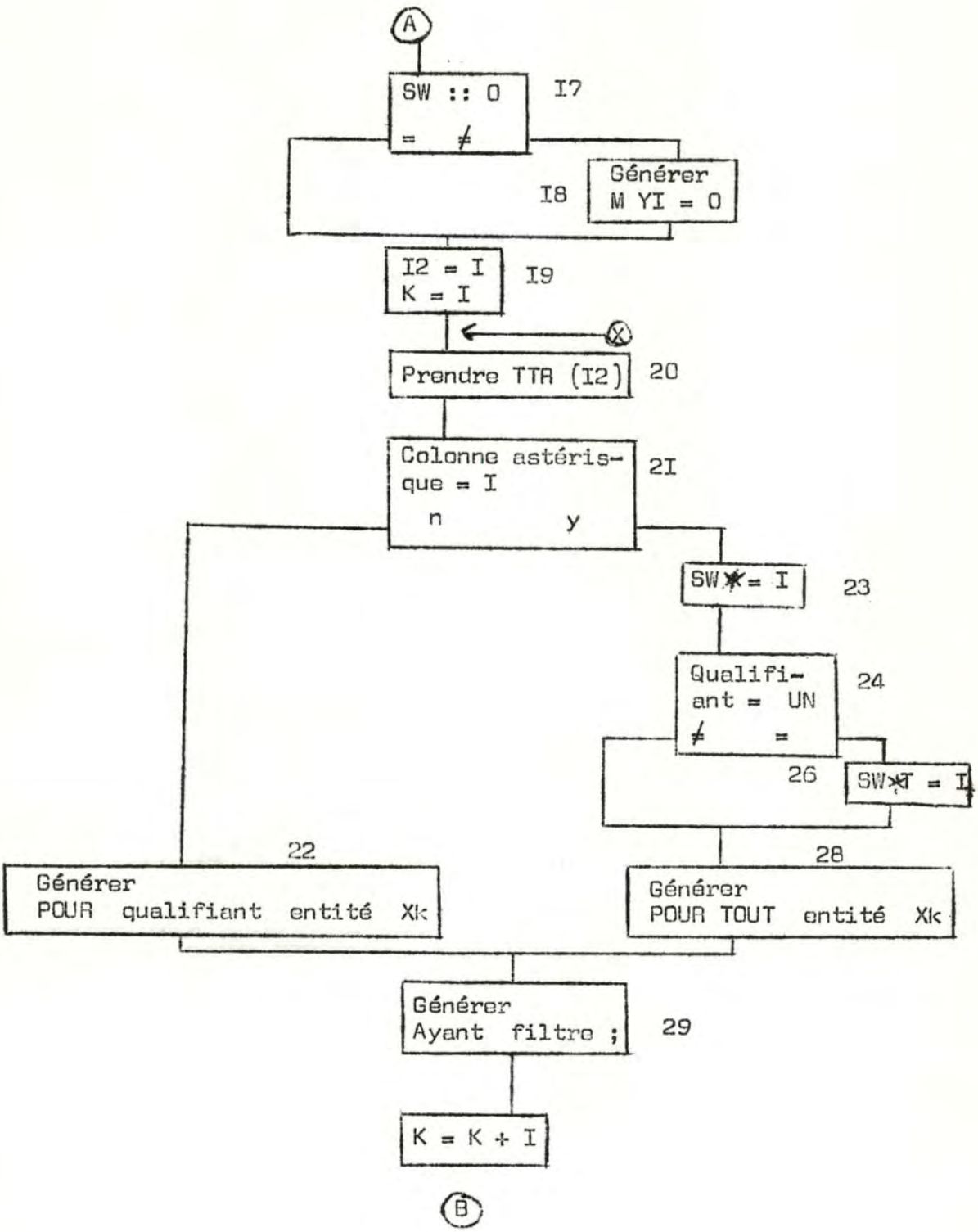
K permettra de générer le nombre de FIN nécessaire.

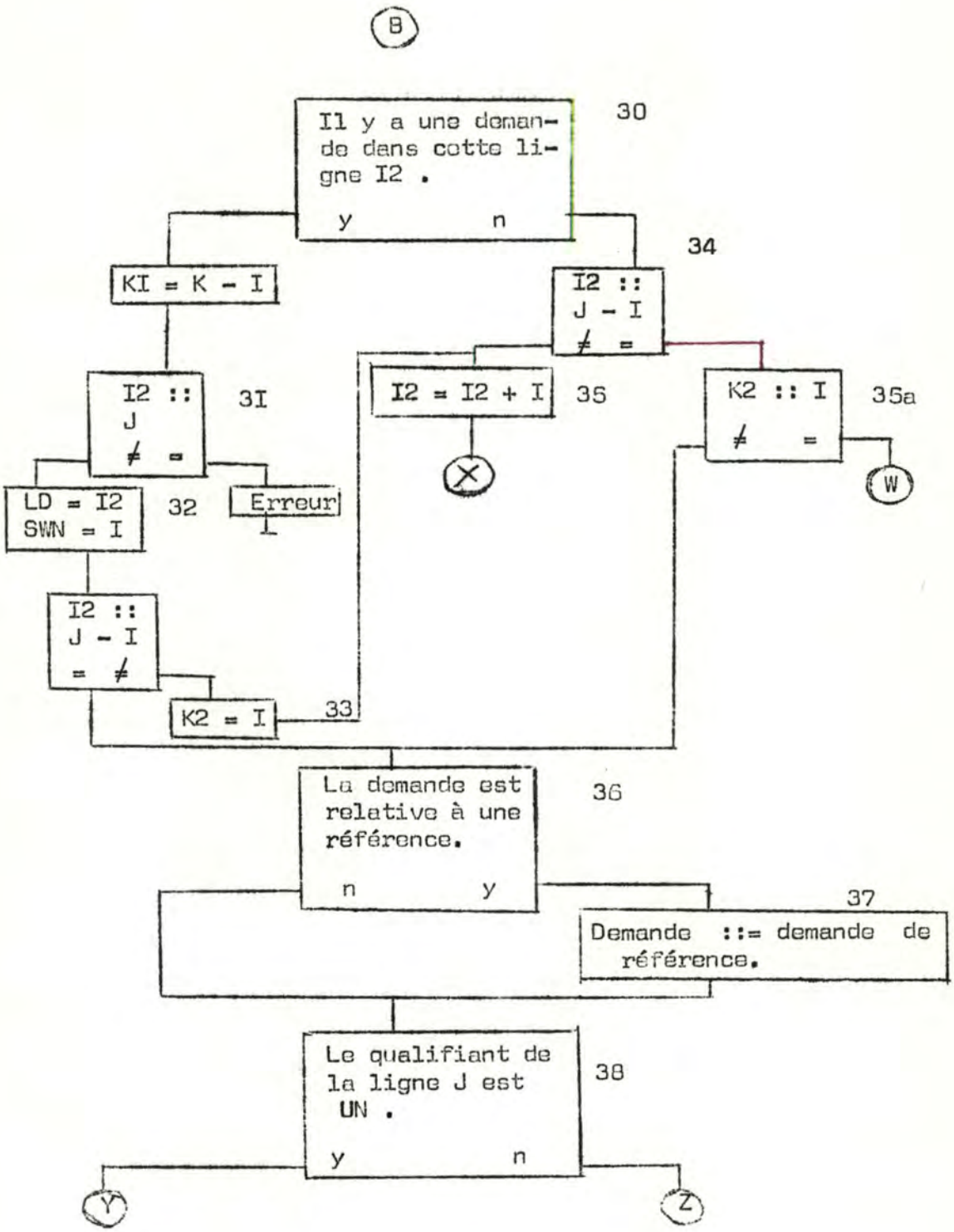

UNEREL
TR.R
:: I
= ≠
3
SW = 0
4
TR.V
:: I
= ≠
6
SW = I
7
Erreur
8
SSRECH
5
SSRECH
9


SSRECH
SWN = 0
SW* = 0
SW*T = 0
LD = 0
K2 = 0
J = 0
I
Chercher l'accès dans la "table vue utilisateur
2
E
Placer une entité, I dans la colonne astérisque si on en a un, nom de référence si on a une référence
3
A2

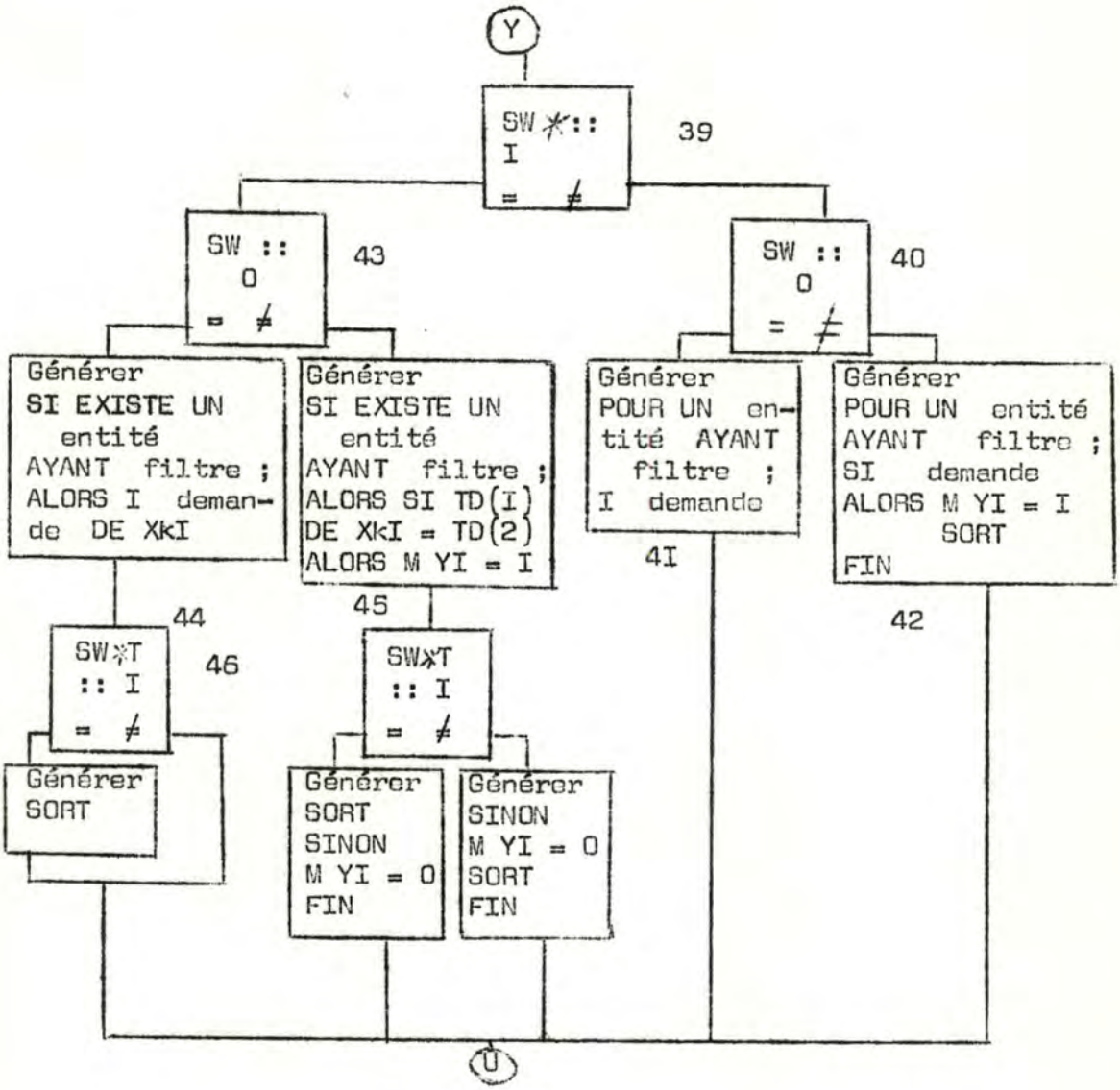


Nous avons ainsi généré la table TTR.

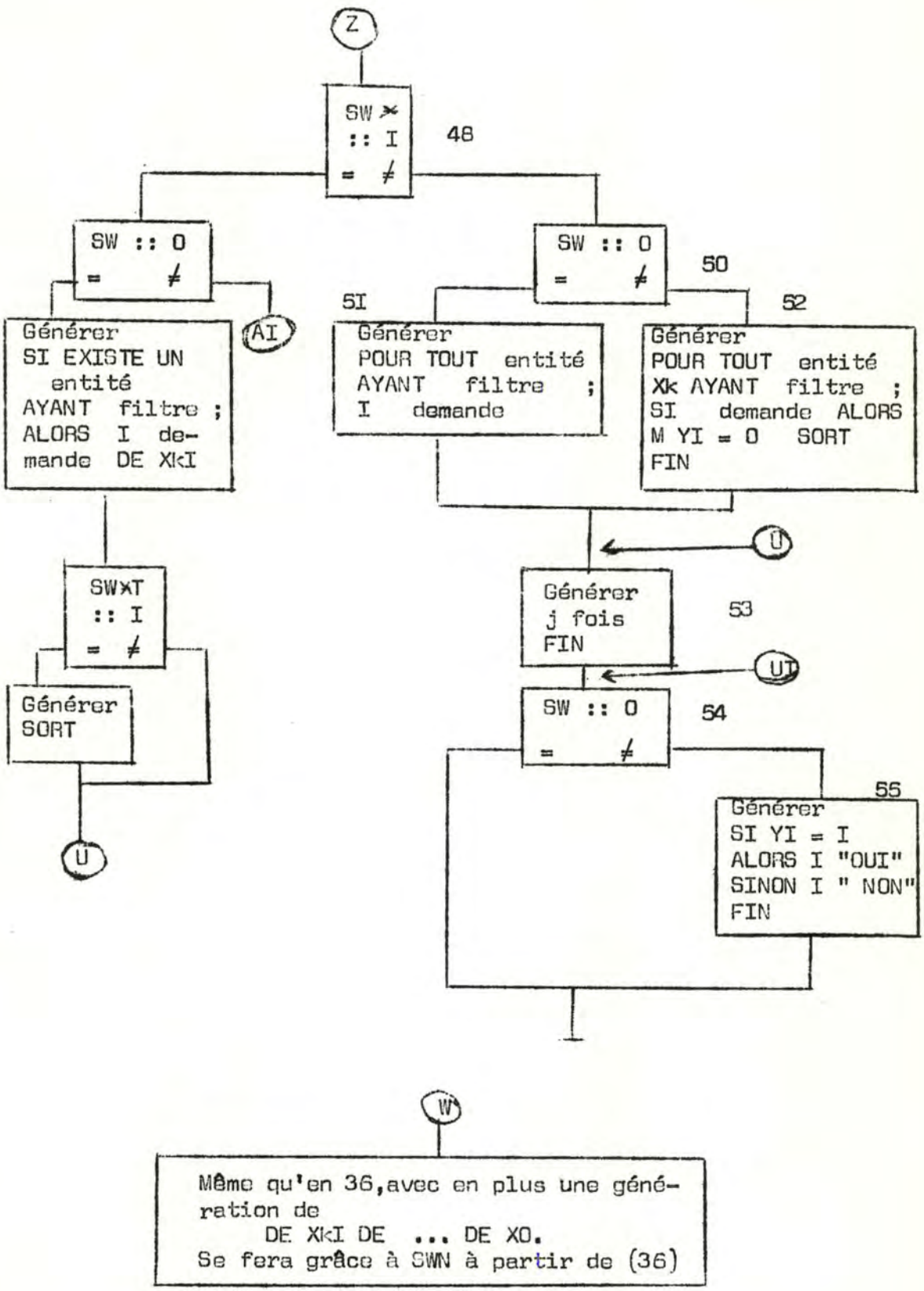

A
SW :: 0
= ≠
I7
I8
Générer
M YI = 0
I2 = I
K = I
I9
X
Prendre TTR (I2)
20
Colonne astéris-
que = I
n y
2I
SW* = I
23
Qualifi-
ant = UN
≠ =
24
26
SW*T = I
22
Générer
POUR qualifiant entité Xk
28
Générer
POUR TOUT entité Xk
Générer
Ayant filtre ;
29
K = K + I
B

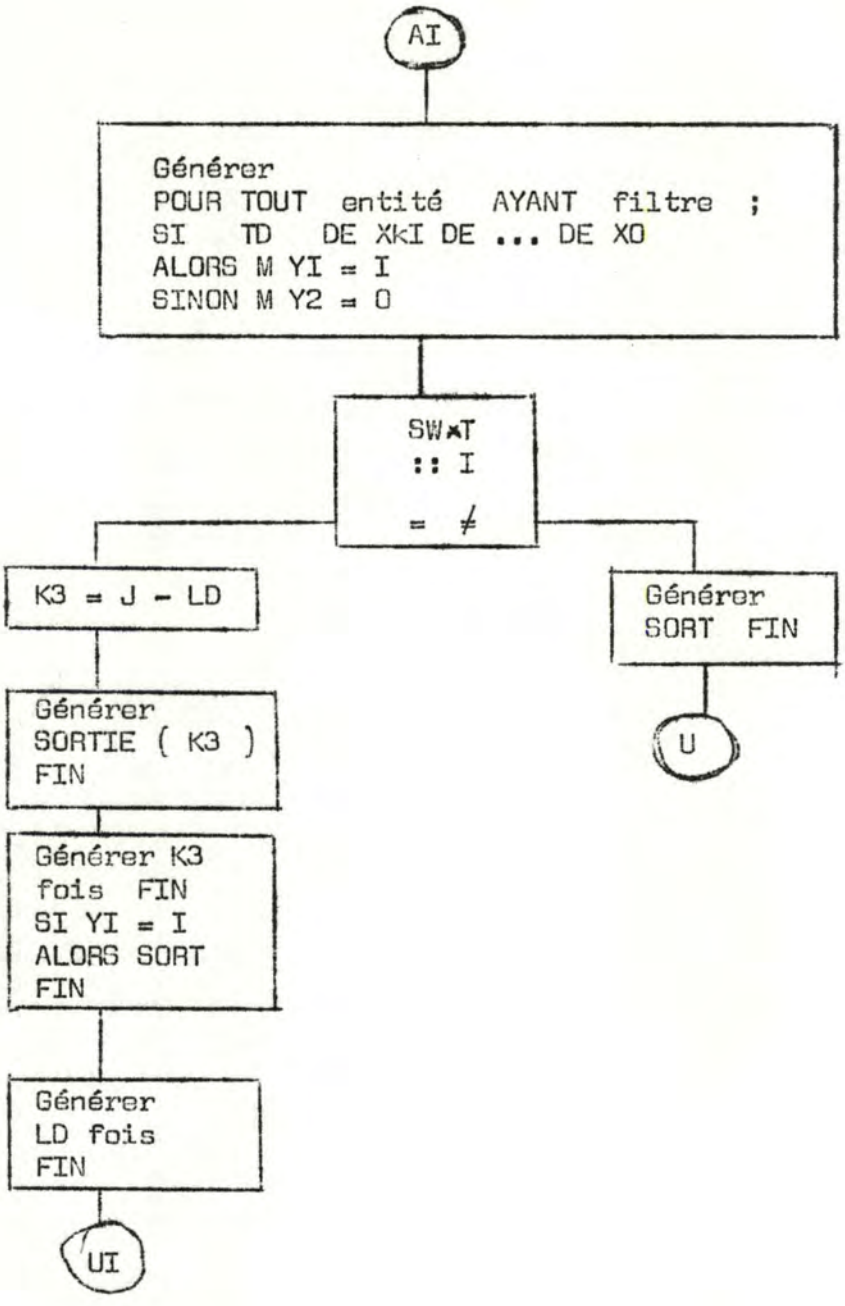


"34 =" et "35a ≠" indiquent que la demande ne se trouve pas dans les (J - I) premières lignes.Elle est donc dans la dernière ligne de TTR.



TD(I) = première partie de TD.
ex.: NOM = JEAN donne TD(I) = NOM
TD(2) = JEAN


Z
SW ✕ :: I = ≠
48
SW :: 0 = ≠
AI
Générer SI EXISTE UN entité AYANT filtre ; ALORS I demande DE XkI
SW✕T :: I = ≠
Générer SORT
U
SW :: 0 = ≠
50
51
52
Générer POUR TOUT entité AYANT filtre ; I demande
Générer POUR TOUT entité Xk AYANT filtre ; SI demande ALORS M YI = 0 SORT FIN
U
Générer j fois FIN
53
UT
SW :: 0 = ≠
54
55
Générer SI YI = I ALORS I "OUI" SINON I " NON" FIN
W
Même qu'en 36, avec en plus une génération de
DE XkI DE ... DE XO.
Se fera grâce à SWN à partir de (36)


AI
Générer
POUR TOUT entité AYANT filtre ;
SI TD DE XkI DE ... DE XO
ALORS M YI = I
SINON M Y2 = 0
SWAT
:: I
= ≠
K3 = J - LD
Générer
SORTIE (K3)
FIN
Générer K3
fois FIN
SI YI = I
ALORS SORT
FIN
Générer
LD fois
FIN
UI
Générer
SORT FIN
U

6.8. Exemples.

Remarquons que l'entité AUTEURS s'écrit avec un "S".L'utilisateur LN peut omettre ce "S".Mais au niveau Socrate,il est nécessaire de respecter l'orthographe des noms.

Nous ne redécrirons pas l'obtention de la table TV (TR,TD,TF, TL,TQ).Cela a été donné au § 5.7.

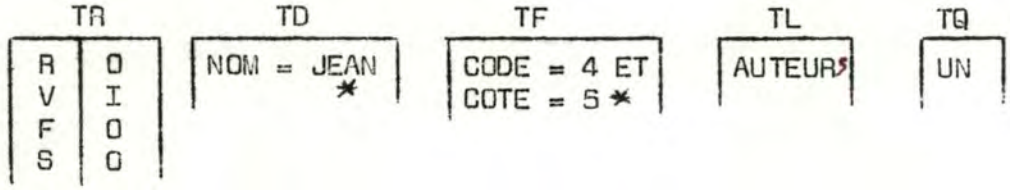
I) Rechercher le nom d'un auteur de code 4 et de cote 5 ?

TR		TD	TF	TL	TQ
R	I	NOM*	CODE = 4 ET	AUTEURS	UN
V	0		COTE = 5 *		
F	0				
S	0				

```
SOCGEN
   0  NON
   I  NON
   2  OUI
   4  UNENTITE
         3  =
         4  SW = 0
         5  SRECH
               I  ≠
               4  K = 0
               5  REGARDE AUTEURS
               6  OUI
              I0  J = I
              I4 |POUR UN AUTEURS AYANT CODE = 4 ET COTE = 5;
              I5  =
              I9 |  I  NOM
              20  OUI
              2I |FIN
              22  J = 0
              23  =
              24  NON

||POUR UN AUTEURS AYANT CODE = 4 ET COTE = 5;
||          I NOM
||
||FIN  ?
```

2) Vérifier si le nom d'un auteur de code 4 et de cote 5 est Jean ?




```
SOCGEN
   0  NON
   I  NON
   2  OUI
   4  UNENTITE
        3  NON
        6  =
        7  SW = I
        8  SRECH
            I  =
            2  NON
            4  K = 0
            6  OUI
           I0  J = I
           I4 | POUR UN AUTEURS AYANT CODE = 4 ET COTE = 5;
           I5  ≠
           I6  NON
           I8 |  SI NOM = JEAN
                    ALORS I "OUI"
                          SORT
                    SINON I "NON"
              |  FIN
           2I | FIN
           22  J = 0
           23  =
           24  NON
```

```
POUR UN AUTEURS AYANT CODE = 4 ET COTE = 5;
     SI NOM = JEAN
            ALORS I "OUI"
                  SORT
            SINON I "NON"
     FIN
FIN ?
```

3) Vérifier si le code des départements de nom X est 7 ?

TR		TD	TF	TL	TQ
R V F S	0 I 0 0	CODE = 7*	NOM = X*	DEPARTEMENT	TOUT

```
SOCGEN
 0  ≠
 I  ≠
 2  =
 4  UNENTITE
     3  ≠
     6  =
     7  SW = I
     8  SRECH
         I  =
         2  OUI
         3  | M  YI = 0
         4  K = 0
         6  NON
         7  K = I
         8  | POUR TOUT BIBLIOTHEQUE
         9  NON
         7  K = 2
         8  | POUR TOUT GESTIONNAIRE
         9  OUI
        I0  J = 3
        I4  | POUR TOUT DEPARTEMENT
            |     AYANT NOM = X ;
        I5  ≠
        I6  OUI
        I7  | SI CODE = 7
            |
            |    ALORS M YI = I
            |
            |    SINON M YI = 0
            |
            |          SORT
            |
            | FIN
```

```
21 |  FIN
22  J = 2
23   >
21 |  FIN
22  J = I
23   >
21 |  FIN
22  J = 0
23  =
24  OUI
25 |  SI YI = I
   |     ALORS  I "OUI"
   |     SINON  I "NON"
   |  FIN
```

```
M YI = 0
POUR TOUT BIBLIOTHEQUE
POUR TOUT GESTIONNAIRE
POUR TOUT DEPARTEMENT AYANT NOM = X ;
            SI CODE = 7
               ALORS  M YI = I
               SINON  M YI = 0
                      SORT
            FIN
FIN  FIN       FIN
SI  YI = I
    ALORS  I "OUI"
    SINON  I "NON"
FIN  ?
```

4) Rechercher le nom des départements qui disposent de MI ?

TR		TD	TF	TL	TQ
R V F S	I 0 0 0	NOM *	* / NOM = MI *	I7	TOUT UN

SOCGEN

0 ≠
I ≠
2 NON
5 OUI
6 UNEREL

3 =
4 SW = 0
5 SSRECH

I SWN = 0 SW* = 0 SW*T = 0
LD = 0 K2 = 0 J = 0

3, 4, 5, 7, I0 donne Ière ligne TTR

ref	*	qualifiant	entité	filtre	demande
	I	tout	bibliothèque		
	I	tout	gestionnaire		
	I	tout	département		nom
	0	un	matériel	nom=MI	

I4 J = I
I5 NON
3, 4, 5, 7, I0 = 2° ligne
I4 J = 2
I5 NON
3, 4, II, I2 = 3° ligne
I4 J = 3
I5 NON
3, 4, 5, 6, 8 = 4° ligne
I4 J = 4
I5 OUI

Placer "NON" dans demande

```
I7  =
I9  I2 = I
     K = I
20  TTR (I)
2I  OUI
23  SW* = I
24  NON
28 | POUR TOUT BIBLIOTHEQUE XI
29  K = 2
30  NON
34  NON
35  I2 = 2
20  TTR (2)
2I  OUI
23  SW* = I
24  NON
28 | POUR TOUT GESTIONNAIRE X2
29  K = 3
30  NON
34  ≠
35  I2 = 3
20  TTR (3)
2I  OUI
23  SW* = I
24  NON
28 | POUR TOUT DEPARTEMENT X3
29  K = 4
30  OUI
    KI = 3
3I  ≠
32  LD = 3
    SWN = I
33  =
36  NON
38  OUI
39  =
43  =
44 | SI EXISTE UN MATERIEL
   |    AYANT NOM = MI ;
   |    ALORS I NOM DE X3
46  ≠
53 | FIN  FIN  FIN  FIN
54  =
```

```
POUR TOUTE BIBLIOTHEQUE XI
POUR TOUT GESTIONNAIRE X2
POUR TOUT DEPARTEMENT X3
    SI EXISTE UN MATERIEL AYANT NOM = MI ;
       ALORS I NOM DE X3
    FIN
FIN  FIN  FIN ?
```

```
30  NON
34  =
35a ≠
36  OUI
37  demande devient
          NOM DE EMPRUNTEUR
38  NON
48  ≠
50  =
5I  |POUR TOUT EMPRUNTEUR
    |  I NOM DE EMPRUNTEUR
53   FIN  FIN
54  =
```

```
POUR UNE BIBLIOTHEQUE XI AYANT NOM = BI ;
POUR TOUT EMPRUNTEUR
     I NOM DE EMPRUNTEUR
FIN  ?
FIN
```

6) Vérifier pour tout emprunteur inscrit à BI si province = Namur et commune = Namur ?

TR		TD	TF	TL	TQ
R V F S	0 I 0 0	PROVINCE = NAMUR ET COMMUNE = NAMUR *	* NOM = BI*	I	TOUT UN

SOCGEN

5 =

UNEREL

6 =

7 SW = I

9 SSRECH

I SWN = 0
SW* = 0
SW*T = 0
J = 0
LD = 0
K2 = 0

3, 4, II, I2, I3 = I° ligne

ref	*	qualifiant	entité	filtre	demande
emprunteur		un tout	bibliothèque emprunteur	nom=BI	province= Namur et commune= Namur

I4 J = 2

I5 NON

3, 4, 5, 6, 9 = 2° ligne

I7 ≠

I8 M YI = 0

I9 I2 = I
K = I

20 TTR (I)

2I NON

22 | POUR UN BIBLIOTHEQUE XI AYANT NOM = BI ;

29 K = 2

30 NON

34 =

35a ≠

```
36  OUI
37  PROVINCE DE EMPRUNTEUR = NAMUR
    ET
    COMMUNE DE EMPRUNTEUR = NAMUR
38  NON
48  ≠
50  ≠
52  POUR TOUT EMPRUNTEUR X2 SI PROVINCE DE EMPRUNTEUR
    = NAMUR ET COMMUNE DE EMPRUNTEUR = NAMUR
    ALORS M YI = I
    SINON M YI = 0
          SORT
    FIN
53   FIN  FIN
55   SI YI = I  ALORS I "OUI"
                SINON I "NON"
                FIN
```

```
M YI = 0
POUR UN BIBLIOTHEQUE XI AYANT NOM = BI ;
POUR TOUT EMPRUNTEUR X2
     SI PROVINCE DE EMPRUNTEUR = NAMUR
        ET COMMUNE DE EMPRUNTEUR = NAMUR
        ALORS M YI = I
        SINON M YI = 0
              SORT
     FIN
   FIN
FIN

SI YI = I  ALORS I "OUI"
           SINON I "NON"
             FIN ?
```

Nous allons traiter une requête en modifiant certains quantificateurs.

Nous supposerons TTR construite, toujours selon le même mécanisme, et nous commencerons le détail à partir du module SSRECH, commande I7.

La requête est la suivante :

Vérifier si le nom du gestionnaire s'occupant de DI est Jules?

S'occupant est la relation n° I5

Son accès est* Bibliothèque, * Gestionnaire, Département.

Nous allons traiter 3 cas :

I. UN gestionnaire, un département

2. un gestionnaire, tout département

3. tout gestionnaire, tout département

La table TTR sera de la forme

ref	*	qualifiant	entité	filtre	demande
	I	tout	bibliothèque		
	I	A	Gestionnaire		nom = Jules
	O	B	Département	nom=DI	

A et B vaudront successivement

UN	UN
UN	TOUT
TOUT	TOUT

La table TV serait

TR

R	O
V	I
F	O
S	O

TD

NOM = JULES*

TF

*
NOM = DI*

TL

I5

TQ

A
B

7) A = UN B = UN

rel	*	qualifiant	entité	filtre	demande
	I	tout	Bibliothèque		
	I	un	Gestionnaire		nom=Jules
	0	un	Département	nom=DI	

J = 3 SWN = 0 SW = I
SW* = 0 LD = 0
SW*T = 0 K2 = 0

```
I7  ≠
I8  M YI = 0
I9  I2 = I
    K = I
20  TTR (I)
2I  OUI
23  SW* = I
24  NON
28  POUR TOUT BIBLIOTHEQUE XI
29  K = 2
30  NON
34  ≠
35  I2 = 2
20  TTR (2)
2I  OUI
23  SW* = I
24  =
26  SW*T = I
28  POUR TOUT GESTIONNAIRE X2
29  K = 3
30  OUI
    KI = 2
3I  ≠
32  LD = 2
    SWN = I
33  =
36  NON
38  OUI
39  =
43  ≠
```

```
46  SI EXISTE UN DEPARTEMENT AYANT NOM = DI ;
|         ALORS SI NOM DE X2 = JULES
|                    ALORS M YI = I
|SW T = I      SORT
|                    SINON M YI = 0
|                    FIN

53| FIN  FIN  FIN
55|SI YI = I  ALORS I "OUI"
              SINON I "NON"
           FIN
```

```
M YI = 0
  POUR TOUT BIBLIOTHEQUE XI
  POUR TOUT GESTIONNAIRE X2
       SI EXISTE UN DEPARTEMENT AYANT NOM = DI ;
          ALORS SI NOM DE X2 = JULES
              ALORS M YI = I
                   SORT
              SINON M YI = 0
  FIN  FIN    FIN  FIN
SI YI = I ALORS I "OUI"
          SINON I "NON"
  FIN ?
```

8) A = UN B = TOUT

ref	*	qualifiant	entité	filtre	demande
	I	tout	Bibliothèque		
	I	un	Gestionnaire		nom=Jules
	0	tout	Département	nom=DI	

SW = I SWN = 0 LO = 0
S = 3 SW* = 0 K2 = 0
SW*T = 0

```
I7  ≠
I8 | M YI = 0
I9  I2 = I
    K = I
20  TTR (I)
2I  OUI
23  SW* = I
24  NON
28 | POUR TOUT BIBLIOTHEQUE XI
29  K = 2
30  NON
34  ≠
35  I2 = 2
20  TTR (2)
2I  OUI
23  SW* = I
24  =
26  SW*T = I
28 | POUR TOUT GESTIONNAIRE X2
29  K = 3
30  OUI
    KI = 2
3I  ≠
32  LD = 2
    SWN = I
33  =
36  NON
38  NON
48  =
```

```
SW ≠ 0
AI |POUR TOUT DEPARTEMENT AYANT NOM = DI ;
   |    SI NOM DE X2 DE XI DE XO = JULES
   |       ALORS M YI = I
   |       SINON M YI = 0
SW*T = I
 K3 = 3 - 2 = I
     ||SORTIE (I)
     ||FIN

     ||FIN
     ||SI YI = I ALORS SORT FINI

     ||FIN FIN

54   ≠
55   ||SI YI = I ALORS I "OUI"
     ||          SINON I "NON"
     ||   FIN

||M YI = 0
|POUR TOUT BIBLIOTHEQUE XI
|POUR TOUT GESTIONNAIRE X2
|POUR TOUT DEPARTEMENT AYANT NOM = DI ;
|          SI NOM DE X2 DE XI DE XO = JULES
|                 ALORS M YI = I
|                 SINON M YI = 0
|                    SORTIE (I)
|          FIN
|FIN
|SI YI = I  ALORS SORT FIN
|FIN FIN
|SI YI = I  ALORS I "OUI"
|           SINON I "NON"  FIN ?
```

SORTIE (I) nous ramène à Gestionnaire

SORT nous permet de sortir de toutes les boucles et nous ramène au dernier "SI".

9) A = TOUT B = TOUT

ref	*	qualifiant	entité	filtre	demande
	I	tout	Bibliothèque		
	I	tout	Gestionnaire		nom=Jules
	0	tout	Département	nom=DI	

SW = I SWN = 0 LD = 0
J = 3 SW* = 0 K2 = 0
SW*T = 0

I7 ≠ | M YI = 0
I9 I2 = I K = I
20 TTR (I)
2I OUI
23 SW* = I
24 NON
28 | POUR TOUTE BIBLIOTHEQUE XI
29 K = 2
30 NON
34 ≠
35 I2 = 2
20 TTR (2)
2I OUI
23 SW* = I
24 NON
28 | POUR TOUT GESTIONNAIRE X2
29 K = 3
30 OUI
KI = 2
3I ≠
32 LD = 2
SWN = I
33 =
36 NON
38 NON
SW ≠ 0

```
AI  POUR TOUT DEPARTEMENT AYANT NOM = DI ;
    SI NOM DE X2 DE XI DE XO = JULES
       ALORS M YI = I
       SINON M YI = O
SW*T ≠ I  SORT
          FIN
53 |  FIN FIN FIN
54  ≠
55 | SI YI = I ALORS I "OUI"
                SINON I "NON"
                    FIN
```

```
M YI = O
POUR TOUT BIBLIOTHEQUE XI
POUR TOUT GESTIONNAIRE X2
POUR TOUT DEPARTEMENT AYANT NOM = DI ;
      SI NOM DE X2 DE XI DE XO = JULES
         ALORS  M YI = I
         SINON  M YI = O
                SORT
     FIN
FIN  FIN  FIN
SI YI = I  ALORS I "OUI"
           SINON I "NON" FIN ?
```

IO) Vérifier si le nom du gestionnaire comptabilisant tout matériel de nom MI est Jules.

Pour cet exemple, nous ajoutons une relation.

27 : Gestionnaire - Matériel, dont l'accès est

*Bibliothèque, *Gestionnaire, *Département, Matériel

et dont le nom est comptabilisant.

TR

R	O
V	I
F	O
S	O

TD

NOM = JULES *

TF

*
NOM = MI*

TL

27

TQ

UN
TOUT

La table TTR se construit comme vu avant.

ref	*	qualifiant	entité	filtre	demande
	I	tout	Bibliothèque		
	I	un	Gestionnaire		nom=Jules
	I	tout	Département		
	O	tout	Matériel	nom=MI	

SW = I

J = 4

SWN = O

SW* = O

SW*T = O

LD = O

K2 = O

Tout ceci est obtenu au cours de la construction de TTR.

```
I7  ≠
I8 |  M YI = 0
I9  I2 = I  K = I
20  TTR (I)
2I  OUI
23  SW* = I
24  NON
28 |  POUR TOUT BIBLIOTHEQUE XI
29  K = 2
30  NON
34  NON
35  I = 2
20  TTR (2)
2I  OUI
23  SW* = I
24  OUI
26  SW*T = I
28 |  POUR TOUT GESTIONNAIRE X2
29  K = 3
30  OUI
    KI = 2
3I  ≠
    LD = 2
33  ≠
    K2 = I
35  I = 3
20  TTR (3)
2I  OUI
23  SW* = I
24  NON
28 |  POUR TOUT DEPARTEMENT X 3
29  K = 4
30  NON
34  =
35a =
 W = R (36) SAUF HIERARCHISATION
36  NON
38  NON
48  ≠
SW ::0 ≠ ⟹ AI
AI |  POUR TOUT MATERIEL AYANT NOM = MI
   |     SI NOM DE X2 DE XI DE XO = JULES
   |     ALORS M YI = I
   |     SINON M YI = 0
SW T::I =
     K3 = 2
   |  SORTIR (2)
   |   FIN
   |   FIN  FIN SI YI = I ALORS SORT FIN
   |   FIN FIN
|SI YI = I  ALORS I "OUI"
            SINON I "NON"
                    FIN
```

```
M YI = 0
POUR TOUT BIBLIOTHEQUE XI
POUR TOUT GESTIONNAIRE X2
POUR TOUT DEPARTEMENT X3
POUR TOUT MATERIEL AYANT NOM = MI ;
     SI NOM DE X2 DE XI DE XO = JULES
        ALORS M YI = I
        SINON M YI = 0
              SORTIE (2)
     FIN
     FIN
FIN
SI YI = I ALORS SORT FIN
FIN
FIN
SI YI = I ALORS I "OUI"
          SINON I " NON"
FIN ?
```

6.9. Combinaison de relations de la "table vue utilisateur".

Il est à remarquer que LN et LR permettent des combinaisons de relations appartenant à la "table vue utilisateur". Cela nous a permis de donner ces deux langages avec une certaine richesse d'interrogation. De plus, ils sont indépendants, dans leur définition, du langage terminal choisi. Il n'en sera pas de même dans leur programmation dans un cas précis.

En effet, certains langages terminaux permettront de réaliser des groupements que d'autres n'autoriseront pas, ceci évidemment dans le cas où on ne désire pas construire un traducteur trop complexe.

Ici, nous avons choisi Socrate comme langage terminal, et dans le cas d'une traduction automatique nous serons amenés à imposer certaines restrictions de façon à ne pas entraîner une phase de traduction trop complexe.

Nous avons déjà pu constater que la génération en Socrate n'est pas très simple même lorsque les requêtes nous semblent peu compliquées Notons que cette difficulté n'est pas uniquement due à la syntaxe de Socrate, mais en bonne partie au fait que nous voulons générer automatiquement ce langage.

A) Jusqu'à présent nous avons parlé de relations appartenant à la"table vue utilisateur" Il est certain que toutes les questions que l'on veut poser à la data base doivent se construire à partir de ces relations. Il serait même très intéressant que toutes les requêtes soient représentées dans cette table. Cela devrait être possible car la structure de la data base doit être construite en fonction des problèmes que l'utilisateur veut résoudre, et il serait souhaitable que celui-ci puisse poser des questions très simples. Cette table peut évidemment s'enrichir au cours de la vie du système, et n'est nullement limitée au contenu stocké lors de sa réalisation initiale.

Il est fort possible que les questions jugées nécessaires au départ se révèlent trop peu nombreuses après un certain laps de temps. A ce moment, la structure peut très bien ne plus être valable pour permettre une génération simple de ces nouvelles questions. Cela provient essentiellement du fait que Socrate est très exigeant de par sa syntaxe. La nécessité d'hiérarchiser une citation, l'appartenance

obligatoire des boucles imbriquées à celles qui leur sont supérieures, la portée de démonstratif Xi rendent très compliquée une génération automatique pour des requêtes apparemment simples, du moins en ce qui concerne la détermination du chemin d'accès. Cela est renforcé par l'ambiguîté des quantificateurs UN et TOUT qui sont les seuls quantifiants permis. Il y a aussi PREMIER et DERNIER, mais comme l'implémentation d'une entité détermine un ordre répondant à la séquence d'arrivée des occurences, et non à un ordre sémantique, ces deux quantifiants ont une portée assez limitée.

Bien su'il soit préférable de mettre toutes les relations permises dans la table, et d'essayer d'éviter au maximum les combinaisons de ces relations, nous allons quand même analyser succinctement la façon de générer Socrate pour certaines de ces combinaisons. Nous en ferons une description sans entrer dans les détails d'un organigramme. Nous avons déjà pu nous rendre compte, grâce aux deux paragraphes précédents, que la génération Socrate n'est pas toujours facile.

Comment allons-nous représenter une relation entre plusieurs entités si nous voulons la stocker dans la table? Nous pouvons facilement donner un nom à une relation unissant deux entités, mais cela n'est plus aussi simple si plusieurs entités sont concernées par la relation.

Le meilleur moyen pour enregistrer de telles relations,serait d'utiliser la macro. On pourrait ainsi réaliser "manuellement" une série de macros représentant des relations complexes à générer, et supprimer ainsi la difficulté de la génération automatique.

Prenons un exemple :

Si nous avons une macro MI, représentant une relation entre Bibliothèque et Département, relation dans laquelle la question porte sur département (cette précision est très importante au niveau de la programmation en Socrate). Il faudra pour chaque macro fixer le nombre et le sens de chacun des paramètres.

L'appel pourrait se présenter comme suit :

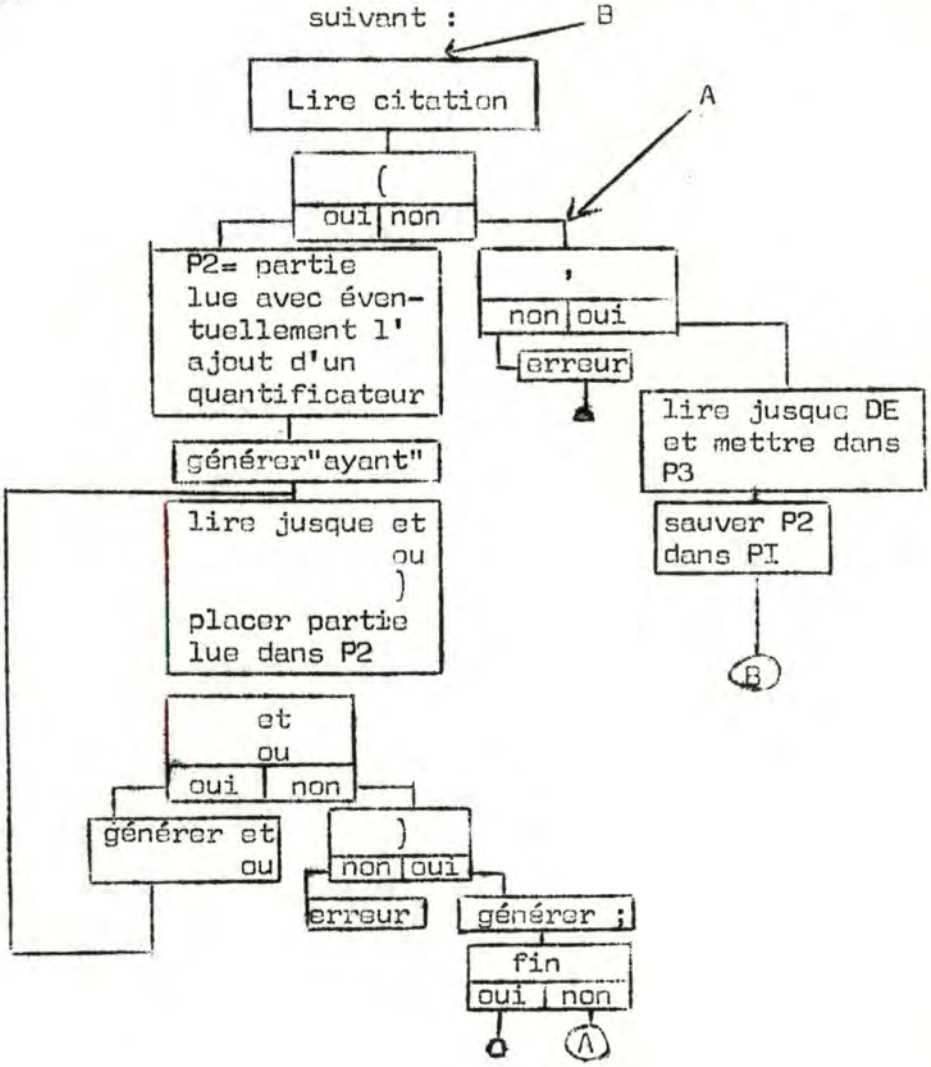
MI : une Bibliothèque (nom = BI),
nom de tout département (code = 8)

et on aurait en Socrate

```
! DEFMAC  SMI ! ! !
! EXP     Pour ! I !
          Pour tout gestionnaire
          Pour ! 2 !
            I ! 3 !
          fin  fin  fin
! FDEF ! ?
```

La macro MI aurait pour unique but de présenter les paramètres correctement pour Socrate, et d'appeler SMI :

appel de SMI : si une Bibliothèque ayant nom = BI; <=> PI
tout Département ayant code = 8; <=> P2
nom <=> P3
sera SMI PI P2 P3 ?
et la macro MI est décrite par l'organigramme suivant :



On a admis qu'il y avait au moins un filtre sur chacune des deux entités. L'appel à SMI sera SMI PI P2 P3 et l'on obriendra

```
Pour une bibliothèque ayant nom = BI ;
Pour tout gestionnaire
Pour tout département ayant code = 8 ;
    I  nom
Fin  Fin  Fin  ?
```

B) Donnons maintenant une autre méthode qui permettra d'accepter certaines combinaisons de relations. Deux cas vont être envisagés :

I. Une des relations est la réunion des autres.

Si nous avons trois relations dont les accès sont représentés dans la "table vue utilisateur" par

```
C, B, A
C, B
B, A
```

nous dirons que la Iere relation est la réunion des 2 autres relativement à leurs accès.

Soit nom et code de tout matériel appartenant à BI et comptabilisé par Jean de code 4.

Nous aurons, comme décrit précédemment,

TR

R	I
V	0
F	0
S	0

TD

nom et
code *

TF

nom = BI *
nom = Jean
et code = 4*

TL

2I	dont
23	de

LQ

tout
un
un

La relation 2I a pour accès

Bibliothèque, gestionnaire, département,matériel

La relation 23 a pour accès

Bibliothèque, gestionnaire, département,matériel

Une relation est bien la réunion des 2 et nous allons construire la table TTR identique à celle décrite dans le paragraphe deux, en se basant sur l'accès de la relation union des autres.

T T R

référence	*	qualifiant	entité	filtre	demande
	0	un	bibliothèque	nom = BI	
	0	un	gestionnaire	nom = Jean et code =4	
	0	tout	département		
	0	tout	matériel		nom et code

Remarquons que lorsque l'on dit q'une relation est l'union des autres, il faut ce que nous appelerons une union complète.

*A	*B	*C	pas d'union possible
A	B	C	

Cela est dû en fait que la programmation ne sera pas la même dans les deux cas, et il est bien évident qu'ici nous devons tenir compte de Socrate.

La table TTRpeut être exploitée par les organigrammes décrits dans le paragraphe 2. La relation combinée n'aura pas à être vérifiée par rapport à un poids quelconque, puisqu'elle est de toute façon reprise dans la "table vue utilisateur".

Il est à remarquer que de telles relations sont relatives à une même branche de l'arbre qui pourrait représenter la structure de la data base.

De telles relations sont encore facilement réalisables et ne seront évidemment pas reprises par des macros.

L'organigramme UNEREL nous permet d'exploiter FTR.Sa génération de TV pourrait être effectuée par UNEREL moyennant quelques ajouts pour la recherche des quantificateurs. A la fin de cette construction de table nous avons : J = 4 SWN = 0 LD = 0

SW = 0 SW* = 0 K2 = 0

SW*T = 0

```
I7   =
I9   I2 = I      = I
20   TTR ( I )
2I   non
22   || Pour une bibliothèque XI ayant nom = BI ;
29   K = 2
30   non
34   ≠
35   I2 = 2
20   TTR ( 2 )
2I   non
22   || Pour un gestionnaire X2 ayant nom = Jean et code = 4 ;
29   K = 3
30   non
34   ≠
35   I2 = 3
20   TTR ( 3 )
2I   non
22   || Pour tout département X3
29   K = 4
30   non
34   =
35 a ≠
36   non
38   non
48   non
50   =
5I   || Pour tout matériel I nom I code
53   || fin  fin  fin  fin
54   =
```

Si on rassemble les générations ,

```
Pour une bibliothèque X1 ayant nom = BI ;
Pour un gestionnaire X2 ayant nom = Jean et code = 4 ;
Pour tout département X3
Pour tout matériel
     I nom
     I code
fin  fin  fin  fin ?
```

2. Les relations sont indépendantes relativement à leur accès

Dans de telles relations, on ne sait pas dire à priori si elles sont facilement réalisables ou pas. En effet, nous avons donné dans la table des accès en prenant le plus intéressant pour la relation, mais il en existe d'autres qui seraient peut-être bons dans des cas où ceux de la table ne conviennent pas. Le fait que les accès de la "table vue utilisateur" ne soient pas compatibles pour une série de relations ne veut donc pas dire qu'il n'est pas possible de générer l'équivalent Socrate.

Nous allons donc essayer de trouver un tel chemin, et c'est là, la dernière étape qui sera examinée dans le passage LR - Socrate.

Nous exposerons cela en nous servant d'un exemple.

Prenons la requête,

Quel est l'ouvrage loué par BI et emprunté par Jean?

La relation "loué par" a pour accès

Bibliothèque, ouvrage (I)

La relation "emprunté par" a pour accès

Emprunteur, ouvrage (ouvrage) (2)

Remarquons que les entités ouvrages signalées dans ces deux accès ne sont pas les mêmes. Cela se constate facilement par le fait que les imbrications supposées par ces accès ne sont pas les mêmes.

En effet (I) = Bibliothèque
↓
Ouvrage

(2) = Emprunteur
↓
Ouvrage ⟵ réference ouvrage

A priori, en examinant uniquement ces accès, on ne peut pas déterminer un chemin commun. On va donc essayer d'en trouver un au moyen de la "table des liens entre entités".

La solution la plus simple serait d'essayer de placer l'entité concernée par la demande, en fin de chemin d'accès. Cela nous évitera des ennuis lorsque dans une suite de boucles imbriquées, on veut un résultat sur l'entité de la première en fonction d'une condition sur celle de la dernière.

Nous allons donc rechercher un chemin qui va nous conduire à OUVRAGE et qui mettra cette entité en rapport avec EMPRUNTEUR et BIBLIOTHEQUE.

Nous consulterons pour cela la "table des liens entre entités".

I. - ouvrage est accessible par Bibliothèque
Bibliothèque, Ouvrage

regardons si emprunteur peut satisfaire cet accès

- emprunteur est accessible par Bibliothèque, ouvrage (emprunteur)

cela est possible donc,

Bibliothèque, Ouvrage (emprunteur)

Voyons s'il n'y en a pas d'autres, auquel cas on choisirait le meilleur.

2. - ouvrage : Auteur, ouvrage (ouvrage)
NON
car n'a ni emprunteur, ni bibliothèque

3. - ouvrage, Emprunteur, ouvrage (ouvrage)
regardons si bibliothèque peut aller avec cela,

- Bibliothèque : Emprunteur, Bibliothèque (bibliothèque)
NON

4. - ouvrage : Bibliothèque, ouvrage, bibliographie (ouvrage)
regardons si Emprunteur peut se réaliser par un accès combinable avec celui-ci
NON

On accepte donc

Bibliothèque, Ouvrage (emprunteur)

On se rend compte que ce choix peut être assez long et est relativement arbitraire puisqu'on décide de placer l'entité concernée par la demande en fin du chemin d'accès.

La table TV correspondante peut être construite par UNEREL modifiée et exploitée par UNEREL.

référence	*	qualifiant	entité	filtre	demande
	0	UN	bibliothèque	nom = BI	
	0	UN	ouvrage	nom de emprunteur = Jean	titre

Comme l'accès total est déduit des différents accès partiels, il faut remarquer que la génération à partir de la table est plus complexe.

On ne peut plus affirmer que, si on a une référence, la ⟨demande⟩ devient ⟨demande de référence⟩ comme dans UNEREL.

Il faudrait construire la table différemment en plaçant emprunteur non plus comme référence mais comme filtre. Ce placement peut se déterminer en vérifiant à quelle entité appartient la caractéristique demandée.

ici titre appartient à ouvrage et non à emprunteur
s'il y avait une confusion, on pourrait s'en tirer par le conversationnel en demandant à l'utilisateur de préciser quelle entité l'intéresse.

L'organigramme UNEREL peut exploiter cette table

I ⟶ I6 : J = 2 SWN = 0 SW*T = 0 K2 = 0
SW = 0 SW* = 0 LD = 0

I7 =
I9 I = I K = I
20 TTR(I)
2I non
22 || Pour une bibliothèque ayant nom = BI ;
29 K = 2
30 NON
34 =

```
35 a ≠
38   OUI
39   ≠
40   =
4I   || Pour un ouvrage ayant nom de emprunteur = Jean ;
                                                  I titre
53   || fin  fin
```

ce qui donne

```
||Pour une bibliothèque ayant nom = BI ;
||Pour un ouvrage ayant nom de emprunteur = Jean ;
  I titre
||fin  fin ?
```

Nous pouvons résumer le paragraphe comme suit:

1. La requête fait intervenir une entité auquel cas elle est générée par UNENTITE
2. La requête fait intervenir une relation, et est alors générée par UNEREL
3. La requête fait intervenir une fonction. On a donc un appel de macro avec arrangement des paramètres pour leur donner une présentation Socrate.
4. La requête fait intervenir plusieurs relations et une est l'union des autres. C'est UNEREL légèrement modifiée qui traiterait ce cas.
5. La requête fait intervenir plusieurs relations dont aucune n'est l'union des autres. On recherche alors un chemin d'accès reliant les entités concernées.

Rappelons que les relations complexes seraient exprimées par une macro. Les requêtes qui seraient refusées par une telle analyse ne sont pas non programmables en Socrate, même par génération automatique, mais l'analyseur permettant de réaliser le passage LR - Socrate serait très complexe.

7. Conclusion.

Nous venons de définir un langage orienté naturel et nous l'avons traduit, par l'intermédiaire d'un langage rigide LR, en Socrate.

Notre hypothèse de travail consistait à donner des noms à toutes les relations et de les imposer à l'utilisateur. Cela nous a permis de lever assez facilement quelques ambiguïtés qu'on retrouve dans la plupart des phrases françaises.

Nous avons travaillé essentiellement avec des tables de travail telles que TV, TTR et des tables permanentes telles les tables des fonctions, des synonymes fonctionnels, des quantificateurs.

Il est certain que le langage naturel défini ici peut être amélioré. On le rendrait plus souple si nous introduisions une table des synonymes relationnels qui permettrait des formulations du genre "Emprunts de Jean" au lieu de "Ouvrages empruntés par Jean". Nous avons aussi souligné le fait que l'attribution d'une fonction de poids à chaque relation devait être une des premières optimisations de ce langage. Il faudrait ensuite supprimer les limites que nous avons apportées :

- pas de relations filtrantes pour des relations caractérisant déjà une demande, si ce n'est une suite de relations successives dans la "structure vue utilisateur".
- pas plusieurs valeurs, reliées par ET, pour une même caractéristique.

Un point très important est à ne pas oublier: il s'agit de la structure de définition Socrate. Pour toute application, il est nécessaire que cette structure soit bien étudiée de façon à permettre une formulation simple des requêtes. Il est indispensable de construire une structure valable, sans quoi les requêtes sont difficiles à rédiger.

Nous avons analysé cinq présentations possibles pour la requête LN, et elles sont reprises en fin du § 6.9. Le cas le plus courant devrait être celui pour lequel les requêtes sont formulées avec une relation.

Si la structure est bien pensée, cela devra se présenter souvent. Des relations non prévues au départ peuvent devenir indispensables. Dans ce cas, il faudra les entrer dans la "table vue utilisateur". Cependant, nous considérons aussi des requêtes relatives à des entités situées sur une même branche si nous donnons une forme d'arbre à la structure Socrate.

Il nous faut encore souligner que dans ce travail, nous sommes restés à l'aspect analyse du problème, sans nous occuper de l'aspect implémentation. Si nous parlons de tables, c'est pour visualiser les éléments qui nous sont nécessaires, et cela ne signifie pas que ces renseignements seront effectivement stockés dans des tables.

Rappelons aussi que nous avons généré les requêtes en Socrate en essayant de donner une formulation standard. Cela nous a permis de réaliser un "traducteur" relativement simple. Il est certain qu'un mécanisme plus poussé étudiant de plus près la forme de la requête donnerait un résultat plus performant. Un tel analyseur sort du cadre de ce travail, du moins quant au but que nous nous étions fixé.

En effet, notre problème était de déterminer une structure intermédiaire qui nous permettrait de définir un LN indépendamment de tout langage rigide. Ensuite, il nous fallait montrer que, pour un langage terminal, la structure intermédiaire, à laquelle on ajoutait un renseignement relatif à l'accès, permettait un passage à ce langage terminal.

Nous avons atteint notre objectif en prenant Socrate pour langage terminal. Certes, nous avons imposé certaines limites aux différents langages quant à leur utilisation, mais comme nous l'avons signalé, les lever n'est pas très compliqué.

C O N C L U S I O N .

Deux sujets ont été traités dans ce travail :

- comparaison du langage naturel proposé par W.A.Woods et du langage Socrate.
- consultation d'une data base au moyen d'un langage orienté naturel.

Dans la première partie, nous avons comparé un langage naturel et un langage rigide en essayant de faire ressortir leurs avantages et leurs inconvénients. Nous y avons nettement remarqué que Socrate était plus intéressant que le langage de Woods tout au moins du point de vue réalisateur. Quant à l'utilisateur, un langage naturel est évidemment plus simple et moins exigeant.

Nous avons pu constater que Woods donnait un langage attaché à la base de données quant à son contenu. En effet, les primitives relevées dans ce langage expriment directement des relations entre des éléments de la base.

Dans la seconde partie, nous avons défini un langage naturel LN dont les primitives sont des mots de commande valables pour toute base de données. Les noms de relations, les tables de fonctions, de caractéristiques,...sont évidemment liés à l'application, mais le langage lui-même n'est pas dédicacé. Notre langage est peut-être plus rigide de par ses noms de relation, mais rappelons qu'il est possible de donner à LN une formulation plus souple en construisant par exemple un dictionnaire de synonymes relationnels.

Nous ne voulons pas nous prononcer sur l'intérêt d'un langage naturel sur un langage rigide ou l'inverse. Cela dépend essentiellement de ce que l'on attend du système et du personnel que l'on compte faire travailler avec un tel système. Cela n'est de toute façon pas le but de ce travail, et demanderait une analyse plus poussée afin de porter un jugement valable.

Addenda.

Pour des contraintes d'impression,il nous a été impossible de donner la version définitive de l'introduction de ce mémoire.Les quelques notes qui vont suivre sont à considérer comme remplaçant les pages un à quatre de ce travail.

Nous espérons que les membres du jury accepteront cette présentation peu rigoureuse,et d'avance,nous les en remercions.

Bibliographie.

1. Semantic for a Question-Answering System.
   Thèse de William A.Woods. Septembre 1967.
   Harvard University.
   Cambridge,Massachusetts.
2. Le Projet Socrate.
   J.R.Abrial - J.Bas - G.Beaume - G.Henneron - R.Morin - G.Vigliano.
   Avril 1970.
   Université de Grenoble,Institut de Mathématiques appliquées.
3. Generalized File Processing.
   William C.Mc Gee. 1969.
   Annual Review in Automatic Programming,vol. 5,part 1.
   IBM Corporation,Palo Alto,California.
3. Data Structures and their representations in Storage.
   M.E.D'Império.
   Annual Review in Automatic Programming,vol. 5,part 2.
   Department of Defense,Washington.
5. On the Role of Natural Language in Man-Machine Communication.
   J.Bruce Fraser.
6. Procedural Semantic for a Question-Answering Machine.
   William A. Woods.
   Fall Joint Computer Conference. 1968.
   Harvard University.
   Cambridge,Massachusetts.
7. A Relational Model of Data for Large Shared Data Bank.
   E. F. Codd.
   IBM Research Laboratory.
   San Jose,California.

8. Answering Questions by Computer:a Logical Study.
J. L. Kuhns.
The Rand Corporation.
9. Data Structure,Theory and Practice.
A. T. Berztiss.
University of Pittsburgh.
Academic Press.
New York and London.
10. Journées "Banques de Données".
AFCET-IRIA. Juin 1971.
Aix-en-Provence.
11. Codasyl Data Base Task Group.
Rapport d'avril 1971.
12. An Introduction to Data Base Design.
John K. Lyon.
Honeywell Information Systems,Inc.
Schenectady,New York.
13. A Survey of Generalized Data Base Management Systems.
Codasyl Systems Committee.
Technical Report. Mai 1969.
14. Feature Analysis of Generalized Data Base Management Systems.
Codasyl Systems Committee.
Technical Report. Mai 1971.

Sources de renseignements.

- Stage réalisé à Grenoble.Renseignements obtenus par l'équipe ayant participé à la réalisation de Socrate pour ECA-AUTOMATION.
- Travaux et réunions de l'équipe de recherche de l'Institut d'Informatique des Facultés Notre-Dame de la Paix de Namur.

Table des figures.

Application 53

Tableau descriptif de la base. 55

Table des caractéristiques. 56

Structure de définition. 57

Structure vue utilisateur et sa forme transposée. 68

Table vue utilisateur. 74

Table des liens entre entités. 78

Table des quantificateurs. 89

Table des caractéristiques virtuelles. 90

Table des fonctions. 90

Table des synonymes fonctionnels. 90

Table de travail TV,composée de TR 92
TD
TF
TQ
TL.

Table de travail TTR,composée de référence 136
astérisque
entité
filtre
demande.

Introduction.

A une époque où les banques de données s'avèrent de plus en plus indispensables,il nous a paru intéressant de nous attarder à ce nouveau concept informatique.

L'évolution en informatique est telle que l'on tend vers une utilisation de l'ordinateur par des personnes qui n'en connaîtront pas, et n'auront pas à en connaître, ni le fonctionnement ni les techniques actuelles de son utilisation.Les systèmes d'exploitation classique étant incapables de satisfaire ce besoin,de nouveaux type de systèmes sont apparus:les banques de données.La mission essentielle d'un tel système consiste à recueillir et à analyser la masse des informations pour les mettre à la disposition des utilisateurs.

Du point de vue langage,nous pouvons établir une distinction assez nette entre deux types:les langages de désignation,de modification de données et de structures,et les langages de traitement,tel COBOL.Au niveau des banques de données,par définition de leur but,il est certain que le premier type de langage est le plus important.Si nous voulons passer au stade du traitement,il suffit de considérer les langages de désignation,de modification comme hôtes des langages de traitement. (bibli. 11,14).C'est à ce premier type de langage que devraient se porter les intérêts des équipes de recherche travaillant sur ce sujet.

Pourquoi redéfinir de nouveaux langages de désignation et de modification?Cela est lié au fait que les banques de données doivent être manipulées par des non-informaticiens.Or les langages existant sont réservés à des programmeurs,et demandent aux utilisateurs une certaine logique qui est différente de la leur.

Il est aussi nécessaire que ces langages soient prévus pour permettre une réelle conversation entre l'utilisateur et le système.Les langages actuels ne sont pas adaptés à ce mode de traitement qui est

primordial pour les banques de données.La banque de données doit en effet mettre à la disposition de l'utilisateur des informations en vue d'apporter une aide à la prise de décision.Cela se retrouve dans des applications de gestion de réservation de place,médicale,administrative...

Nous venons de mettre en évidence quelques propriétés fondamentales d'un langage de consultation de banques de données.Nous voulons donc qu'un tel langage soit simple car il sera mis à la disposition de non-informaticiens.Il doit être défini en fonction de ce que l'utilisateur veut exprimer et avec le but de lui faciliter la tâche.Dans le cas contraire,on retombe inévitablement sur un langage de programmation.

L'aspect conversationnel est lui aussi très important.Le minimum que l'on peut exiger d'un tel langage est qu'il permette des procédures questions-réponses relativement au contenu de la base.Il est aussi intéressant de prévoir une aide à l'utilisateur quant à la formulation de sa requête.Cet aspect peut comprendre la construction d'une question entièrement guidée par le système,ou plus simplement la possibilité de permettre à l'utilisateur de ne pas rédiger des requêtes complètes,c'est-à-dire reprenant tous les noms définis dans la structure pour préciser une question.Par exemple,il doit pouvoir demander "Quels sont les ouvrages écrits par Proust" sans devoir préciser que Proust est le nom de l'auteur de ces ouvrages.

Une caractéristique essentielle de ces langages est aussi leur aspect naturel.En effet,si nous voulons permettre à de non-programmeurs d'utiliser facilement une data base,il est nécessaire de leur fournir un outil convenable.Un langage naturel est assez facile en ce sens qu'il n'oblige pas l'utilisateur à travailler avec une syntaxe inhabituelle, selon une logique particulière.

Nous opposons la notion de langage naturel à celle de langage rigide. Un tel langage est orienté programmation.Il y a évidemment plusieurs degrés de rigidité d'un langage,et un langage naturel peut se définir comme étant le moins rigide possible.

Dans ce travail,nous nous attachons tout d'abord à une comparaison entre un langage naturel et un langage rigide.Ensuite nous définissons

un langage naturel et établissons une méthode de passage de ce langage naturel à un langage rigide.

Avant d'introduire le travail réalisé,il nous faut faire remarquer que nous nous intéressons uniquement au langage de désignation.Nous avons abusivement employé le terme "consultation"pour parler de désignation.Cela provient du fait que nous illustrons cette analyse par quelques exemples d'interrogation de banques de données.

Il nous a semblé intéressant d'analyser deux systèmes de gestion de ces nouveaux " êtres" informatiques.Il s'agit d'un langage orienté naturel proposé par Woods (bibli. 1,6),et d'un système complet,dont le langage peut être qualifié de rigide,de nom Socrate (bibli. 2).Woods propose un langage naturel subset de l'anglais tandis que Socrate est un langage plus rigide ne pouvant être manipulé qu'après une période d'adaptation relativement longue.

Le choix de Socrate nous a été dicté par un stage réalisé à Grenoble,tandis que nous avons découvert Woods à travers un de ses articles et sa thèse (biblio. 1).

Cette comparaison définit la première partie de ce travail.Nous présentons succinctement chacun des deux softwares puis nous les comparons en nous limitant à l'aspect interrogation de la base,seul point traité par Woods.Notons que Socrate permet de réaliser d'autres opérations sur une banque de données.Il s'agit de la définition,création et mise à jour de la base.

Nous analysons principalement l'influence que peut avoir le contenu de la data base sur chacun des deux systèmes,ainsi que les liens entre les langages et ce contenu.Nous pourrons constater que le langage naturel est fortement lié aux informations de la data base,tandis que Socrate en est indépendant.

Woods ne donne aucun renseignements quant à l'accès à la banque de données.Nous nous sommes alors demandés s'il n'était pas possible d'envisager une méthode telle celle employée par Socrate,ou au moins prévoir un système libérant l'utilisateur de tout problème d'accès.

Le résultat de l'analyse sémantique de Woods donne un appel de macros qui peut définir un langage rigide.Il est intéressant de voir si on peut remplacer ce dernier par Socrate,et d'analyser le passage d'un langage naturel à Socrate.De cette façon,on complète le système proposé par Woods au moyen de Socrate,et on enrichit Socrate d'un langage orienté naturel.Nous obtenons ainsi le schéma suivant:

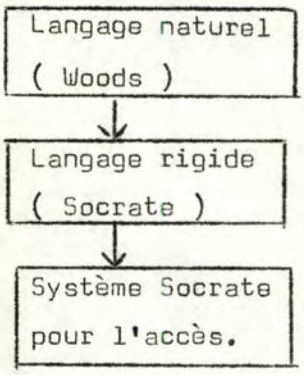


Schéma 1.

Il est certain que le problème que nous analysons ici sera étudié en prenant comme langage naturel un sous-ensemble du français.

Nous venons de parler de langage naturel.Il est certain qu'il est intéressant de permettre à un utilisateur d'interroger une data base dans un langage proche de sa langue,c'est-à-dire ayant un vocabulaire et une syntaxe à peu près identique.Il ne faut pas penser qu'un langage naturel laisse aux utilisateurs la possibilité d'interroger la base en toute liberté.Il y a un certain vocabulaire admis et une syntaxe nécessaire,mais la formulation d'une requête donne une phrase naturelle.Le langage est naturel en ce sens que l'on obtiendrait un langage à peu près identique si on regroupait une série de phrases posées par des utilisateurs dans le cadre d'une application bien déterminée,cette dernière n'étant pas encore informatique,mais manuelle.

La comparaison Woods-Socrate nous a permis de soulever le problème représenté par le schéma 1.Nous allons nous attacher,dans une seconde partie à la solution de ce problème,en le généralisant un peu,et obtenant ainsi un problème découlant toujours du schéma 1,mais en rapport plus étroit avec des travaux réalisés à l'Institut d'Informatique.

En effet,une équipe de recherche est chargée de définir un système de gestion de banque de données en collaboration avec d'autres facultés. Dans le cadre de ce projet,Namur doit réaliser un langage naturel de gestion de data base,mais la structure des données et le langage rigide soutenant ce langage naturel sont définis par d'autres personnes.Il faut donc trouver un moyen permettant de définir ce langage naturel sans avoir connaissance des autres éléments.C'est un essai de solution de ce problème que nous allons établir ici.Nous nommerons le langage naturel, sous-ensemble du français,par LN.Nous pouvons donner une représentation de ce travail par le schéma suivant:

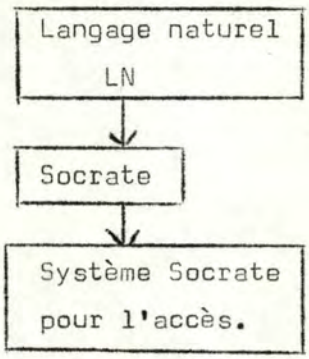


Schéma 2.

Il nous faut donc trouver une étape intermédiaire permettant de définir le langage naturel sans connaître la structure des données et le langage rigide.Il doit donc être indépendant de tout langage rigide.

Nous savons que tout langage de consultation de data base travaille sur des données ayant une forme bien déterminée.On définit ainsi une structure propre à un système.Si nous associons à chaque langage la structure qu'il exploite,en nommant SN celle associée à LN et

SR celle correspondant à Socrate,nous obtenons le schéma suivant dans lequel nous ne représentons plus l'accès qui est compris dans Socrate:

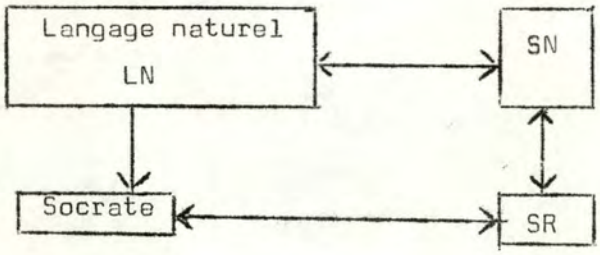


Schéma 3.

Nous avons parlé de structure SN associée à LN.Ce n'est pas une structure au sens classique.En effet,on n'a aucune notion d'accès attachée au langage naturel,et SN ne servira qu'à donner à l'utilisateur l'ensemble des relations logiques existant entre les différents éléments de la data base.Nous pouvons dire que SN est simplement un tableau représentant tous les chemins d'accès reliant les entités du fichier.Nous avons choisi de garder la notion d'entité,c'est-à-dire de bloc formé de plusieurs caractéristiques élémentaires et qui peut avoir plusieurs réalisations.Cette notion d'entité et de caractéristique est définie dans ce travail en pages 8 et 9.

Il est certain que dans l'ensemble (structure,langage),la structure est l'élément principal.Ayant une structure,on définit un langage capable de l'exploiter.Il est donc normal que pour résoudre notre problème on s'intéresse à la détermination d'une structure intermédiaire SI indépendante de toute structure associée à un langage rigide.Nous obtenons le schéma suivant:

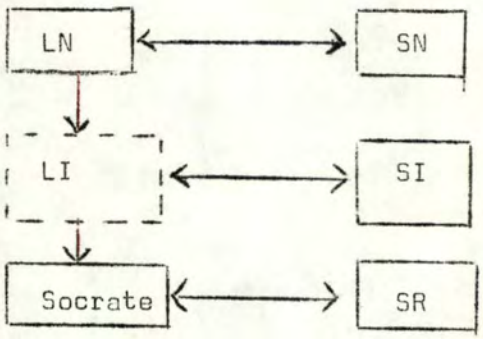


Schéma 4.

Pour respecter la symétrie ensemble des langages,ensemble des structures,nous définissons un langage LI qui est associé à SI.Il est à remarquer que l'élément qui nous intéresse est la structure et non le langage. Ce dernier n'est cependant pas inutile.En effet,si nous le définissons assez proche de LN,il peut être utilisé par des utilisateurs déja familiarisés avec l'informatique.Il est donc intéressant de prévoir l'analyse de LN et sa transformation en Socrate d'une manière telle que l'on puisse facilement insérer LI en intermédiaire.

Cette optique est adoptée dans ce travail.Nous devons donc définir SI assez proche de SN, c'est-à-dire que SI contiendra entre autres les renseignements donnés par SN.Nous pouvons représenter cela comme suit:

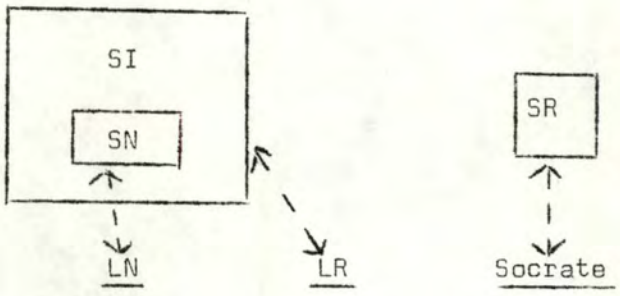


Schéma 5

En fait,SN n'existe que par l'intermédiaire de SI.Nous l'avons définie uniquement parce que l'utilisateur ne possédera pour formuler sa requête qu'une partie de la structure SI,qui est la structure SN.

La structure SI va donc contenir les renseignements donnés par SN,à savoir:noms de relations,nom de l'entité origine de la relation, nom de l'entité sommet de la relation,type de relation,notion d'ordre. Nous lui ajoutons un renseignement relatif à l'accès,c'est-à-dire une description du chemin à parcourir dans la stucture Socrate pour exécuter la relation.C'est le seul lien qui existe entre la structure SI et la structure Socrate.

Nous avons donc défini une structure SI indépendante du langage rigide choisi,sauf évidemment en ce qui concerne l'accès.Ce renseignement ne nous est indispensable qu'au dernier stade de la réalisation du travail demandé,c'est-à-dire au moment de faire le lien avec le langage rigide et sa structure.