50376
1986
97

T H E S E

PRÉSENTÉE À

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

POUR OBTENIR LE TITRE

DE

DOCTEUR EN INFORMATIQUE

PAR

BERNARD LEFEBVRE

systeme expert en

IDENTIFICATION BACTERIENNE

DEFINITION ET MISE EN APPLICATION

D' UNE

INTERFACE PROLOG - SGBD

SOUTENUE LE 1ER JUILLET 1986, DEVANT LA COMMISSION D'EXAMEN

MEMBRES DU JURY :

M. CHRISTIAN CARREZ, PRÉSIDENT
M. HENRY FARRENY, RAPPORTEUR
M. GEORGES STAMON, RAPPORTEUR
M. CLAUDE LANGRAND, EXAMINATEUR
M. JEAN-PAUL DELAHAYE, EXAMINATEUR
M. HENRI LECLERC, INVITÉ
M. GÉRARD COMYN, DIRECTEUR DE THÈSE

Je remercie tout d'abord Monsieur le Professeur Christian CARREZ qui me fait l'honneur de présider ce Jury.

Je remercie les Professeurs Henry FARRENY et Georges STAMON qui, bien que très pris par leurs importantes responsabilités, ont accepté de me consacrer une part de leur temps et me font le grand honneur d'être les rapporteurs de cette thèse.

Je remercie le Professeur Gérard COMYN, Directeur de cette thèse, auprès de qui j'ai toujours trouvé le soutien amical et les conseils qui m'ont permis d'achever ce travail. Cette aide directe est cependant peu vis à vis de l'exemple de générosité dans l'effort, d'esprit d'entreprise et de dynamisme qu'il m'a donné dans les activités où j'ai eu le privilège de lui être associé.

Je remercie le Professeur C. LANGRAND et Jean-Paul DELAHAYE d'accepter d'être les examinateurs de cette thèse. Dans des domaines différents, leur compétence, la qualité du travail qu'ils réalisent, m'ont servi de modèle.

Je remercie le Professeur H. LECLERC, qui dirige l'unité INSERM que j'ai fréquenté durant de nombreuses années, d'accepter de faire partie de ce Jury. J'ai toujours trouvé auprès de lui et quand il le fallait le soutien sans lequel ce travail n'aurait pas pu voir le jour.

Cette thèse est l'aboutissement d'années de travail et de collaboration étroite et sans nuages avec Françoise GAVINI, Chargée de recherche à l'INSERM. Son esprit d'ouverture et sa grande intelligence ont été pour beaucoup responsables de la réussite de l'équipe que nous avons formée.

Je tiens tout particulièrement à remercier Janine DESCARPENTRIES qui a assuré avec compétence et avec le souci de perfection qu'on lui connaît la saisie électronique de cette thèse. Elle n'a pas hésité, une nouvelle fois, à consacrer une part de ses loisirs à l'achévement de ce travail.

CHAPITRE I

I N T R O D U C T I O N

Un travail d'application peut mettre en évidence des besoins spécifiques qu'il est difficile de satisfaire par des moyens existants, soit parce que ces moyens sont inaccessibles, soit parce qu'ils n'existent pas ou sont inadaptés. On est ainsi amené à développer des outils originaux dont l'utilisation dépasse le cadre de l'application initiale.

C'est l'exposé d'une démarche de ce type qui fait l'objet du développement suivant. Elle a pour domaine l'identification en bactériologie.

I - IDENTIFICATION EN BACTERIOLOGIE

I-1.- Position du problème.

La population bactérienne est soumise à un système de classification qui permet de regrouper les organismes ou souches bactériennes en espèces, unités de base de la classification, qui sont elles-mêmes regroupées en genre et les genres en familles ([GAV1 76], [GAV2 76] , [CAV 77] , [GAV1 80] , [GAV 81]).

Le processus d'identification consiste à reconnaître une bactérie inconnue en définissant son appartenance à l'un des groupes caractérisé préalablement par classification (espèce, genre, famille). La validité de l'identification sera donc dépendante de celle de la classification. Or, l'évolution des techniques et les nombreux travaux développés dans ce domaine font que la classification bactérienne et la définition des espèces sont en perpétuel remaniement.

L'identification d'une bactérie repose sur son profil de réponses à un certain nombre de tests biochimiques, correspondant à sa capacité de dégrader tel ou tel substrat ou à se développer dans certaines conditions. Les réponses à ces tests sont de type OUI NON, ce qui permet un codage de la réponse en + , - ou sous forme chiffrée en 0,1.
Pour un ensemble déterminé d'espèces, on dispose d'un ensemble particulier de tests. Le couplage espèces-tests constitue un **"kit d'identification"**.

La réalisation d'un kit d'identification repose sur le choix d'un ensemble de tests discriminants ([GAV2 80] , [LEF1 82] , [LEF2 82]). Pour une espèce donnée, il faut sélectionner une série aussi courte et performante possible de ces tests.

Les critères retenus reposent en général sur la minimisation d'une erreur d'identification. La gamme des procédés de sélection est vaste. Les techniques sont souvent empiriques mais heureusement une longue phase de validation permet d'aboutir à un résultat souvent incontestable.

Certains de ces kits ont été miniaturisés pour des raisons commerciales et de facilité d'emploi. L'utilisateur ensemence une série de cupules (une vingtaine en général) qui contiennent un réactif coloré. Il laisse la réaction s'opérer (souvent 24 heures) et observe les variations de couleur. Il note et code le résultat sous forme octale. Il obtient ainsi un **"profil"** de réponses qu'il faut interpréter.

• Dans la plupart des cas, l'**identification** fait appel à des méthodes probabilistes basées sur la formule de Bayes. En fonction du profil des réponses observé, on calcule une probabilité d'appartenance à chaque espèce. On retient l'espèce la plus probable.

Ce calcul peut être réalisé par un logiciel fourni avec le kit ([LEF1 82] , [LEF2 82]), mais le plus souvent l'utilisateur dispose d'un "codeur" dans lequel il trouve un classement des profils octaux les plus courants.

Certaines caractéristiques peuvent y être portées qui nuancent le résultat ou mettent l'accent sur d'autres possibilités. On distingue souvent à cet égard des identifications douteuses dûes à l'obtention d'une probabilité d'appartenance trop faible ou à l'existence de **caractères atypiques.** Il s'agit, dans le premier cas, d'une situation de concurrence trop vive avec d'autres espèces possibles, dans l'autre cas, de résultats peu fréquents, voire jamais rencontrés pour l'espèce considérée.

Ces situations ambigues peuvent amener à conseiller la mise en oeuvre de **tests complémentaires** qui permettent d'affiner le résultat. Voici quelques exemples tirés du codeur "API20EC". Ils concernent des souches de la famille des **Enterobacteriacae** qui proviennent toutes de l'environnement (sol, eau,...).

Profil : 1464442, très bonne identification à l'espèce
Escherichia-coli
(probabilité : 0.999).

Profil : 6617401, très bonne identification à l'espèce
Serratia-liquefaciens
(probabilité : 0.998)
présence d'un caractère atypique : **Inositol +.**

Profil : 6776401, très bonne identification au genre
Enterobacter.

Espèces probables :

- **E.amnigeneus** (probabilité : 0.523)
  présence de caractères atypiques : **Glycerol +, Sorbose -;**

- **E.cloacae** (probabilité : 0.468)
  présence des caractères atypiques : **Esculine - , Glycerol +, Saccharose + , Salicine -**

Pour une meilleure identification, réaliser le test complémentaire de croissance à 41° : **E.amnigeneus +, E.cloacae - .**

Pour une meilleure identification, réaliser le test complémentaire de croissance à 41° : **E.amnigeneus +, E.cloacae - .**

I-2.- La pratique du laboratoire

Les développements logiciels ont été réalisés pour un laboratoire de recherche en bactériologie spécialisé dans la taxonomie bactérienne ([LEF1 82] , [LEF2 82]). Pour ses activités, ce laboratoire est amené à développer de nouvelles méthodes d'identification, de nouveaux kits. Il agit également comme conseil et est conduit à traiter un grand nombre d'identifications douteuses.

Lors de sa réception, une souche bactérienne d'une famille donnée est répertoriée en fonction de sa provenance, de son origine (sang, eau,...), d'un code de référence unique. Elle subit ensuite les tests d'un ou de plusieurs kits d'identification. Pour chaque kit, on peut alors obtenir, par utilisation d'un logiciel spécifique :

- Un profil d'identification.
- La liste des caractères complémentaires (tests et modalités de réponses).
- La liste des espèces identifiées et les probabilités correspondantes.
- Les caractères atypiques (tests et modalités de réponses).

A la lumière de ces résultats et de l'expérience acquise, le bactériologiste juge de la qualité de l'identification à l'espèce la plus probable. Les critères de jugement sont nombreux, ils évoluent avec les connaissances du domaine. Certaines identifications jugées comme bonnes à une époque donnée peuvent être remises en cause par la découverte d'une nouvelle espèce.

II - REALISATION D'UN SYSTEME EXPERT EN IDENTIFICATION BACTERIENNE.

II-1.- Motivation

La réalisation est principalement motivée par la prise en compte des évolutions et respecte les contraintes suivantes :

1 - Il faut donner à l'utilisateur la possibilité de mettre en place une base de connaissances où seront précisés les critères qui permettent de juger une identification. Il doit pouvoir les consulter, modifier à sa convenance, ajouter de la connaissance de la manière la plus conviviale possible.

Il faut en conséquence donner aux critères de jugement une forme déclarative plus adaptée à la mise à jour et à la consultation. Cette mise en place est possible puisque ces critères peuvent être définis sous forme de règles qui ont toutes la forme :

si liste de conditions **alors** conclusion.

2 - La découverte de nouvelles espèces peut amener à reconsidérer certaines identifications. Il importe donc de conserver les résultats et de les rendre facilement accessibles et modifiables.

3 - Il faut intégrer dans un système global la démarche existante qui permet la définition et l'utilisation de kits d'identifications.

II-2.- Caractéristiques

L'aspect convivial à donner au système, ses caractéristiques évolutives, l'existence d'une base de connaissances que l'on peut décrire à l'aide de règles logiques conduisent naturellement à une solution système expert [LEF 83] . Les composants principaux en sont :

- Une interface utilisateur chargée de la gestion du dialogue : introduction des profils à identifier, nature du kit d'identification à appliquer, édition des résultats, demande d'explications et de justifications, gestion de la connaissance.

- Un mécanisme permettant d'obtenir les probabilités d'appartenance aux espèces, les caractères atypiques et complémentaires.

- Un moteur d'inférence chargé d'exploiter ces résultats à l'aide des règles d'interprétation.

- Une base de connaissances formée de ces règles.

- Une base de faits formée de l'ensemble des résultats acquis et interprétés.

II-3.- Contraintes et choix d'implantation.

Les caractéristiques du système pouraient être considérées comme conventionnelles s'il n'y avait l'obligation de gérer une base de faits très importante puisque composée de l'ensemble des résultats acquis (plus de 10000 éléments actuellement et en constante évolution)

Il faut opter pour un outil qui concilie les aspects système expert et base de données.

Il faut que l'interface utilisateur soit suffisamment développée et personnalisée pour que l'utilisateur bactériologiste puisse gérer, après un temps d'accoutumance la totalité du système (interrogation et mise à jour de la B.D., définition et actualisation de la base de connaissances).

Le développement a dû etre réalisé sur un PC-AT disposant de 640 K de mémoire centrale.

Or, les systèmes de gestion de bases de données ne permettent pas encore une utilisation des connaissances suffisante sur le plan déductif.

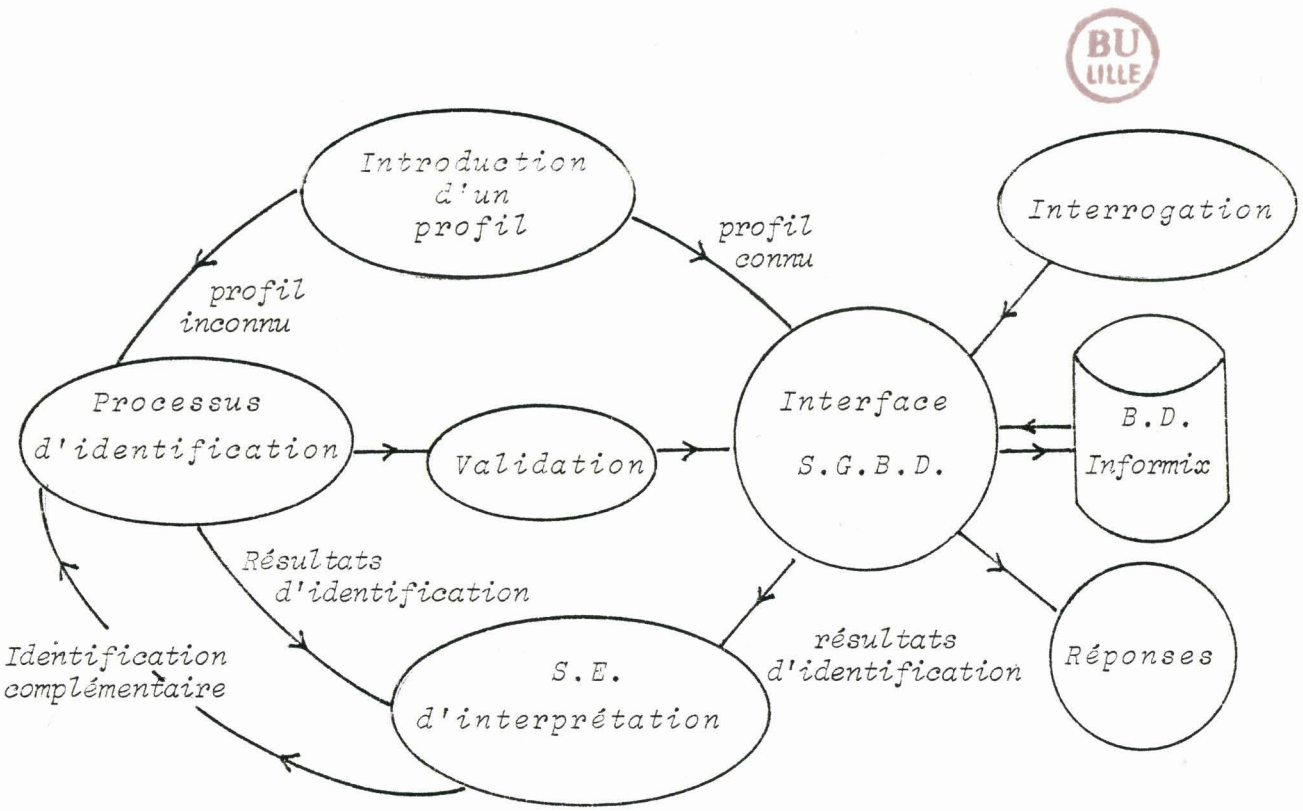
Les générateurs de systèmes experts actuels ne couvrent pas, ou mal, la totalité des aspects évoqués. De conception trop générale, ils ne prennent pas assez en compte la spécificité de l'application.

Le choix de PROLOG comme langage d'implantation résulte d'un compromis. Il permet la réalisation du moteur d'inférence et la gestion des règles puisqu'il est conçu pour satisfaire des besoins de ce type. On peut lui adjoindre des capacités complémentaires qui lui permettent d'utiliser et de modifier les données contenues dans une base. On pourra trouver en [COND 86, chap. 4] un exposé général sur la coopération PROLOG-SGBD.

Cette interface est développée dans le cadre d'un S.G.B.D. particulier : **Informix.** De type relationnel, ce S.G.B.D. offre en effet toutes les fonctionnalités requises. Il permet deux formes de consultation à l'aide d'un langage d'interrogation de type SEQUEL et d'un gestionnaire de transactions du type Q.B.E. [INFOR 84] . Il est en outre doté d'un générateur d'états et d'une interface avec le langage C. Cette interface est utilisée pour construire le lien avec PROLOG.

II-4.- Architecture du système d'identification

Le système complet repose sur un ensemble de logiciels qui utilisent et alimentent la Base de Données chargée de recevoir les résultats.



Le processus d'identification est de réalisation purement procédurale. L'identification est obtenue en effet essentiellement par calculs de probabilités et ne présente guère d'aspect déclaratif.

Ce processus repose sur un ensemble de programmes développés depuis une dizaine d'années qui permettent la définition et l'utilisation de kits d'identification. Les fondements théoriques et l'historique de la réalisation sont exposés dans le chapitre 2.

L'interface S.G.B.D. est au coeur de l'ensemble. Elle gère l'accès à une base de données destinée à recevoir les résultats relatifs aux identifications. Elle est utilisée par le Système Expert d'interprétation et alimentée par le système d'identification. Elle peut être interrogée parallèlement et fournir des informations spécifiques qui permettent l'évolution des données qui régissent le système.

Les caractéristiques de cette interface sont exposées au chapitre III. Les aspects techniques sont développés dans les annexes .

Le rôle du Système Expert proprement dit est réduit à l'interprétation des résultats. Plus que les données qui permettent la réalisation d'une identification, c'est en effet dans ce domaine que les évolutions et les modifications sont les plus fréquentes. C'est aussi dans ce domaine que les connaissances s'expriment naturellement de façon déclarative. Les cas d'identifications douteuses révélés par l'interprétation peuvent amener à la demande d'une identification complémentaire et au retour au processus d'identification. Le Système Expert est présenté au chapitre IV. Le programme PROLOG correspondant est commenté dans l'annexe correspondante.

CHAPITRE II

RECONNAISSANCE DE SOUCHES BACTERIENNES

CONCEPTION ET UTILISATION DE KITS D'IDENTIFICATION

Ce chapitre est consacré à l'exposé de réalisations logicielles entreprises dans le cadre de l'identification bactériologique entre 1976 et 1982. Ces développements font partie du système tel qu'il a été présenté au paragraphe II-4 de l'introduction. Ils s'y retrouvent soit sous leur forme initiale (module de sélection de tests discriminants et de conception de kit d'identification) soit ont été reécrits pour être intégrés au système expert (module d'identification).

I - INTRODUCTION

L'unité INSERM d'écotoxicologie du Professeur LECLERC a entamé, depuis 1975, un vaste programme de recherches dont la responsabilité incombe à Melle GAVINI. Il a pour but de pallier à la mauvaise connaissance des bactéries se développant dans les milieux naturels. Ces bactéries ne présentant pas de caractère pathogène, leur étude systématique n'a donc pas été entreprise jusqu'à ces dernières années.

Dans un premier temps, ce programme s'est limité au groupe coliforme de la famille des entérobactéries. Un millier de ces souches furent isolées d'eaux polluées de diverses provenances ou de terres. Chacune de ces souches fut cultivée dans un milieu nutritif.

Les germes cultivés, identiques à la souche mère, sont soumis à une batterie de tests (220 dans le cas de ce programme de recherches). Ces tests mettent en évidence certaines caractéristiques biochimiques. Ils peuvent révéler la présence d'un enzyme ou montrer l'utilisation d'un substrat (sucres, acides aminés).

Le résultat positif du test se traduit par une coloration du milieu réalisée par un indicateur spécifique de la réaction. On le code généralement :

0 : absence de réaction,
1 : réaction positive,

on a alors affaire à un test à 2 modalités ou dichotomique.

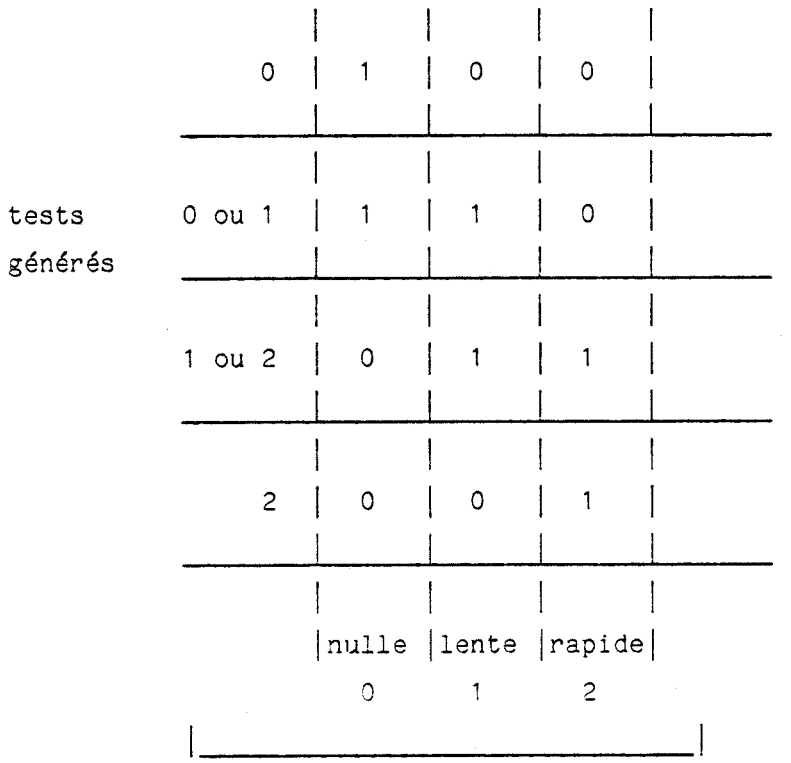
Seule la réponse 1 est considérée comme significative.

Dans certains cas, pour affiner le résultat, on fait intervenir la notion de durée; les réponses se codent alors, dans le cas d'un test à 3 modalités :

0 : absence de réponse
1 : réaction positive en 48 h.,
2 : réaction positive en 24 h..

Généralement, ces tests, dits ordinaux, ont de 3 à 5 états qui peuvent être ordonnés par valeurs croissantes.

Pour permettre l'analyse de ces résultats par un ensemble de méthodes taxonomiques, des tests ont été transformés en tests binaires. Chaque test ordinal a été partionné sous forme de scalogramme de GUTTMAN. Cette transformation est illustrée par l'exemple du test à 3 modalités précédent.



Quand on calcule les proximités à l'aide d'un indice de similarité qui fait intervenir le nombre de 1 en commun, cette méthode de codage respecte l'ordre existant entre les modalités du test initial. Un organisme ayant une réponse rapide sera plus éloigné d'un organisme donnant une réponse nulle que d'un organisme donnant une réponse lente.

Pour l'analyse d'un tableau de données binaires, on dispose d'un large éventail de méthodes. Certaines sont d'un usage courant en bactériologie : ce sont principalement les méthodes de classification hiérarchiques. Trois de ces méthodes ont été choisies qui diffèrent par l'utilisation de critères d'agrégation différents. Chacune d'elles donne un éclairage particulier et révèle certaines caractéristiques des données ([SNEATH 1963] ,[DEL 71]).

Certaines classes de souches peuvent être identifiées sans ambiguïté. Leur composition n'est pas affectée par un changement de critère d'agrégation. Dans d'autres cas, la présence de souches intermédiaires rend l'interprétation difficile.

L'analyse factorielle des correspondances permet, en fournissant un résumé facilement interprétable des distances entre individus, l'élimination de cette catégorie de souche et rend la composition des classes plus homogène.

L'utilisation de ces techniques d'analyse a amené la découverte de nouvelles espèces. Ces résultats ont été confirmés au moyen d'hybridations qui mirent en évidence des différences génériques significatives.

L'identification de souches inconnues à ces nouveaux groupes ne peut être envisagée de manière courante sur la base des 220 caractères initialement choisis. Le problème du choix d'un sous-ensemble de tests discriminants fut rapidement posé. La solution a été envisagée dans le cadre d'une méthodologie de fondement Bayesien qui s'appuie sur des résultats de MM. TERRENOIRE et TOUNISSOU [TER 76] . Elle a nécessité l'écriture d'un logiciel original sur micro-ordinateur.

Un premier programme est destiné au choix de tests discriminants,

Le second permet une identification de souches inconnues à l'aide des tests précédents.

II - DEFINITION DE LA MATRICE DE DONNEES

Il a été décidé de sélectionner deux sous-ensembles de tests formant deux clés d'identification. La première clé est destinée aux **entérobastériacaes** d'origine médicale, la seconde aux **entérobactéricaes** issues de l'environnement.

Les données proviennent de deux sources :

- l'une d'elles, originale, est formée des résultats obtenus après classification,

- l'autre, pour les espèces déjà répertoriées, émane d'une compilation de résultats publiés et communément acceptés.

Malgré la provenance hétérogène de ces sources, une matrice de probabilités conditionnelles a pu être définie entre chaque espèce et les éléments d'un sous-ensemble de caractères réputés pour leurs bonnes propriétés : indépendance, facilité de mise en oeuvre, fiabilité, etc...

On notera :

$$p(q^i_r/c_j)$$

la probabilité d'obtenir la modalité q^i_r du test i pour une souche appartenant au groupe c_j. Ces probabilités ont été estimées par les fréquences observées ou publiées. La qualité de ces estimations est parfois discutable pour les groupes nouveaux dont l'effectif est

faible. Elles constituent cependant une première ébauche, présentement la seule accessible.

Par contre, l'estimation des probabilités a priori $p(c_j)$ est plus délicate et prête à controverse. Les spécialistes ([PYB 68] , [FRIED 73] , [LAP 73]) s'accordent généralement pour considérer que l'équiprobabilité des groupes est l'hypothèse la moins mauvaise tant les facteurs liés au milieu rendent illusoires les tentatives d'estimation fondées sur des dénombrements.

III - SELECTION DE CARACTERES DISCRIMINANTS

La forme des données et la nature de l'objectif à atteindre conduisent à utiliser une méthode de sélection pas à pas. Le caractère retenu à chaque étape optimise un critère. Le processus se poursuit jusqu'à la réalisation d'une condition limite.

Le critère à optimiser consiste à minimiser la probabilité d'erreur quand on choisit, pour un profil de réponses donné relatif à un ensemble de tests sélectionnés, le groupe d'appartenance le plus probable.

Si Q est l'ensemble des tests sélectionnés, la probabilité d'erreur pour un profil x de réponse parmi l'ensemble Π_Q des profils possibles s'exprime par :

$$Pe(c/x) = 1 - \underset{i \leq j \leq N}{\text{Max}} \; p(c_j/x)$$

($p(c_j/x)$ est la possibilité d'appartenance au groupe c_j pour un individu ayant fourni le profil x de réponses aux tests de Q).

Si l'on considère l'ensemble des profils possibles, la probabilité d'erreur moyenne vaut :

$$Pe(Q) = \sum_{x \in \Pi_Q} p(x)\ Pe(c/x)$$

La recherche de $Pe(Q)$ minimum, même pour un ensemble Q construit itérativement, se heurte rapidement à l'obstacle que constitue la taille de Π_Q : plus d'un million de profils différents pour un ensemble de 20 tests dichotomiques.

TOUNISSOUX [TOUN 80] a cependant mis en évidence un bon encadrement de cette probabilité, plus facilement calculable. On choisira, en conséquence, de minimiser la borne supérieure $T(Q)$ de cet encadrement :

$$T(Q) = B(Q) \quad \textbf{si} \quad B(Q) \leq 1/2$$

$$T(Q) = 2\ B(Q)/(1 + 2\ B(Q)) \quad \textbf{si} \quad B(Q) > 1/2$$

où $$B(Q) = \sum_{j<k} (p(c_j)\ p(c_k))^{1/2} \prod_{r=1}^{R} I_{jk}(q_r)$$

$p(c_j)$ = la probabilité a priori d'appartenance au groupe c_j (prise égale à $1/N$)

$$I_{jk}(q_r) = \sum_{i=1}^{m_r} (p(q^i{}_r/c_j)\ p(q^i{}_r/c_k))^{1/2}$$

Les probabilités conditionnelles $p(q^i_r/c_j)$ sont extraites de la matrice de données. L'encadrement obtenu est valide si dans chaque g hypothèse n'est pas généralement pleinement réalisée, mais, comme leroupe les tests peuvent être considérés comme indépendants. Cette notent TERRENOIRE et TOUNISSOUX [TER 76] l'indication fournie par T(Q) suffit au choix d'un "bon" sous-ensemble des tests.

IV - IDENTIFICATION DE SOUCHES

L'identification d'une souche à un groupe c est établie en considérant, pour un profil x de réponses produit par la souche, le maximum des probabilités conditionnelles $p(c_j/x)$. La méthode utilisée repose sur les mêmes hypothèses que celles mentionnées pour le processus de sélection. On suppose, en outre, que l'individu à identifier est bien un membre de la population étudiée.

Les probabilités a posteriori $p(c_j/x)$ sont évaluées à l'aide des probabilités $p(x/c_j)$ en appliquant la formule de Bayes :

$$p(c_j/x) = p(x/c_j)\ p(c_j)/p(x)$$

Dans l'espace Π_Q des profils possibles, un profil particulier x se développe sous la forme :

$$x = \bigcap_{r=1}^{m_Q} q_r{}^{i(x)}$$

où $m_Q = \mathrm{Card}(Q)$.

$q_r \in Q$ et $q_r{}^{i(x)}$ est le sous-ensemble de Π_Q formé des profils possédant la même modalité que x au test q_r .

$p(x)$ peut se développer sous la forme :

$$p(x) = \sum_{j=1}^{n} p(x/c_j) = \sum_{j=1}^{n} p(x/c_j)\, p(c_j)$$

L'hypothèse d'indépendance déjà évoquée permet d'expliciter $p(x/c_j)$:

$$p(x/c_j) = \prod_{r=1}^{m_Q} p(q_r{}^{i(x)}/c_j) \ .$$

Les probabilités $p(c_j/x)$ peuvent ainsi être estimées à l'aide des fréquences observées, définies dans la matrice des données.

V - LE LOGICIEL D'APPLICATION

Une première version du logiciel, écrit en Fortran, a été mis en place sur micro processeur, sous CPM ([LEF1 82] , [LEF2 82] .

Pour utiliser au maximum une taille mémoire limitée à 54 K, le logiciel a été découpé en modules indépendants qui communiquent entre eux par l'intermédiaire de fichiers. Il peut prendre en compte 50 groupes et un ensemble de tests présentant au plus 100 modalités.

- Un premier module permet l'entrée et le contrôle des données : il s'agit essentiellement de fréquences estimant les probabilités $p(q^i{}_r/c_j)$.

- Vient ensuite un module de sélection de caractères discriminants. Il utilise le fichier de données précédent et nécessite l'introduction de paramètres spécifiques :

* nombre de caractères à sélectionner
* erreur maximum souhaitée,

(ces valeurs sont utilisées par le programme qui décide l'arrêt du processus de sélection dès qu'un de ces critères est satisfait).

* liste des groupes à identifier,
* liste des tests utilisables,

(ces listes permettent de spécifier la portée du processus de sélection).

- Le dernier module est consacré à l'identification d'individus nouveaux.

Il peut opérer sur toute clé répertoriée, ou sur une clé introduite par l'utilisateur lors de la phase d'initialisation. Le profil de réponses x est ensuite introduit pour permettre le calcul des probabilités "absolues" $p(x/c_j)$ et celui des probabilités "relatives" $p(c_j/x)$ pour l'ensemble des groupes c_j. Conformément à une pratique courante en bactériologie, certaines fréquences ont été modifiées : les valeurs nulles de $p(q^i_r/c_j)$ ont été relevées à 1 % et de façon symétrique les valeurs égales à 100 % , abaissées à 99 % . L'absence de certaines modalités n'est généralement pas considérée comme définitive, une mutation pouvant remettre en cause un tel résultat. Les nouvelles valeurs, arbitrairement choisies, permettent de ne pas rejeter a posteriori un profil même si celui-ci n'a pas été observé en laboratoire. Le programme édite un tableau regroupant les probabilités "absolues" et "relatives" calculées pour chaque classe c_j . L'utilisateur peut ainsi contrôler la validité des résultats fournis. Une édition supplémentaire met en évidence, pour chaque modalité du profil introduit, sa probabilité d'observation dans le groupe d'appartenance le plus probable. Elle permet de compléter le contrôle précédent.

Dans le cas où l'appartenance à d'autres groupes ne peut être rejetée, le programme fournit une liste des groupes pour lesquels le profil a une probabilité d'observation supérieure à un seuil donné. On propose à l'utilisateur une nouvelle sélection de caractères discriminants. Cette sélection est faite parmi les tests ne figurant pas déjà dans la clé. Elle a pour objet de mieux séparer les groupes pour lesquels le doute subsiste. Si l'utilisateur le souhaite, le programme édite une liste de $p(q^i_r/c_j)$ pour chaque nouveau groupe à discriminer.

VI - APPLICATIONS

Deux clés d'identification ont été constituées à l'aide du logiciel précédent. Elles correspondent à des préoccupations différentes :

L'une est destinée au contrôle des aliments et des eaux. Elle concerne 30 espèces dont 7 nouvelles. Elle a permis la réalisation d'une version "industrielle" qui, achevée en 1985, est maintenant commercialisée par la société "API" sous le sigle API20EC.

L'autre est destinée au diagnostic médical. Elle discrimine 35 espèces déjà toutes répertoriées.

Dans les deux cas, des contraintes techniques ont imposé une limitation entre 24 et 20 du nombre de tests sélectionnés. Cette quantité est acceptable en regard du taux d'erreur théorique dont une majoration est calculée par le module de sélection.

L'efficacité de ces clés a été démontrée sur plus de 800 souches. Là où la comparaison était possible (espèces connues) d'autres systèmes d'identification ont été utilisées (API et Micro ID). Une coïncidence de 100 % a été observée avec au moins l'un des systèmes précédents.

Certaines souches restent non identifiées (6 %). Toutes d'origine hydrique, elles peuvent appartenir à des nouveau groupes.

VII - CONCLUSION

Ces résultats satisfaisants ne sauraient masquer le caractère incomplet d'un tel système.

Les techniques bactériologiques sont en effet en constante évolution.

La connaissance de la flore bactérienne s'améliore parallélement mais reste très incomplète. Certains résultats acquis peuvent être remis en question.

Le caractère non exhaustif des données de base se traduit par l'absence d'identification pour certaines souches. Il laisse également craindre que des identifications erronnées puissent se produire pour les souches dont l'appartenance à la population de référence ne peut être établie de façon sûre.

Le expérimentations ultérieures ont pu remédier à certaines de ces lacunes, mais seule la conception d'un système évolutif et auto-correcteur peut apporter une solution générale et satisfaisante aux problèmes soulevés.

La première version présentée dans ce chapitre a fait place progressivement à un nouveau système qui tente de rémédier à ces défauts.

Les chapitres suivants sont consacrés à l'exposé de ses caractéristiques essentielles en commençant par la coopération PROLOG-S.G.B.D. autour de laquelle s'organise une bonne partie des traitements.

CHAPITRE III

INTERFACE PROLOG-S.G.B.D. INFORMIX

La structure, les propriétés d'une interface PROLOG-INFORMIX y sont développées et présentées dans le cadre de l'application à l'identification bactérienne.

On présente, pour commencer, le système d'information de l'application et son implantation sous la forme d'une base de données relationnelle.

Cette base est ensuite utilisée pour illustrer les fonctions des différents niveaux de l'interface.

Cet exposé est général et met l'accent sur les possibilités de chacun des niveaux. Il évite autant que possible les aspects techniques propres à la réalisation.

Pour ceux-ci et pour une présentation complète de la programmation, on peut se reporter aux annexes 1, 2, 3 et 4.

I - SCHEMA CONCEPTUEL DE LA BASE

Une analyse des informations à ranger dans la base fait apparaître les ensembles d'entités suivants :

clé : nom d'un kit d'identification composé d'un ensemble d'espèces et de tests discriminants.

espèce : nom d'une espèce faisant partie d'un kit d'identification.

test : nom d'un test biochimique utilisé dans un kit et servant à l'identification.

test complémentaire : nom d'un test pouvant être utilisé en complément d'un kit pour préciser une identification douteuse.

profil : une séquence de réponses aux tests d'un kit permettant une identification. Cette séquence est souvent codée sous forme d'un nombre octal. Elle concerne une souche bactérienne.

espèce identifiée : espèce d'appartenance probable pour une souche bactérienne. On lui associe une probabilité d'obtention du profil de réponses pour l'expèce considérée.

caractère atypique : test et modalité de réponse anormale pour l'espèce la plus probable.

caractère complémentaire : test et modalité de réponse complémentaire précisant l'identification.

référence : numéro de référence d'une souche bactérienne identifiée.

Une clé est composée d'espèces, de tests et de tests complémentaires, elle concerne un ensemble de profils de réponses.

Pour chaque profil, on obtient une liste d'espèces identifiées et de probabilités, une liste de caractères atypiques et une liste de caractères complémentaires.

Un profil correspond aux réponses obtenues lors de l'identification de certaines souches bactériennes. Chaque souche est référencée par un numéro.

Les associations correspondantes sont donc les suivantes, elles sont du type 1-n :

clé --> espèce
clé --> test
clé --> test complémentaire
clé --> profil
clé x profil --> caractère atypique
(test x modalité)
clé x profil --> caractère complémentaire
(test x modalité)
clé x profil --> espèce identifiée
(espèce x probabilité)
clé x profil --> référence de souche

Ces associations conduisent à la définition des relations :

ESPECES : clé, espèce, référence d'espèce,
TESTS : clé, test, référence de test,
TESTS COMPLEMENTAIRES : clé, test complémentaire,
référence de test complémentaire,
PROFILS : clé, profil, référence de profil,
ESPECES-IDENTIFIEES : référence de profil, référence
d'espèce, probabilité,

CARACTERES-ATYPIQUES : référence de profil, référence de test complémentaire, modalité,

PROVENANCES : référence de profil, référence de souche.

Les composants "référence de profil" caractérisent de manière unique un couple **clé x profil.** Ils assurent de façon plus économique les lien entre les relations PROFILS, ESPECES-IDENTIFIEES, CARACTERES-COMPLEMENTAIRES.

Les composants références d'espèce, de test, de test complémentaire, jouent un rôle analogue et correspondent aux couples **cle x espèce, clé x test, clé x test complémentaire.**

II - INTERFACE PROLOG-INFORMIX

II-1.- Objectifs

La réalisation doit répondre à plusieurs objectifs d'importance et de difficulté croissante.

Le premier sans lequel les autres ne pourraient être satisfaits consiste à ajouter à PROLOG les primitives qui lui permettent de gérer les données rangées dans une base.

Le second est de faciliter les accès au contenu de la base. Un utilisateur non averti doit pouvoir rédiger une requête sans connaître les détails relatifs à l'implantation.

La satisfaction de ces objectifs passe par la réalisation de plusieurs niveaux d'interfaçage de fonctionnalités croissantes.

II-2.- Interface de niveau 1

Le niveau 1 met en oeuvre les fonctions définies dans Informix pour la liaison avec le langage C. Elles sont ajoutées à l'interpréteur Prolog sous forme de prédicats évaluables.

II-2-1.- Interface Informix, langage C

L'interface repose sur l'utilisation de fonctions qui permettent d'une part l'acquisition de connaissances sur la structure de la base et, d'autre part, la manipulation des données.

Toutes les informations relatives à l'implantation de la base sont accessibles :

- le nombre de relations et, pour chacune d'elles :
  le nom, le nombre d'éléments du n-uplet composant et sa longueur.

- pour chaque élément dse n-uplet : le nom, le type (chaîne, entier, composé,...) et la longueur.

- pour chaque élément composé : les éléments de la composition et leur type.

- pour chaque index d'un n-uplet : le nom et le type (index ou non index, duplicable ou non duplicable).

On dispose de fonctions permettant l'ouverture et la fermeture de la base et des fichiers relations. On peut ajouter, supprimer, modifier un n-uplet de relation, ajouter ou supprimer un index.

Les accès aux n-uplets d'une relation peuvent être effectués soit de façon indexée soit de façon séquentielle. Pour illustrer ces possibilités, voici deux exemples de programmes C utilisant l'interface :

- Le premier réalise un parcours séquentiel des n-uplets de la relation des ESPECES :

```
main()
{
char enr[100];
int j,cond;
short type, longueur, perms;
char *listechamp[10],nomchamp[10][10];

/*ouverture de la base*/
dbselect(DBOPEN,"bacter");

/*ouverture de la relation ESPECES*/
dbselect(FILEOPEN,"esp");

/*définition d'une vue sur l'ensemble des composants de la
relation ESPECES*/
for (j=0;
    !dbnfield(j,"esp",nomchamp[j],xtype,xlongueur,xperms);
    j++) listechamp[j]=nomchamp[j];

/*initialisation pour un accès séquentiel*/
dbselfield("esp",0,ACCSEQUENTIAL);

/*recherche du premier n-uplet*/
cond=dbfind("esp",FIRST,0,xlongueur,enr);
```

```
while(cond=0))
  enr[longueur] = 0;
  /*édition du composant nom d'espèce*/
  printf("%\n",enr+12);

  /*recherche du n-uplet suivant*/
  cond = dbfind("esp",NEXT,"",xlongueur,enr);
)

  /*fermeture de la relation et de la base*/
  dbselect(FILECLOSE,"esp");
  dbselect(DBCLOSE,"bacter");
)
```

- Le second est une recherche indexée du premier n-uplet de la relation des TESTS dont la valeur pour le composant nom de test est supérieure à "APP"

```
main()
(
char enr[100];
int j,cond;
short type,longueur,perms;
char *listechamp[10],nomchamp[10][10];

dbselect(DBOPEN,"bacter");
dbselect(FILEOPEN,"car");

for(j=0;
   !dbnfield(j,"car",nomchamp[j],xtype,xlongueur,xperms);
   j++) listechamp[j]=nomchamp[j];
dbsetfileview("car",listechamp,(int)j);
```

```
/*initialisation pour un accès indexé sur le nomposant nom de
test*/
dbselfield("car","carlib",ACCKEYED);

/*recherche du premier n-uplet dont la valeur est supérieure
à APP*/
dbfind("car",GREATER,"APP",2longueur,enr);
printf("%\n",enr+12);

dbselect(FILECLOSE,"car");
dbselect(DBCLOSE,"bacter");
)
```

Ces exemples illustrent les caractéristiques de l'interface avec le langage C . La programmation des accès à la base est lourde et délicate. On est loin des possibilités offertes par les langages d'interrogation classiques. A ce niveau la base se comporte comme un ensemble de fichiers séquentiels indexés pour lesquels on peut définir et utiliser un nombre quelconque d'index.

II-2-2.- Les prédicats évaluables d'accès à la base

Ils constituent une extension de l'interpréteur PROLOG et utilisent les fonctions de l'interface au langage C. A la fonction dbselect(DBOPEN,"bacter") correspond ainsi le prédicat ouvrir-bd("bacter"). Après vérification du type de l'argument qui doit être instancié à un nom, le prédicat réalise l'ouverture de la base. Il est évalué à vrai si ces opérations sont effectuées avec succès.

De même, info-numchamp("esp",1,n,t,ℓ) donne, à l'image de la fonction dbnfield(1,"esp",n,t,ℓ) , le nom (n) , le type (t) et la longueur (ℓ) du composant numéro 1 de la relation des espèces.

A l'aide de ces prédicats évaluables, on peut écrire en PROLOG l'équivalent d'un programme C :

```
ouvrir-bd("bacter")
ouvrir-fich("car")
init-ind("car","carlib")
trouver-clé("car","carlib",grand,"APP")
vue("car","carlib".nil,v-carlib.nil)
écrire(v-carlib)
fermer-fich("car")
fermer-bd("bacter");
```

On traduit dans cet exemple le programme de recherche indexée cité dans le paragraphe précédent. Lors de l'ouverture de la relation ouvrir-fich définit une vue sur l'ensemble des composants. Trouver-clé réalise la recherche indexée et vue extrait d'un tampon interne à l'interpréteur la valeur du composant nom de test (carlib).

L'interface de niveau 1 répond à l'un des objectifs poursuivis : PROLOG peut accéder aux informations rangées dans une base. Les probabilités sont cependant trop élémentaires et l'utilisation délicate. Il s'agit là néanmoins du point de départ obligé pour la constitution d'une interface plus évoluée.

II-3.- Interface de niveau 2

Cette interface est écrite en PROLOG (le programme est commenté dans l'annexe Interface 2). Ce niveau réalise une interrogation comportant des sélections relatives à plusieurs composants. La combinaison de ces sélections sur plusieurs relations permet de **situer cette interface au niveau des langages d'interrogation classiques.**

On accède aux opérations de sélection, produit et projection comme le montre l'exemple suivant qui utilise la base "bactéries" .

On désire connaître le nom des espèces identifiées et les probabilités correspondantes pour le profil de réponses "0617401" de la clé "APINOV".

II-3-2.- Réalisation à l'aide d'Informix

Dans le langage d'interrogation d'Informix, cette requête s'exprime par :

```
print esplib,id_prob where pro_profil = "0617401"  and
pro_cle = "APINOV" joining pro_ref = id_ref and
id_comp = esp_comp.
```

Elle met en jeu les relations des espèces, des profils et des espèces identifiées. On sélectionne dans la relation PROFILS les n-uplets qui correspondent à "0617401" et à la clé "APINOV". Le composant référence de profil met en correspondance les n-uplets de la relation PROFILS et ceux de la relation ESPECES-IDENTIFIEES (pro_ref = id_ref). On trouve ainsi dans la relation des espèces identifiées le composant relatif à la probabilité. Le nom s'obtient dans la relation des espèces par la mise en oeuvre de la jointure qui porte sur la référence d'espèce (id_comp = esp_comp).

```
PROFILS :
     clé(pro_clé) : "APINOV"
     profil (pro_profil) ; "0617401"
     référence de profil (pro_ref)  <--------
                                              |
 ESPECES IDENTIFIEES :                        |
     référence de profil (id_ref)    <-------
     référence d'espèce (id_comp)    <-------
     probabilité (id_prob)                    |
                                              |
 ESPECES :                                    |
     espèce (esplib)                          |
     référence d'espèce (esp_comp)   <-------
```

II-3-2.- Réalisation à laide de Prolog

La formulation de cette question, dans le cadre de l'interface PROLOG, utilise le prédicat recherche-mult. Elle se présente de manière classique :

```
  recherche-mult(
       <"profils",
            <"pro_profil",égal,"0617401",v-profil>.
            <"pro_cle,égal,"APINOV",v-clé>.
            <"pro_ref",o1,v1,v-ref>.nil>.
       <"ident",
            <"id_ref",egal,v-ref,x>.
            <"id_comp",o2,v2,v-comp>.
            <"id_prob",o3,v3,v-prob>.nil>.
       <"esp",
            <"esp_comp",egal,v-comp,y>.
            <"esplib",o4,v4,v-lib>.nil>.nil)
```

Recherche-mult définit une suite ordonnée de sélections relative aux relations "profils", "ident", "esp". Chaque sélection peut porter sur plusieurs composants et utiliser ou non des opérateurs (égal, grand, petit, pteq, gdep).

La mise en oeuvre d'une sélection au niveau d'un composant particulier dépend de l'existence et de la nature des paramètres composants, opérateur, valeur de comparaison. Les renseignements nécessaires sur le type de composant, l'existence d'un index... etc,... sont extraits de la base à l'aide des prédicats correspondants du niveau 1.

On commence par l'examen du paramètre opérateur. S'il n'est pas précisé (la variable correspondante est libre), on considère que l'on a affaire à un parcours séquentiel que l'on réalise de manière classique : recherche du premier élément puis des éléments suivants à l'aide des prédicats du niveau 1. Tous les éléments trouvés sont pris en compte.

S'il est précisé, deux cas peuvent se produire suivant qu'il existe ou non une valeur de comparaison.

L'absence d'une telle valeur entraîne la réalisation d'un parcours séquentiel comme dans le cas précédent.

Sa présence implique un choix des éléments à retenir. Ce choix est réalisé en tenant compte de l'existence d'un index pour le composant qui permet un positionnement direct sur le premier élément à retenir.

La recherche des éléments suivants dépend de l'opérateur et du type de l'index (duplicable ou non duplicable). Elle est réalisée de façon séquentielle. Si le composant n'est pas indexé, on met en oeuvre un parcours séquentiel au cours duquel ne seront retenus que les éléments convenables.

Les sélections sur composant que l'on trouve dans une requête relative à une relation ne peuvent pas jouer le même rôle.

Il faut définir la sélection principale : celle par laquelle le traitement va commencer. Elle détermine l'ensemble des n-uplets à examiner et à passer au crible des critères restants. Les éléments incompatibles sont rejetés et ne participent pas à la solution. Le coût de l'opération est ainsi étroitement lié à l'importance des éléments retenus lors de la première étape.

Il importe donc de choisir dans une suite de sélections celle qui est la plus précise. On exprime pour ce faire une préférence générale pour les opérateurs égal, pour un composant indexé et non duplicable... Un des intérêts de PROLOG est de permettre l'expression de la stratégie de choix sous forme de règles. Elle est ainsi clairement définie et en conséquence facilement modifiable.

L'utilisation de connaissances plus précises, relatives à une base particulière, peut contribuer à améliorer les coûts de traitement. Dans le cas de la base bactéries, on précise par exemple que le composant référence de profil est à choisir s'il est en compétition avec un autre composant dans le cadre d'une sélection. Le champ des valeurs trouvées est dans ce cas généralement plus restreint.

Ce travail de mise en place est une opération coûteuse qu'il est intéressant d'éviter lorsque se représente une sélection qui offre des caractéristiques analogues :

- mêmes composants,
- mêmes opérateurs,
- présence de valeurs de comparaison aux mêmes emplacements.

On définit pour une série de sélections sur une relation, **un schéma type qui peut s'identifier à toute requête comparable**. On utilise à cette occasion les possibilités de PROLOG relatives à l'unification. On implante ce schéma type sous la forme d'un objet non instancié doté de caractéristiques et de propriétés :

- nom de composant,
- opérateur défini ou non défini,
- valeur de sélection présente ou non présente.

Si la requête porte sur plusieurs relations comme dans l'exemple proposé, l'éventuelle liaison (jointure) est réalisée en PROLOG à l'aide de variables qui, selon le cas, reçoivent des valeurs de comparaison ou des valeurs trouvées.

Sans l'exemple précédent, la jointure entre les relations PROFILS et ESPECES-IDENTIFIEES s'exprime en PROLOG par l'intermédiaire de la variable v-ref. La valeur trouvée par la sélection dans la relation des profils est transportée dans la sélection suivante comme valeur de comparaison. La jointure entre les relations des espèces identifiées et des espèces est réalisée à l'aide d'un procédé analogue.

II-4.- Interface de niveau 3

La formulation d'une requête à l'aide de l'interface de niveau 2 nécessite une bonne connaissance de l'implantation de la base. Il est souhaitable, dans la mesure du possible, de soulager l'utilisateur de certains aspects relatifs à cette implantation et non directement liés à l'expression de la question.

Pour déterminer quels sont les caractères atypiques relatifs au profil "0617401" de la clé "APINOV" , il est indispensable, dans le cadre de l'interface de niveau 2, de préciser, en plus des éléments qui se rapportent à la question :

- les noms de relations qui interviennent,
- les noms des composants qui sont impliqués,
- les liens entre les relations constituantes.

```
recherche-mult(
    <"profils",
        <"pro_profil",egal,".617401",v-profil>.
        <"pro_cle",egal,"APINOV",v-cle>.
        <"pro_ref",o1,v1,v-ref>.nil>.
    <"atyp",
        <"at_ref",egal,v-ref,x>.
        <"at_comp",o2,v2,v-comp>.
        <"at_mod",o3,v3,v-mod>.nil>.
    <"car",
        <"car_comp",egal,v-comp,y>.
        <"carlib",o4,v4,v-lib>.nil>.nil)
```

La formulation idéale doit se limiter à mentionner les composants strictement utiles à la question. Ils doivent apparaître sous la forme de noms d'ensembles d'entités et non sous uneforme interne parfois rébarbative :

```
profil = "0617401"
clé = "APINOV"
test ?
modalité ?
```

La prise en compte est réalisée par l'interface de niveau 3. Elle utilise un ensemble de connaissances sur la structure de la base explorée. Parmi celles-ci certaines sont déjà accessibles :

nom des relations,
nom des composants,
type des composants,
nature de l'indexation.

D'autres doivent être ajoutées. Elles concernent :

- d'une part le rapport entre nom d'entité et composant interne (utilisé dans le cadre d'une relation),

- d'autre part les liens existant entre les relations qui permettent la réalisation des jointures.

II-4-1.- Choix d'une relation

Dans la question précédente l'entité "clé" peut figurer :

dans la relation des espèces sous le nom esp_cle
dans la relation des tests sous le nom car_cle,
dans la relation des tests complémentaires sous le nom carc_cle...

La détermination du lien nom d'entité,composant repose sur l'utilisation de prédicats qui expriment les connaissances de cette nature :

Le prédicat champ-lie associe le nom d'entité au nom du composant interne :

```
champ-lie("cle","carc_cle").
```

Le prédicat champ-relat associe le nom du composant au nom de la relation :

```
champ-relat("carc_cle","carc").
```

Le prédicat nom-relation permet de renommer une relation de manière à rendre plus explicite le dialogue avec l'utilisateur :

```
nom-relation("carc","tests complémentaires")
```

Pour la détermination de la relation à associer au nom d'entité, l'interface de niveau 3 propose 3 solutions qui utilisent ces connaissances :

1 - une relation déjà utilisée et liée à l'entité,

2 - la relation optimale (celle qui offre la possibilité de sélection la plus précise au sens déjà évoqué dans le cadre de l'interface de niveau 2).

3 - une liste de noms de relations possibles. L'utilisateur doit, dans ce cas, choisir dans cette liste.

II-4-2.- Etablissement des liens

L'examen de la question est réalisé de manière séquentielle. On extrait une relation de la suite des sélections et, après choix d'une relation à associer à l'entité courante, on examine les rapports entre cette sélection et les précédentes.

On utilise le prédicat liaison qui décrit le lien éventuel existant entre deux relations. Entre la relation des tests et celle des profils, il existe un lien que le prédicat liaison présente de la façon suivante :

```
liaison("car","profils",
     <"car","car_comp">.
     <"atyp","at_comp">.
     <"atyp","at_ref">.
     <"profils","pro_ref">.nil).
```

Liaison exprime que les relations des tests ("car") et des profils sont liés par l'intermédiaire de la relation des caractères atypiques ("atyp") au moyen des composants référence de test ("car_comp","at_comp") et référence de profil ("at_ref","pro_ref").

Le lien entre relations, s'il existe, doit être unique. On ne peut, en effet, admettre l'existence de cycles qui engendreraient par commutativité et transitivité une infinité potentielle de liaisons. La présence de tels cycles révèle en général l'existence d'une entité cachée qu'une analyse incomplète n'a pu révéler.

La requête initiale est ainsi progressivement compilée. Chaque nouveau composant et chaque nouvelle liaison vient compléter la sélection à soumettre à l'interface de niveau inférieur.

Voici un exemplaire du dialogue généré par l'interface et relatif à la question précédente :

sélection;

"profil" = "0617401"
"clé" = "APINOV"
"test" ? ?
"modalité" ? ? .

On examine la sélection qui porte sur le composant profil.

Les valeurs sélectionnées doivent être égales à 0617401.

Le composant profil est un composant de la relation des profils.

On examine la sélection qui porte sur le composant clé.

Les valeurs sélectionnées doivent être égales à APINOV.

La relation choisie est la relation des profils qui contient un composant du même type. Ce choix doit-il être retenu ? (o/n) : **n** .

Il faut déterminer quelle est la relation associée au composant clé.

Ce choix peut-il être laissé à l'initiative du système ? (o/n) : **o** .

La détermination va être réalisée en prenant la relation optimale : celle qui permet la sélection la plus précise.

Espèces est la relation choisir : êtes-vous d'accord (o/n) : **n**

Il faut déterminer quelle est la relation associée au composant clé.

Ce choix peut-il être laissé à l'initiative du système ? (o/n) : **n**

Dans la liste suivante, il faut sélectionner une relation :

Relation des espèces
Relation des tests
Relation des tests complémentaires
Relation des profils
Relation des espèces identifiées
Relation des caractères complémentaires
Relation des caractères atypiques.

Quel est le nom de la relation choisie : **"tests"**.

Le composant clé est un composant de la relation des tests.

On examine la sélection qui porte sur le composant test.

Le composant test est un composant de la relation des tests.
On examine la sélection qui porte sur le composant modalité.

Le composant modalité est un composant de la relation des caractères atypiques.

II-4-3.- Apprentissage

Si cette compilation interactive ne peut être évitée lorsqu'une demande particulière se présente pour la première fois, on peut faire en sorte que le travail réalisé puisse servir pour une demande analogue. A terme, ce mécanisme d'apprentissage doit permettre de compléter l'interface et de la rendre totalement autonome pour un ensemble de besoins particuliers.

Dans ce but, l'interface peut définir et ranger dans la base de connaissances un couple :

(question type, question compilée type).

Chacun des éléments de ce couple est constitué d'une manière comparable à celle évoquée dans le cadre de l'interface de niveau 2.

Schéma de question type :

profil = (valeur de profil)
clé = (valeur de clé)
test ? ?
modalité ? ?

On exprime dans ce schéma :

que les composants profil et clé doivent être connus et que la sélection porte sur un choix de valeurs égales.

que les composants test et modalité ne doivent comporter aucun critère de sélection.

Schéma de question type compilée :

```
"profils"
    "pro_profil" egal (valeur de profil) v-profil
    "pro_clé" egal (valeur de clé) v-cle
    "pro_ref" ? ? v-ref
"atyp"
    "at_ref" egal v-ref x
    "at_comp" ? ? v-comp
    "at_mod" ? ? v-mod
"car"
    "car_comp" egal v-comp y
    "car_lib" ? ? v-lib
```

On retrouve dans ce schéma les variables (valeur de profil) et (valeur de clé) qui figurent dans la question type.

Le couple est instancié chaque fois que se présente une requête comparable. L'instanciation permet l'attribution de valeurs aux variables (valeur de clé) et (valeur de profil). Les autres composants restent indéterminés.

La question compilée relative à la question type est A ainsi immédiatemment obtenue.

A la question :

```
profil = "0235674"
cle = "API20EC"
test ? ?
modalité ? ?
```

correspond immédiatement la question compilée :

```
"profils"
    "pro_profil" egal "0235674" v-profil
    "pro_clé" egal "API20EC" v-cle
    "pro-ref" ? ? v-ref
"atyp"
    "at_ref" egal v-réf x
    "at_comp" ? ? v-comp
    "at_mod" ? ? v-mod
"car"
    "car_comp" egal v-comp y
    "carlib" ? ? v-lib
```

III - CONCLUSION

Un prototype de cette interface est actuellement opérationnel sur un PC-AT modèle 2 d'une mémoire de 640 K. Il est utilisé à titre expérimental pour l'interrogation de plusieurs bases de résultats d'identification.

Bien qu'écrite dans une version de PROLOG ancienne et relativement peu performante, cette version de l'interface donne les résultats demandés dans des délais convenables (qui ne découragent pas son utilisation) mais qu'il serait souhaitable d'améliorer.

L'amélioration pourrait porter sur l'utilisation d'un PROLOG plus efficace qui puisse être étendu par des fonctions écrites en LATTICE C (ce n'est pas courant sur PC !), ou sur l'acquisition d'un matériel plus adapté et plus puissant (financement ?).

La réalisation de l'interface dans un langage de haut niveau : PROLOG permet une adaptation facile de cet outil d'interrogation à d'autres S.G.B.D. . La décomposition en niveaux présente en particulier l'avantage de limiter les modifications au niveau 1 (le seul concerné par les caractéristiques du S.G.B.D.). Cette adaptabilité pourrait à terme conduire à une approche multibase : les niveaux 2 et 3 s'appuyant sur des niveaux 1 spécifiques et relatifs à des S.G.B.D. particuliers.

CHAPITRE IV

SYSTEME EXPERT D'INTERPRETATION

DE L'IDENTIFICATION

I - ROLE ET ASPECTS GENERAUX DU SYSTEME EXPERT

Le jugement d'une identification fondé sur l'expérience et les connaissances du bactériologiste repose sur un ensemble de résultats :

- les probabilités d'appartenance aux espèces,

- l'existence et la nature des caractères atypiques.

Le chercheur en bactériologie peut qualifier l'identification comme étant excellente, bonne, douteuse,... relativement à une espèce, un genre,... à partir de critères précis. Son expérience lui permet, dans les cas douteux, de conseiller la réalisation de tests complémentaires susceptibles de lever certaines ambiguités.

Ce jugement qualitatif est établi en fonction de l'utilisation d'un kit d'identification particulier qui ne concerne qu'une certaine catégorie d'espèces. Le kit a été conçu à partir d'un état de connaissances dans le domaine concerné. Entre la conception, la réalisation industrielle et la commercialisation, il s'écoule un temps suffisamment long pour que cet état ait évolué de manière significative. De nouvelles espèces sont fréquemment découvertes, elles peuvent entrer dans le champ d'application du kit et il faut en tenir compte.

Le chercheur en bactériologie, confronté à cette évolution, précise, en fonction des caractéristiques des espèces nouvelles, quels sont les profils de réponses qui, relativement au kit d'identification, sont susceptibles d'interagir avec les nouvelles espèces. Ses connaissances lui permettent dans un tel cas de conseiller les tests supplémentaires à mettre en oeuvre pour préciser l'identification.

Le rôle du système expert est d'assimiler ce type de connaissance et donc de rendre compte des évolutions que le système d'identification proprement dit est lui-même incapable d'intégrer parce qu'il repose pour une part essentielle sur l'utilisation de kits miniaturisés et commercialement diffusés dont la constitution est figée pour une durée assez longue.

II - LA BASE DE FAITS

Les résultats d'identification qui alimentent la base de faits proviennent de 2 sources :

- la base de données pour les cas répertoriés,
- un logiciel d'identification intégré au système pour les nouveaux cas ou les cas spéciaux.

La base de faits à produire est formée des prédicats :

profil(profil identifié)
espèce-identifiée (nom d'espèce, probabilité)
probmax (probabilité)
espèce-nouvelle (nom d'espèce, catégorie)
car-atypique (nom de test, modalité de réponse)
car-complémentaire (nom de test, modalité de réponse)

On y trouve :

- le profil de réponses obtenu par l'utilisateur d'un kit d'identification particulier (prédicat profil),
- la liste des espèces d'appartenance possibles et les probabilités associées (prédicat espèce-identifiée);
- la liste des espèces nouvelles que le kit d'identification ne prend pas en compte et qui pourraient concerner la souche étudiée,
- la liste des caractères atypiques du profil (modalité de réponses anormales pour l'espèce d'appartenance la plus probable),
- la liste des tests complémentaires et les modalités de réponse qui précisent une identification que les seuls tests du kit ne permettent pas d'obtenir correctement (prédicat car-complémentaire).

La base de faits ainsi constituée contient l'ensemble des éléments nécessaires et suffisants à la documentation et l'interprétation d'une identification. Il n'y a pas actuellement de nécessité à demander des informations complémentaires au cours de la résolution.

II-1.- La base de faits est produite à partir de la base de données.

Pour générer les prédicats de la base de faits, on utilise un sous-ensemble de l'interface d'interrogation PROLOG-INFORMIX.

Le niveau 3 de l'interface a été utilisé pour obtenir une formulation utilisable par le niveau 2 des interrogations de la forme suivante :

```
1  "profil" = valeur de profil
   "cle" = valeur de clé
   "reference de profil" ? ?
   "espece" ? ?
   "probabilite" ? ?

2  "reference de profil" = valeur de référence
   "test" ? ?
   "modalite" ? ?

3  "reference de profil" = valeur de référence
   "test complementaire" ? ?
   "modalite complementaire" ? ?
```

L'interrogation 1 permet pour un profil de réponse et un kit d'utilisation donné d'obtenir les noms et probabilités des espèces identifiées.

L'interrogation 2 retrouve dans la base la liste des tests atypiques et leurs modalités.

L'interrogation 3 réalise le même traitement pour les test complémentaires.

C'est la forme "compilée" de ces interrogations qui est utilisée par les interfaces de niveau 2 et 1. Les réponses obtenues sont alors mises en forme pour produire les prédicats de la base de faits.

II-2.- La base de faits est produite par un logiciel d'identification.

Si le profil de réponses de la souche bactérienne étudiée ne figure pas dans la base, l'utilisateur du système peut faire appel à un logiciel d'identification pour produire les résultats demandés.

La conception de ce logiciel repose sur les principes énoncés au Chapitre II. Il calcule les probabilités d'appartenance aux espèces d'un kit d'identification et détermine quels sont les caractères atypiques.

Il est directement utilisable à partir de prédicats évaluables ajoutés à l'interpréteur et pour des raisons de compatibilité il a été réécrit en langage C.

II.-2-1.- Les prédicats évaluables d'identification

Ils sont au nombre de 5.

- ident-init initialise les variables de l'intepréteur pour une nouvelle identification.

- caractère-complémentaire(x-prof,x-test,x-mod) : ajoute un caractère complémentaire (x-test,x-mod) à la liste de ces caractères. Dans ce cas le profil (x-prof) contient les réponses supplémentaires correspondantes. Ce prédicat doit être évalué pour chaque test complémentaire avant toute tentative d'identification.

- identification-espèce (x-prof,i,x-esp,x-prob) : donne relativement au profil x-prof la i-ième probabilité (x-prob) d'identification à une espèce (x-exp). Ce prédicat, comme les suivants, lance, si cela n'a pas déjà été réalisé, l'ensemble des calculs relatifs à une identification. Les résultats sont rangés dans une série de variables de l'interpréteur et peuvent être utilisés ultérieurement par d'autres appels aux prédicats d'identification pourvu que l'on reste dans le même contexte.

- caractère-atypique (x-prof,i,x-numc) : produit le numéro (x-numc) du i-ème test aytpique d'une identification (x-prof). Si le numéro d'ordre (i) excèse le nombre de tests atypiques rencontrés, le prédicat envoie la valeur 0 comme numéro de test.

- proba-réponse (x-profi,x-numc,x-prob) : donne la probabilité x-prob d'apparition de la modalité du test de numéro x-numc qui figure dans le profil de réponses (x-prof) lorsque l'on considère l'espèce la plus probable.

Le programme PROLOG d'identification utilise ces prédicats pour constituer la base de faits du système expert comme si elle avait été extraite de la base par l'interface S.G.B.D. La réalisation de ce travail nécessite l'utilisation d'informations complémentaires sur la constitution d'un kit.

II-2-2.- Les prédicats de définition d'un kit.

Le calcul des probabilités qu'exige une identification est réalisé à partir de données élémentaires qui sont les probabilités de réponse à une modalité d'un test pour une espèce donnée. Ces valeurs sont extraites d'un fichier relatif qui concerne outre les tests propres à un kit ceux qui peuvent être utilisés comme tests complémentaires.

Les probabilités sont rangées test par test pour l'ensemble des espèces.

Les prédicats évaluables qui utilisent ce fichier donnent comme résultat les numéros d'ordre des tests et espèces tels qu'ils figurent dans le fichier. Les prédicats de définition d'un kit réalisent la correspondance nécessaire entre noms et numéros.

- fichier-kit (x-nomfich,x-kit) établit le lien entre un fichier de probabilités et un kit d'identification.

- espèce (x-nume,x-nomesp) associe un numéro d'ordre et un nom d'espèce appartenant au kit.

- test(x-numc,x-numk,x-nomtest) réalise l'association correspondante pour les tests d'un kit. Il donne par l'intermédiaire de x-numk le numéro d'ordre du test dans le kit (il peut être différent du numéro d'ordre dans le fichier : x-numc). Les tests complémentaires utilisables sont également fournis par ce prédicat qui leur attribue un numéro d'ordre x-numk égal à 0.

- nb-tests-kit(x-nbt) instancie la variable x-nbt avec le nombre de tests qui composent un kit.

III - LA BASE DE CONNAISSANCES

La base de connaissances est constituée des règles qui permettent l'interprétation des résultats d'une identification. Ces règles n'utilisent que les éléments fournis par la base de faits. Il n'est pas nécessaire d'entrer interactivement des informations complémentaires au cours de la résolution.

III-1.- Les composants d'une règle

Les éléments susceptibles d'intervenir sont :

- la forme d'un profil

Certains profils de réponses peuvent interférer avec certaines espèces nouvelles. Ils présentent parfois des caractéristiques précises qui correspondent à des réponses particulières à certains tests du kit. Certaines règles utilisent ces réponses particulières.

Exemple :
Si la réponse au test ORNITHINE-DECARBOXYLASE est + alors déduire que le profil est orniplus.

D'autres règles parmi les plus nombreuses concernent un ensemble assez vaste de profils. Il faut dans ce cas pouvoir donner une forme synthétique à cet ensemble pour éviter une multiplication trop importante de ces règles. L'implantation à l'aide du langage PROLOG permet beaucoup de souplesse et facilite la résolution de ce type de problème très spécifique du domaine d'application. On utilise un prédicat "CORRESPOND" qui compare le profil étudié à une liste où sont regroupés l'ensemble des cas possibles.

Exemple :
Si le profil étudié correspond à (4 6) (2 6) (0 1 2 3) (1 2 3) 0 (0 1) (0 1) alors déduire que le profil est du type cdav.

- L'identification possible à une espèce
  Cette caractéristique est souvent utilisée en complément d'autres propriétés.

Exemple :
Si le profil est orniplus et si la souche est identifiable à l'espèce E.AGGLOMERANS-1 alors déduire que le profil est probablement eag1.

- L'appartenance possible à une espèce nouvelle.

C'est un facteur qui intervient dans l'expression de nombreuses règles. On distingue 3 catégories de souches bactériennes qui peuvent appartenir à des espèces nouvelles.

La catégorie 1 est formée de souches qui ont même profil de réponses qu'une souche de collection de l'espèce nouvelle. Elles forment la catégorie la plus connue.

La catégorie 2 regroupe les souches dont l'appartenance à une espèce nouvelle est probable sur la base de tests du kit.

La catégorie 3 comporte les souches dont l'identification à une espèce nouvelle est possible bien que présentant certains aspects divergents.

Voici deux règles de ce type extraites de la base de connaissances API20EC :

* si le profil est de type cdav et si aucune espèce nouvelle n'est possible, alors déduire que le profil est du type cdavseul.

* si l'appartenance à une espèce nouvelle est possible (catégorie 2) alors conclure que le profil a déjà été rencontré et qu'il y a identification probable à l'espèce nouvelle.

- La présence ou l'absence de caractères atypiques.

Elle peut servir à nuancer la qualité d'un résultat :

Si aucune espèce nouvelle n'est possible et si aucun test atypique n'est présent, alors déduire que l'identification est normale.

- La présence de caractères complémentaires

- La probabilité du genre

Dans le cas où plusieurs espèces sont probables, certaines de ces espèces peuvent appartenir à un même genre. La probabilité du genre n'est pas un résultat enregistré dans la base des identifications. Le calcul est réalisée par un prédicat évaluable ajouté à l'interpréteur. Si la probabilité de l'espèce identifiée est <990 et si la probabilité du genre est >990 alors conclure à une très bonne identification au genre.

III-2.- Le type des règles

Ces exemples illustrent et mettent en évidence les différentes formes de règles. On distingue 2 grandes catégories :

- Les règles déuctives qui permettent la définition de résultats intermédiaires qui peuvent être utilisés dans d'autres règles. Dans les règles précédemment citées on trouve, en particulier : "le profil est du type cdav" ou "le profil est probablement eag1". Elles comportent des variables du système explicitement cités comme "profil ou impliqués comme "probabilité" dans "probablement eag1" et "espèce identifiable". Ces variables peuvent être instanciées lors de la résolution en fonction des caractéristiques de l'identification et des résultats déjà acquis. Les valeurs doivent être transportées dans une

règle (entre condition et résultat) et entre règles.

- Les règles concluantes. Dans ce cas la conclusion est une phrase où peuvent apparaître certains mots qui ont la signification de variables. Leur valeur provient de la base de faits via les conditions vérifiées et les règles déductives utilisées lors de la résolution.

IV - L'INTERFACE UTILISATEUR POUR LA GESTION DE LA BASE DE CONNAISSANCES

Le rôle de cette interface est de représenter le plus fidèlement possible le langage utilisé par le bactériologiste dans sa pratique quotidienne. L'entrée d'une règle, l'édition de son contenu, la suppression de règle doivent s'exprimer dans la syntaxe du spécialiste. PROLOG est un outil adapté à la définition d'une telle interface et répond bien à ces exigences particulières.

Son fonctionnement est illustré par les exemples de règles qui ont été mentionnés dans le chapitre précédent. On montre successivement l'entrée d'une règle et l'édition dse son contenu.

IV-1.- Entrée des règles

Pour entrer une nouvelle règle on utilise le prédicat "AJOUTER". La règle est analysée. En cas de succès, elle est traduite dans une forme interne et ajoutée à la base de connaissance. Le lecteur intéressé par les aspects techniques peut consulter l'annexe "Système Expert".

```
>ajouter;
si le test est ORNITHINE-DECARBOXYLASE et
la modalité est +
alors profil orniplus.

si le profil est
(4 6) (2 6) (0 1 2 3) (1 2 3) 0 (0 1) (0 1)
alors profil'cdav.

>ajouter;
si profil orniplus et
l'espèce identifiable est E_AGGLOMERANS-1
alors profil probabilité eag1.

>ajouter;
si profil cdav et
pas d'espèce nouvelle
alors profil cdavseul.

>ajouter;
si quelque soit le profil et
appartenance possible à une espèce nouvelle
alors conclure profil déjà rencontré et
identification probable à espèce nouvelle.

>ajouter;
si la probabilité maximum est  <990  et
la probabilité du genre est  >990
alors conclure très bonne identification à genre.
```

```
>ajouter;
si profil probabilité eag1 et profil cdav
alors conclure
appartenance possible à  C_DAVISAE pour profil.
```

IV-2.- Edition des règles

Les règles sont implantées sous forme de faits PROLOG. Le fait "REGLE" coprrespond aux règles déductives, le fait "REGLE-FINALE" aux règles concluantes. Dans chacun de ces cas on trouve dans le prédicat deux arguments : la liste des conditions et la conclusion. Les règles ainsi représentées peuvent être éditées sous une forme intelligible à l'aide des prédicats "LISTER-REGLE" , "LISTER-SUIVANT" ou "LISTER-CONCLUSION" ainsi que le montrent les exemples suivants :

```
>lister-règle(1);

SI
? cdav
il n'est pas vrai que : espèce nouvelle ? de catégorie ?
ALORS DEDUIRE
? cdavseul

>lister-suivante;

SI
? orniplus
espèce identifiable E.AGGLOMERANS-1
avec la probabilité ?
ALORS DEDUIRE
? ? eag1
```

```
>lister-suivante;

SI
profil ? correspond à
(4.6.nil).(2.6.nil).(0.1.2.3.nil).
(1.2.3.nil).0.(0.1.nil).(0.1.nil).nil
ALORS DEDUIRE
? cdav

>lister-suivante;

SI
le caractère est ORNITHINE-DECARBOXYLASE +
modalité de test +
ALORS DEDUIRE
? orniplus

>lister-conclusion(1);

SI
? ? eag1
? cdav
ALORS CONCLURE
appartenance possible à C.DAVISAE pour ?

>lister-suivante;

SI
probabilité ? de l'espèce identifiée ? est infer 990
probabilité ? du genre ? est super 990
ALORS CONCLURE
très bonne identification à ?
```

```
>lister-suivante;

SI
profil ? correspond à ?
espèce nouvelle
? de catégorie 2
ALORS CONCLURE
? déjà rencontré et identification probable à ?
```

Le cadre syntaxique a été défini en fonction des besoins du bactériologiste de la façon la plus exhaustive possible. En cours de développement, il est apparu cependant la nécessité d'augmenter et de modifier la syntaxe établie à l'origine. La structure du module de communication a été étudiée pour faciliter ce type d'intervention qui reste cependant du domaine du spécialiste.

Ajouter une condition prédéfinie à une règle s'effectue en intervenant simplement au niveau analyse, au niveau variable, au niveau interprétation et au niveau édition. Chacun de ces niveaux est réalisé par un prédicat spécifique :

- ANA-CONDA pour l'analyse,
- VARIABLE pour la définition d'une variable,
- EST pour l'interprétation d'une règle prédéfinie,
- EXPL pour l'édition.

V - LE MOTEUR DE RESOLUTION

Les règles qui constituent la base de connaissance sont prises en compte par un moteur de résolution entièrement écrit en PROLOG. Le moteur fonctionne en chaînage avant. Il utilise la base de faits constituée à partir de la base ou du logiciel d'identification et l'ensemble des propriétés déduites, pour établir de nouveaux résultats. La base de faits s'enrichit au fur et à mesure des déductions sous la forme de prédicats "DEDUIT". Si une conclusion est atteinte, elle est enregistrée par l'intermédiaire d'un prédicat "CONCLUSION".

Le moteur est mis en marche par le prédicat "INTERROGATION" qui demande à l'utilisateur les éléments d'information indispensables à la constitution de la base de faits initiale :

```
>interrogation;
Quel est le profil à recherche ? :
"6213011"
Extension complémentaire ? :
" "
Recherche dans la base (o/1) ? : 0
```

Les résultats de l'identification demandée sont édités :

```
PROFIL : 6213.11  CLE : API20EC
```

L'interprétation peut commencer. Le moteur d'inférence produit l'ensemble des conclusions possibles sauf s'il rencontre un type de règle particulier : les règles terminales qui arrêtent immédiatement la résolution. Les conclusions sont éditées dès qu'elles sont obtenues

```
: appartenance possible à C.DAVISAE pour 6213011
vérifier les tests MELIBIOSE(-),ARABINOSE(-) , RHAMNOSE(-)
pas d'identification possible avec la base API 20 EC
```

Les résultats intermédiaire ont été gardés ainsi que l'ensemble des conclusions atteintes.

```
déduit(<cdav,nil>) ->;
déduit(<esak,nil>) ->;
déduit(<eag1,x1714.6.nil>) ->;
déduit(<orniplus,x1670.nil>) ->;
déduit(<cdav,"6213011".nilm>) ->;

conclusion ("identique"."a"."une"."souche"."de".
   "collection"."de"."C.DAVISAE".nil) ->;
conclusion ("pas"."d"."'"."identification"."possible".
   "avec"."la"."base"."API".20."EC".nil) ->;
conclusion("verifier"."les"."tests"."MELIBIOSE"."("."-".")".
   ","."ARABINOSE"."("."."")"."."."RHAMNOSE"."("."-".")".
   nil) ->;
conclusion("appartenance"."possible"."a"."C.DAVISAE".
   "pour"."6213011".nil) ->;
```

Il est alors possible de demander une trace du raisonnement utilisé. On utilise un moteur spécifique qui fonctionne en marche arrière. On part des conclusions et on remonte par les faits déduits jusqu'aux faits initiaux. Chaque règle utilisée est éditée avec les valeurs obtenues pour les variables pendant la résolution.

```
>pourquoi;

Le résultat : .
pas d'identification possible avec la base API 20 EC
a été démontré car :
probabilité 994 de l'espèce identifiée E.SAKAZAKII
est super 990
test atypique MELIBIOSE de modalité -
```

Le résultat :
vérifier les tests MELIBIOSE (-) ,
ARABINOSE (-), RHAMNOSE (-)
a été démontré car :
cdav
esak

La propriété intermédiaire :
cdav
a été démontrée car
profil ? correspond à
(4.6.nil).(2.6.nil).(0.1.2.3.nil).
(1.2.3.nil).0.(0.1.nil).(0.1.nil).nil

La propriété intermédiaire :
esak
a été démontrée car
espèce identifiée E.SAKAZAKII avec la probabilité ?

Le résultat :
appartenance possible à C.DAVISAE pour 6213011
a été démontrée car :
6213011 6 eag1
6213011 cdav

La propriété intermédiaire :
6213011 6 eag1
a été démontrée car 6213011 orniplus
espèce identifiable E.AGGLOMERANS-1
avec la probabilité 6

La probabilité intermédiaire :
6213011 orniplus
a été démontrée car
le caractère est ORNITHINE-DECARBOXYLASE +
modalité de test +

La propriété intermédiaire :
6213011 cdav
a été démontrée car
profil 6213011 correspond à
(4.6.nil).(2.6.nil).(0.1.2.3.nil).
(1.2.3.nil).0.(0.1.nil).(0.1.nil).nil

VI - CONCLUSION

Le Système Expert est actuellement complétement écrit. Il est testé sur un ensemble réduit d'une cinquantaine de règles qui s'enchaînent jusqu'à une profondeur de 3. Les performances obtenues permettent d'envisager la constitution d'une version plus complète qui s'enrichira et se modifiera par la prise en compte de connaissances nouvelles. Les outils nécessaires ont été développés dans ce sens.

L'utilisation de PROLOG comme langage de génération de Système Expert a été concluante. Au cours du développement, ce langage a montré ses aptitudes à résoudre et traiter les aspects particuliers du système à concevoir : interface utilisateur spécifique, utilisation d'une base de données... Il a rapidement permis la définition d'un prototype qui a pu facilement être adapté conformément aux souhaits du bactériologiste. La méthodologie utilisée, la structuration du programme en niveaux fonctionnels, l'interfaçage avec un SGBD, peuvent être réutilisés et servir de cadre à la mise en place de nouveaux systèmes experts dans des domaines différents.

PROLOG a aussi montré certaines limites. Il n'est pas raisonnable de l'utiliser pour mettre en oeuvre certains calculs importants (PROLOG n'est pas un langage de programmation scientifique). Ces limites ont été dépassées en ajoutant à l'interpréteur un ensemble de prédicats évaluables écrits dans un langage plus approprié. Cette possibilité offerte par PROLOG II est fondamentale.

Les performances, jugées acceptables, pâtissent de l'utilisation d'une version de PROLOG ancienne et peu rapide. Elles pourraient être multipliées par 5 ou plus par l'emploi d'interpréteurs ou compilateurs plus récents. Ceux-ci ne permettent malheureusement pas l'interfaçage sur IBM-AT avec le SGBD INFORMIX.

A N N E X E S

SYSTEME EXPERT
EN IDENTIFICATION BACTERIENNE

Le système expert d'interprétation de résultats d'identifications bactérienne est de conception classique. Il comporte :

- une base de faits formée des résultats d'identification,
- une base de connaissance : les règles d'interprétation,
- un moteur d'inférence qui produit en marche avant un ensemble de résultats à partir des bases prédédentes,
- une interface utilisateur dont les fonctions principales sont : la trace du raisonnement utilisé pour obtenir l'ensemble des résultats et la gestion des régles de la base de connaissance.

Son originalité tient à son intégration avec une base de données de résultats d'identifications et un logiciel d'identification. La base de faits est constituée à partir d'éléments extraits du SGBD par l'intermédiaire d'une interface spécifique ou produite par le logiciel d'identification.

Ce système expert est entièrement écrit en PROLOG II et l'on trouvera dans ce qui suit le détail du programme et les commentaires assortis.

```
interrogation ->
        exl("Quel est le profil à rechercher  ? :")
        in-chaine(p)
        exl("Extension complémentaire ? :")
        in-chaine(x-ext)
        question(p,x-ext)
        edition
        moteur;
```

Le prédicat INTERROGATION permet d'amorcer le processus d'identification et d'interprétation du résultat. On demande le profil de la souche à identifier relatif a un kit qui a servi à l'identification. Le prédicat QUESTION oriente vers l'utilisation de la base ou du logiciel d'identification. On obtient dans les deux cas la liste des espèces identifiées et leurs probabilités d'appartenance, la liste des espèces nouvelles pour lesquelles une identification est possible, la liste des caractères atypiques et la liste des caractères complémentaires. Avec cet ensemble de résultats QUESTION construit une base de faits PROLOG formée des prédicats :

```
PROFIL(x-prof)
PROBMAX(x-prob)
ESPECE-IDENTIFIEE(x-esp,x-prob)
```

```
ESPECE-NOUVELLE(x-esp,x-cat)
CAR-ATYPIQUE(x-car,x-mod)
CAR-COMPLEMENTAIRE(x-car,x-mod)
```

Dans ce qui précède la variable PROLOG :

x-prof représente un profil à identifier,
x-prob une probabilité,
x-esp un nom d'espèce,
x-cat une catégorie d'espèce nouvelle,
x-car un nom de test atypique,
x-mod une modalité de réponse à un test.

Voici à titre d'exemple la base de faits produite à partir du profil "1616400" pour le kit "API20EC". On notera l'absence de caractères complémentaires :

```
profil("1616400") ->;
probmax(649) ->;
espece-identifiee(
     "E.AMNIGENUS                    ",1) ->;
espece-identifiee(
     "E.AGGLOMERANS-1                ",2) ->;
espece-identifiee(
     "E.ADECARBOXYLATA               ",3) ->;
espece-identifiee(
     "K.ASCORBATA                    ",30) ->;
espece-identifiee(
     "K.CRYOCRESCENS                 ",315) ->;
espece-identifiee(
     "B.AGRESTIS                     ",649) ->;
espece-nouvelle("E.VULNERIS                          ",2) ->;
car-atypique("LYSINE-DECARBOXYLASE              ","+") ->;
```

Le prédicat EDITION réalise l'impression des résultats. Le prédicat MOTEUR utilise les règles du système d'interpétation pour inférer une conclusion relative à la qualité du résultat.

```
edition -> page  profil(x-p,x-ext)
     exm("PROFIL : ") exm(x-p) exm(" ") exm(x-ext)
     ligne ligne
     exm("Identification à l'espèce :") ligne ligne
     probmax(p)
     espece-identifiee(e,p) exm(e) exm(" ") ex(p)
     ligne ligne
     ex1("Autres identifications possibles :")
     ligne edit-esp(p)
     ligne ex1("Caractères atypiques :")
     ligne edit-at
     ligne ex1("Caractères complémentaires :")
     ligne edit-comp /;
edition ->;
```

```
edit-esp(p) ->
     espece-identifiee(e,p') dif(p,p')
     exm(e) exm("  ") ex(p') ligne echec;
edit-esp(p) ->;

edit-at ->
     car-atypique(c,m) exm(c) exm("  ") exm(m)
     ligne echec;
edit-at ->;

edit-comp ->
     car-complementaire(c,m) exm(c) exm("  ") exm(m)
     ligne echec;
edit-comp ->;

moteur -> regle(c,a)
     non(deduit(a))
     verif-cond(c) ajout(<deduit(a),nil>) / moteur;
moteur -> regle-finale(c,a)
     verif-cond(c)
     edit-predi(a) ajout(<conclusion(a),nil>) /;
moteur -> regle-terminale(c,a)
     verif-cond(c)
     edit-predi(a) ajout(<conclusion(a),nil>) echec;
moteur ->;

verif-cond(nil) ->;
verif-cond(c.l-c) -> verif-condi(c)
     verif-cond(l-c);

verif-condi(c) -> c;
verif-condi(c) -> deduit(c);
```

MOTEUR est le prédicat qui décrit en PROLOG le mécanisme de résolution. Dans la base de connaissance du système on trouve trois types de règles :

Les règles qui permettent de déduire un résultat intermédiaire. Elles sont définies à l'aide du prédicat REGLE.

Les règles qui concluent sur un résultat qualifiant l'interprétation. Elles se trouvent dans le prédicat regle-terminale.

Les règles qui concluent de manière définitive et qui arrètent la résolution. Elles sont définies à l'aide du prédicat regle-finale.

Toutes ces règles comportent deux arguments. Le premier contient la liste des conditions d'application, le second la conclusion. Les conditions d'application sont établies par le

prédicat VERIF-COND.

Le moteur fonctionne en marche avant. Il utilise les règles déductives pour inférer un ensemble de résultats intermédaires. Chacun d'eux est ajouté à la base de connaissance par l'intermédiaire d'un prédicat DEDUIT. La première règle du paquet MOTEUR est réutilisée récursivement tant qu'une règle déductive permet d'inférer un résultat intermédiaire nouveau. Les règles 2 et 3 concernent les regles concluantes. Une utilisation réussie de la règle 2 amène une conclusion définitive et l'arret de la résolution.

```
pourquoi -> conclusion(a)
     regle-finale(c,a)
     verif-cond(c)
     ligne
     exl("Le resultat definitif :") edit-predi(a)
     exl("a ete demontre car :")
     edit-cond(c) ligne / parceque(c);
pourquoi -> conclusion(a) regle-terminale(c,a)
     verif-cond(c)
     ligne
     exl("Le resultat :") edit-predi(a)
     exl("a ete demontre car :")
     edit-cond(c) ligne parceque(c) echec;
pourquoi ->;

parceque(nil) -> /;
parceque(p.l-c) -> deduit(p) expl(p,m)
     ligne
     exl("La propriete intermediaire :")
     edit-predi(m)
     exl("a ete demontree car") regle(c,p)
     edit-cond(c) /
     parceque(c) parceque(l-c);
parceque(e.l-c) -> parceque(l-c);
```

POURQUOI est un moteur d'inférence en marche arrière qui partant de la conclusion obtenue retrace en l'expliquant le raisonnement utilsé. On part des règles concluantes en recherchant en premier lieu celle, si elle existe, qui permet une conclusion définitive. On examine par le prédicat PARCEQUE les causes déclanchantes qui sont ou des faits établis ou le résultat de déductions antérieures. Dans ce dernier cas on réutilise de manière récursive le prédicat POURQUOI pour compléter la trace du raisonnement.

```
lister-regle(n) -> tete(regle) moins(n,1,n')
     descendre(n') tampon-neuf(prendre-regle(r))
     eg(r,regle(c,a)) editer-regle(r);

lister-conclusion(n) -> tete(regle-terminale)
     moins(n,1,n') descendre(n')
```

```
        tampon-neuf(prendre-regle(r))
        eg(r,regle-terminale(c,a))
        editer-regle(r);

lister-conclusion-definitive(n) -> tete(regle-finale)
        moins(n,1,n') descendre(n')
        tampon-neuf(prendre-regle(r))
        eg(r,regle-finale(c,a))
        editer-regle(r);

lister-suivante -> descendre(l)
        tampon-neuf(prendre-regle(r)) editer-regle(r);
```

Les prédicats LISTER permettent d'afficher à l'écran le contenu de chacune des règles déductives pour chacune des catégories. L'argument permet de se positionner à l'intérieur d'un paquet de règles sur la règle d'ordre n. Le prédicat EDITER-REGLE utilise les prédicats EDIT-COND et EDIT-PRED pour éditer les conditions et la conclusion d'une règle d'une catégorie donnée. On extrait le texte explicatif relatif à une condition à l'aide du prédicat EXPL qui associe à la représentation interne d'une règle sa représentation externe.

```
prendre-regle(r) -> sortie("tampon")
        lister(l) sortie("console")
        entree("tampon")
        in(r) entree("console");

editer-regle(regle(c,a)) -> ligne
        exl("SI") edit-cond(c) exl("ALORS DEDUIRE")
        expl(a,m) edit-predi(m) ligne /;
editer-regle(regle-terminale(c,a)) -> ligne
        exl("SI") edit-cond(c) exl("ALORS CONCLURE")
        edit-predi(a) ligne /;
editer-regle(regle-finale(c,a)) -> ligne
        exl("SI") edit-cond(c)
        exl("ALORS CONCLURE DEFINITIVEMENT")
        edit-predi(a) ligne /;
editer-regle(fin-du-monde) -> ligne
        exl("FIN DE LA LISTE DES REGLES")
        ligne monter(l);

edit-cond(nil) -> /;
edit-cond(non(p).l-c) ->
        exm("il n'est pas vrai que : ")
        edit-pred(p) / edit-cond(l-c);
edit-cond(p.l-c) -> edit-pred(p) /
        edit-cond(l-c);

edit-pred(est(p)) -> / expl(p,m) edit-predi(m);
edit-pred(p) -> expl(p,m) / edit-predi(m);
```

```
edit-predi(nil) -> ligne /;
edit-predi(a.l-a) -> edit-conc(a) exm(" ") /
     edit-predi(l-a);

edit-conc(a) -> exm(a) /;
edit-conc(a) -> libre(a) / exm("?");
edit-conc(a) -> ex(a);

expl(testat(c.m.nil),
  "test atypique".c."de modalite".m.nil) ->;
expl(testcomp(c.m.nil),
  "test complementaire".c."de modalite".m.nil) ->;
expl(carac(c.m.nil),
  "le caractère est ".c." ".m.nil)->;
expl(modat(c.m.nil),
  "modalite de test atypique".m.nil) ->;
expl(modcomp(c.m.nil),
  "modalite de test complementaire".m.nil) ->;
expl(modkit(c.m.nil),"modalite de test".m.nil) ->;
expl(genr(g.nil),"genre".g.nil) ->;
expl(espid(e.p.nil),
  "espece identifiee".e."avec la probabilite".p.nil) ->;
expl(esp(e.p.nil),
  "espece identifiable".e.
  "avec la probabilite".p.nil) ->;
expl(espn(e.k.nil),
  "espece nouvelle".e."de categorie".k.nil) ->;
expl(probg(g.o.p.v.nil),
  "probabilite".p."du genre".g."est".o.v.nil) ->;
expl(probm(e.o.p.v.nil),"probabilite".p.
  "de l'espece identifiee".e."est".o.v.nil) ->;
expl(prob(e.o.p.v.nil),"probabilite".p.
  "de l'espece identifiable".e."est".o.v.nil) ->;
expl(prof(p.l-p.nil),
   "profil".p."correspond a".l-p.nil) ->;
```

La gestion des règles est complétée par le prédicat SUPPRIMER-REGLE et le prédicat AJOUTER.

```
supprimer-regle -> supprimer(l);
```

AJOUTER permet l'adjonction d'une règle nouvelle dans la base de connaissance. Il autorise l'entrée de la règle dans un langage proche de celui du Bactériologiste. Son vocabulaire et sa grammaire sont explicités dans ce qui suit.

```
<regle> :- si <liste-conds> alors <fin>
<liste-conds> :- <cond> et <liste-conds>
                 / <cond>
<fin> :- conclure definitivement <conclu>
         / conclure <conclu>
         / <concl>
```

```
<cond> :- pas de <conda>
          / pas d' <conda>
          / il n'est pas vrai que <conda>
          / <conda>
<conclu> :- <mod> <conclu>
            / <ponct> <conclu>
            / <entier> <conclu>
            / <nom> <conclu>
            / <variable> <conclu>
            / <mot> <conclu>
<concl> :- <liste-var> <ident> <liste-var>
<conda> :-  le test atypique est <nom>
            / test atypique
            / le test complementaire est <nom>
            / test complementaire
            / le test est <nom>
            / la modalite atypique est <mod>
            / la modalite complementaire est <mod>
            / la modalite est <mod>
            / le genre est <nom>
            / l'espece identifiee est <nom>
            / l'espece identifiable est <nom>
            / l'espece nouvelle est <nom>
            / espece nouvelle
            / identique a une souche de collection de
              <nom>
            / identique a une souche de collection
            / appartenance possible a une espece nouvelle
            / appartenance possible a <nom>
            / rapprochement possible d'une espece nouvelle
            / rapprochement possible de <nom>
            / la probabilite du genre est <oper> <entier>
            / la probabilite maximum est <oper> <entier>
            / la probabilite est <oper> <entier>
            / le profil est <prof>
            / quelque soit le profil
            / <pred>
<pred> :- <liste-var> <ident> <liste-var>
<liste-var> :- <variable> <liste-var> / <vide>
<prof> :- <comp> <prof>
          / <comp>
<comp> :- ( <liste> )
          / <entier>
          / <var>
<liste> :- <entier> <liste>
           / <entier>
<var> :- p <entier>
        / p
<ident> :- <mote> - <ident>
           / <mote>
<mote> :- <mot> <entier>
          / <mot>
<nom> :- <motmaj> - <noms>
```

```
          / <motmaj> _ <noms>
          / <motmaj>
<noms> :- <nomi> - <noms>
          / <nomi> _ <noms>
          / <nomi>
<nomi> :- <entier>
          / <motmaj>
<motmaj> :- <maj> <motmaj>
            / <maj>
<ponct> :-  ' / , / ; / : / ( / ) / %
<oper> :- < / > / <= / >= / =
<mod> :- + / -
<mot> :- <minuscule> <mot>
         / <minuscule>
<maj> :- A/B.../Z
<minuscule> :- a/b.../z
```

L'analyse syntaxique d'une règle du système expert se programme facilement en Prolog à l'aide de règles qui correspondent à la grammaire.

La phrase à analyser est entrée sous forme d'une liste à l'aide du prédicat Prolog II "IN-PH". A chaque étape on réalise l'analyse d'une tête de liste qui correspond à une unité syntaxique et on produit une queue de liste qui reste à analyser.

A la première règle de la grammaire correspond en Prolog le prédicat "ANA-REGLE(r,r-i,a)". Il analyse la règle r, produit la règle compilée r-i et génère un arbre de variables a. La règle doit commencer par le mot "si" se poursuivre par une liste de conditions analysée par le prédicat "ANA-LISTE-CONDS". Elle doit continuer par le mot "alors" et se terminer par une conclusion anlysée par le prédicat "ANA-FIN".La règle compilée r-i est un "fait PROLOG" qui se présente sous l'une des formes suivantes :

regle(liste de conditions, conclusion intermédiaire),
regle-finale(liste de conditions, conclusion finale),
regle-terminale(liste de conditions, conclusion définitive)

La liste des conditions d'application de la règle du système expert est formée de prédicats "EST", de conclusions intermédaires ou de la négation de ces éléments.

Un prédicat "EST" correspond à une situation particulière décrite en Prolog à l'aide de prédicats de la base de faits ou de prédicats prédéfinis en Prolog. Il peut comporter des variables propres au système expert.

Une conclusion intermédiaire provient d'une règle non terminale. Elle peut utiliser des variables du système expert placées avant ou aprés l'identificateur qui la caractérise (la notation est quelconque : préfixée, postfixée ou infixée).

Les variables du système expert sont décrites par un

prédicat "VARIABLE". Leur nom est formé d'une suite de mots, on leur associe une représentation interne.

nom de variable	représentation interne
test	testkit
modalite	modt
test atypique	testat
modalite atypique	modat
test complementaire	testcomp
modalite complementaire	modcomp
espece identifiee	espid
espece identifiable	esp
espece nouvelle	espn
espece nouvelle1	espn1
espece nouvelle2	espn2
espece nouvelle3	espn3
genre	genr
probabilite maximum	probm
probabilite genre	probg
probabilite	prob
profil	prof

Lors de l'analyse d'une règle à chaque rencontre d'une variable on explore l'arbre des variables à la recherche d'un couple <repésentation interne,variable prolog>. Cela permet d'unifier entre elles toutes les occurrences des variables Prolog attachées à une même représentation interne. Ce travail est réalisé par le prédicat "DANSA".

```
ajouter -> positionner in-ph(r)
     ana-regle(r,r-i,a) ajout(<r-i,nil>);

ana-regle(mt-si.l,r,a) ->
     ana-liste-conds(l,mt-alors.l-r,l-p,a)
     ana-fin(l-r,".".nil,l-p,r,a) /;
ana-regle(l,r,a) -> ligne
     exm("UNE REGLE  DOIT S'ECRIRE SOUS LA FORME :")
     ligne
     exm("
     SI condition [et condition ...] ALORS conclusion")
     ligne echec;

ana-fin(mt-conclure.mt-definitivement.l,l-r,l-p,
     regle-finale(l-p,p),a) ->
     ana-conclu(l,l-r,p,a) /;
ana-fin(mt-conclure.l,l-r,l-p,
     regle-terminale(l-p,p),a) ->
     ana-conclu(l,l-r,p,a) /;
ana-fin(l,l-r,l-p,regle(l-p,p),a) ->
     ana-concl(l,l-r,p,a) /;
ana-fin(l,l,p,a) -> ligne
     exm("CONCLUSION INCORRECTE") ligne / echec;
```

```
ana-liste-conds(l,l-r,c.l-c,a) -> ana-cond(l,l',c,a)
     ana-conds(l',l-r,l-c,a) /;
ana-liste-conds(l,l,l-c,a) -> ligne
     exl("LISTE DE CONDITIONS INCORRECTE") / echec;

ana-conds(mt-et.l,l-r,l-c,a) ->
     ana-liste-conds(l,l-r,l-c,a) /;
ana-conds(l,l,nil,a) -> /;

ana-cond(mt-pas.mt-de.l,l-r,non(p),a) ->
     / ana-conda(l,l-r,p,a);
ana-cond(mt-pas.mt-d."'".l,l-r,non(p),a) ->
     / ana-conda(l,l-r,p,a);
ana-cond(mt-il.mt-n."'".mt-est.mt-pas.mt-vrai.mt-que.l,
     l-r,non(p),a) -> / ana-conda(l,l-r,p,a);
ana-cond(l,l-r,p,a) -> ana-conda(l,l-r,p,a);

ana-conda(mt-le.mt-test.mt-atypique.mt-est.l,l-r,
  est(testat(c.m.nil)),a) ->
  ana-test(l,l-r,c) /
  dansa(<testat,c>,a) dansa(<modat,m>,a);
ana-conda(mt-test.mt-atypique.l,l,
  est(testat(c.m.nil)),a) -> /
  dansa(<testat,c>,a) dansa(<modat,m>,a);
ana-conda(mt-le.mt-test.mt-complementaire.mt-est.l,l-r,
  est(testcomp(c.m.nil)),a) ->
  ana-testc(l,l-r,c') /
  si-sinon(val(c',""),succes,eg(c',c))
  dansa(<testcomp,c>,a) dansa(<modcomp,m>,a);
ana-conda(mt-test.mt-complementaire.l,l,
  est(testcomp(c.m.nil)),a) -> /
  dansa(<testcomp,c>,a) dansa(<modcomp,m>,a);
ana-conda(mt-le.mt-test.mt-est.l,l-r,
  est(carac(c.m.nil)),a) ->
  ana-test(l,l-r,c) /
  dansa(<modt,m>,a) dansa(<testkit,c>,a);
ana-conda(mt-la.mt-modalite.mt-atypique.mt-est.l,l-r,
  est(modat(c.m.nil)),a) ->
  ana-mod(l,l-r,m) /
  dansa(<modat,m>,a) dansa(<testat,c>,a);
ana-conda(mt-la.mt-modalite.mt-complementaire.mt-est.l,
  l-r,est(modcomp(c.m.nil)),a) ->
  ana-mod(l,l-r,m)
  / dansa(<modcomp,m>,a) dansa(<testcomp,c>,a);
ana-conda(mt-la.mt-modalite.mt-est.l,l-r,
  est(modkit(c.m.nil)),a) ->
  ana-mod(l,l-r,m) /
  dansa(<modt,m>,a) dansa(<testkit,c>,a);
ana-conda(mt-le.mt-genre.mt-est.l,l-r,
  est(genr(g.nil)),a) -> ana-genre(l,l-r,g) /
  dansa(<genr,g>,a);
ana-conda(mt-l."'".mt-espece.mt-identifiee.mt-est.l,l-r,
```

```
  est(espid(e.p.nil)),a) ->
  ana-esp(1,1-r,e) /
  dansa(<espid,e>,a) dansa(<probm,p>,a);
ana-conda(mt-1."'".mt-espece.mt-identifiable.mt-est.1,
  1-r,est(esp(e.p.nil)),a) ->
  ana-esp(1,1-r,e) / dansa(<esp,e>,a) dansa(<prob,p>,a);
ana-conda(mt-1."'".mt-espece.mt-nouvelle.mt-est.1,1-r,
  est(espn(e.k.nil)),a) ->
  ana-nom(1,1-r,e) /
  dansa(<espn,e>,a);
ana-conda(mt-espece.mt-nouvelle.1,1,
  est(espn(e.k.nil)),a) -> /
  dansa(<espn,e>,a);
ana-conda(mt-identique.mt-a.mt-une.mt-souche.mt-de.
  mt-collection.mt-de.1,1-r,
  est(espn(e.1.nil)),a) ->
  ana-nom(1,1-r,e) /
  dansa(<espnl,e>,a);
ana-conda(mt-identique.mt-a.mt-une.mt-souche.
  mt-de.mt-collection.1,1,est(espn(e.1.nil)),a) -> /
  dansa(<espnl,e>,a);
ana-conda(mt-appartenance.mt-possible.mt-a.mt-une.
  mt-espece.mt-nouvelle.1,1,
  est(espn(e.2.nil)),a) -> /
  dansa(<espn2,e>,a);
ana-conda(mt-appartenance.mt-possible.mt-a.1,1-r,
  est(espn(e.2.nil)),a) ->
  ana-nom(1,1-r,e) /
  dansa(<espn2,e>,a);
ana-conda(mt-rapprochement.mt-possible.mt-d."'".mt-une.
  mt-espece.mt-nouvelle.1,1,
  est(espn(e.3.nil)),a) -> /
  dansa(<espn3,e>,a);
ana-conda(mt-rapprochement.mt-possible.mt-de.1,1-r,
  est(espn(e.3.nil)),a) ->
  ana-nom(1,1-r,e) /
  dansa(<espn3,e>,a);
ana-conda(mt-la.mt-probabilite.mt-du.mt-genre.mt-est.1,
  1-r,est(probg(g.o.p.v.nil)),a) ->
  ana-oper(1,1',o) ana-entier(1',1-r,v) /
  dansa(<probg,p>,a) dansa(<genr,g>,a);
ana-conda(mt-la.mt-probabilite.mt-maximum.mt-est.1,1-r,
  est(probm(e.o.p.v.nil)),a) ->
  ana-oper(1,1',o) ana-entier(1',1-r,v) /
  dansa(<probm,p>,a) dansa(<espid,e>,a);
ana-conda(mt-la.mt-probabilite.mt-est.1,1-r,
  est(prob(e.o.p.v.nil)),a) ->
  ana-oper(1,1',o) ana-entier(1',1-r,v)
  / dansa(<esp,e>,a) dansa(<prob,p>,a);
ana-conda(mt-le.mt-profil.mt-est.1,1-r,
  est(prof(p.1-p.nil)),a) ->
  ana-prof(1,1-r,1-p) / dansa(<prof,p>,a);
ana-conda(mt-quelque.mt-soit.mt-le.mt-profil.1,1,
```

```
  est(prof(p'.p.nil)),a) ->
  / dansa(<prof,p'>,a);
ana-conda(l,l-r,p,a) -> ana-pred(l,l-r,p,a) /;
ana-conda(l,l-r,p,a) ->
  ligne exm("CONDITION INCONNUE OU MAL ECRITE") ligne
  ex(l) ligne echec;

ana-test(l,l-r,c) -> ana-nom(l,l-r,cl) test(i,j,c)
  strcmp(c,cl,2) /;
ana-test(l,l,c) -> ligne exl("TEST INCONNU") ex(l)
  ligne echec;

ana-testc(l,l-r,c) -> ana-nom(l,l-r,cl) test(i,0,c)
  strcmp(c,cl,2) /;
ana-testc(l,l,c) ->
  ligne exl("TEST COMPLEMENTAIRE INCONNU") ex(l)
  ligne echec;

ana-esp(l,l-r,c) -> ana-nom(l,l-r,cl) espece(i,c)
  strcmp(c,cl,2) /;
ana-esp(l,l,c) -> ligne exl("ESPECE INCONNUE") ex(l)
  ligne echec;

ana-genre(l,l-r,c) -> ana-nom(l,l-r,cl) genre(c)
  strcmp(c,cl,2) /;
ana-genre(l,l,c) -> ligne exl("GENRE INCONNU") ex(l)
  ligne echec;

ana-prof(l,l-r,p.l-p) ->
  ana-comp(l,l',p) ana-prof(l',l-r,l-p) /;
ana-prof(l,l-r,p.nil) -> ana-comp(l,l-r,p) /;

ana-comp("(".l,l-r,l-p) -> / ana-liste(l,")".l-r,l-p);
ana-comp(e.l,l,e) -> entier(e) /;
ana-comp(l,l-r,v) -> ana-var(l,l-r);

ana-liste(e.l,l-r,e.l-p) ->
  entier(e) ana-liste(l,l-r,l-p) /;
ana-liste(e.l,l,e.nil) -> entier(e);

ana-var(mt-p.c.l,l) -> entier(c) /;
ana-var(mt-p.l,l) ->;

ana-concl(l,l-r,<m,l-v>,a) -> ana-liste-var(l,l',l-vl,a)
  ana-ident(l',l'',m')
  boum(m,m')
  ana-liste-var(l'',l-r,l-v2,a) / conc(l-vl,l-v2,l-v)
  ssi(non(expl(<m,l-v>,m-exp)),
    ajout-expl(m,m',l-vl,l-v2));
ana-concl(l,l-r,m,a) -> ligne exm("ERREUR CONCLUSION")
  ligne ex(l) ligne echec;

ajout-expl(m,m',l-vl,l-v2) ->
```

```
  liste-libre(l-v1,l-lv1)
  liste-libre(l-v2,l-lv2)
  conc(l-lv1,m'.l-lv2,m-exp)
  conc(l-lv1,l-lv2,l-lv)
  ajout(<expl(<m,l-lv>,m-exp),nil>);

liste-libre(nil,nil) -> /;
liste-libre(x.l,x'.l') -> liste-libre(l,l');

ana-pred(l,l-r,<m,l-v>,a) -> ana-liste-var(l,l',l-v1,a)
  ana-ident(l',l'',m')
  boum(m,m')
  ana-liste-var(l'',l-r,l-v2,a) / conc(l-v1,l-v2,l-v)
  expl(<m,l-v>,m-exp);
ana-pred(l,l-r,m,a) ->
  ligne exm("ERREUR PROPRIETE NON DEFINIE")
  ligne ex(l) ligne echec;

ana-liste-var(l,l-r,v.l-v,a) -> variable(l,l',v,a) /
  ana-liste-var(l',l-r,l-v,a);
ana-liste-var(l,l,nil,a) ->;

ana-conclu(l,l-r,v.l-p,a) ->
  ana-mod(l,l',v) / ana-conclu(l',l-r,l-p,a);
ana-conclu(l,l-r,v.l-p,a) ->
  ana-ponct(l,l',v) / ana-conclu(l',l-r,l-p,a);
ana-conclu(l,l-r,v.l-p,a) ->
  ana-entier(l,l',v) / ana-conclu(l',l-r,l-p,a);
ana-conclu(l,l-r,v.l-p,a) ->
  ana-nom(l,l',v) / ana-conclu(l',l-r,l-p,a);
ana-conclu(l,l-r,v.l-p,a) -> variable(l,l',v,a) /
  ana-conclu(l',l-r,l-p,a);
ana-conclu(e.l,l-r,e'.l-p,a) ->
  ana-mot(e,e') / ana-conclu(l,l-r,l-p,a);
ana-conclu(l,l,nil,a) ->;

variable(mt-test.mt-atypique.l,l,v,a) ->
  / dansa(<testat,v>,a);
variable(mt-modalite.mt-atypique.l,l,v,a) ->
  / dansa(<modat,v>,a);
variable(mt-test.mt-complementaire.l,l,v,a) -> /
  dansa(<testcomp,v>,a);
variable(mt-modalite.mt-complementaire.l,l,v,
  "modalite complementaire",a) -> /
  dansa(<modcomp,v>,a);
variable(mt-test.l,l,v,a) ->
  / dansa(<testkit,v>,a);
variable(mt-modalite.l,l,v,a) ->
  / dansa(<modt,v>,a);
variable(mt-espece.mt-identifiee.l,l,v,a) -> /
  dansa(<espid,v>,a);
variable(mt-espece.mt-identifiable.l,l,v,a) -> /
  dansa(<esp,v>,a);
```

```
variable(mt-espece.mt-nouvelle.1,1,v,a) -> /
  dansa(<espn,v>,a);
variable(mt-espece.mt-nouvelle.1.1,1,v,a) -> /
  dansa(<espn1,v>,a);
variable(mt-espece.mt-nouvelle.2.1,1,v,a) -> /
  dansa(<espn2,v>,a);
variable(mt-espece.mt-nouvelle.3.1,1,v,a) -> /
  dansa(<espn3,v>,a);
variable(mt-genre.1,1,v,a) -> / dansa(<genr,v>,a);
variable(mt-probabilite.mt-maximum.1,1,v,a) ->
  / dansa(<probm,v>,a);
variable(mt-probabilite.mt-genre.1,1,v,a) ->
  / dansa(<probg,v>,a);
variable(mt-probabilite.1,1,v,a) ->
   / dansa(<prob,v>,a);
variable(mt-profil.1,1,v,a) ->
  / dansa(<prof,v>,a);

ana-ident(l,l-r,m) -> ana-mote(l,"-".l',m') /
     ana-ident(l',l-r,m'')
     strcat(m',"-",ml) strcat(ml,m'',m);
ana-ident(l,l-r,m) -> ana-mote(l,l-r,m);

ana-mote(e.l,l-r,m) -> ana-mot(e,e')
     ana-entier(l,l-r,v) /
     decstr(v,v') strcat(e',v',m);
ana-mote(e.l,l,e') -> ana-mot(e,e');

ana-nom(l,l-r,m) -> ana-motmaj(l,l',ml)
     ana-noms(l',l-r,m2) / strcat(ml,m2,m);

ana-noms("-".l,l-r,m) -> ana-nomi(l,l',ml) /
     ana-noms(l',l-r,m2)
     strcat(ml,m2,m3) strcat("-",m3,m);
ana-noms("_".l,l-r,m) -> ana-nomi(l,l',ml) /
     ana-noms(l',l-r,m2)
     strcat(ml,m2,m3) strcat(".",m3,m);
ana-noms(l,l,"") ->;

ana-nomi(l,l-r,m) -> ana-entier(l,l-r,e) / decstr(e,m);
ana-nomi(l,l-r,m) -> ana-motmaj(l,l-r,m);

ana-motmaj(e.l,l-r,m) -> maj(e,el)
     ana-motmaj(l,l-r,ml) /
     strcat(el,ml,m);
ana-motmaj(e.l,l,m) -> maj(e,m);

ana-oper("<"."=".l,l,infeg) -> /;
ana-oper(">"."=".l,l,supeg) -> /;
ana-oper("=".l,l,val) -> /;
ana-oper("<".l,l,infer) -> /;
ana-oper(">".l,l,super) -> /;
```

```
ana-mot(m,m') -> boum(m,m'') strmid(m'',1,3,"mt-") /
     strlen(m'',l) strmid(m'',4,l,m');

ana-mod("+".l,l,"+") ->;
ana-mod("-".l,l,"-") ->;

ana-ponct("'".l,l,"'") -> /;
ana-ponct(",".l,l,",") -> /;
ana-ponct(";".l,l,";") -> /;
ana-ponct(":".l,l,":") -> /;
ana-ponct("(".l,l,"(") -> /;
ana-ponct(")".l,l,")") -> /;
ana-ponct("%".l,l,"%") -> /;

ana-entier(e.l,l,e) -> entier(e);
```

On trouve ci après la définition des predicats "EST" qui peuvent apparaitre dans les règles compilées du système expert. Ils utilisent les prédicats de la base de faits et d'autres prédicats comme "EXPANSION" ou "CORRESPOND" qui sont définis en Prolog.

```
est(carac(c.m.nil)) ->
  caractere(c,m) /;
est(testat(c.m.nil)) ->
  car-atypique(c,m) /;
est(testcomp(c.m.nil)) ->
  car-complementaire(c,m) /;
est(modat(c.m.nil)) ->
  car-atypique(c,m) /;
est(modcomp(c.m.nil)) ->
  car-complementaire(c,m) /;
est(modkit(c.m.nil)) ->
  caractere(c,m) /;
est(genr(g.nil)) ->
  genre(g) /;
est(espid(e.p.nil)) ->
  probmax(p) espece-identifiee(e,p) /;
est(esp(e.p.nil)) ->
  espece-identifiee(e',p) /;
est(espn(e.k.nil)) ->
  espece-nouvelle(e',k) compare(e',e) /;
est(probg(g.o.p.v.nil)) ->
  genre(g) proba-genre(g,p) <o,p,v> /;
est(probm(e.o.p.v.nil)) ->
  probmax(p) <o,p,v> espece-identifiee(e,p) /;
est(prob(e.o.p.v.nil)) ->
   espece-identifiee(e,p) <o,p,v> /;
est(prof(p.l-p.nil)) ->
  profil(p,x-ext) expansion(p,l-p')
  correspond(l-p',l-p) /;
```

L'INTERFACE PROLOG_INFORMIX DE NIVEAU 3

Ce niveau d'interface complète les précédents. On utilise ici les possibilités de PROLOG pour intégrer les connaissances qu'un utilisateur averti peut avoir de l'organisation et de l'utilisation d'une base de données particulière. Ces connaissances permettent la génération à partir d'une formulation aussi simple que possible d'une série de demandes à destination de l'interface de niveau 2.

INITIALISATION est le prédicat d'entrée dans l'interface de niveau 3. On descend dans un espace de travail (monde en Prolog II) spécifique, on ouvre la base de nom n-b. On vérifie que l'espace de travail contient la base de faits nécéssaire à l'utilisation de l'interface et dans le cas contraire on la génère à l'aide de INITIALISATION''.

```
initialisation(n-b) ->
   descendre(n-b)
   lg-ligne(130)
   ouvrir-bd(n-b)
   initialisation'(n-b);

initialisation'(n-b) -> nom-base(n-b) /;
initialisation'(n-b) ->
   / ajout(<nom-base(n-b),nil>)
   initialisation''(n-b);
```

INITIALISATION'' utilise des prédicats de l'interface de niveau 1 pour compléter une base de faits relative à la B.D utilisée. Cette base de faits se compose de prédicats qui fournissent certains renseignements sur l'implantation d'un système d'information particulier et est utile à la réalisation des traitements ultérieurs. Certains de ces faits sont générés automatiquement lors de cette phase d'initialisation. D'autres doivent être introduits par le concepteur de la base. Dans la première catégorie on note les prédicats suivants :

NOM-BASE(n-b) : nom de la base concernée,

CHAMP-RELAT(n-c,n-r) : nom (n-r) de la relation à laquelle appartient le constituant (n-c),

CHAMP-COMPO(n-c,c-c,i-c,n-c') : nom du constituant complexe (n-c) auquel peut appartenir un composant élémentaire (n-c'), (c-c) est le nombre de composants et i-c le numéro d'ordre du composant,

TYPE-INDEX(n-r,n-c,t-i) : type de l'index (t-i) associé à un constituant (n-c) d'une relation (n-r) (index unique, duplicable, constituant non indéxé).

Dans la deuxième on trouve les prédicats :

NOM-RELATION(n-r-e,n-r) : donne le nom externe (n-r-e) associé à la relation (n-r),
CHAMP-LIE(n-ent,n-c) : nom de l'entité (n-ent) associée au constituant n-c,
LIAISON(n-r,n-r',l-l) : l-l est une liste qui detaille le lien existant entre deux relations (n-r) et (n-r'),
CLASSEMENT(i,n-r,n-c,o,v-c) : ordonne (i) les différentes requètes en fonction de leurs caractéristiques.

```
initialisation''(n-b) ->
   obtient-num-relat(n-r,i)
   ouvrir-fich(n-r)
   gener-champ-relat(n-r) fermer-fich(n-r) echec;
initialisation''(n-b) ->;

gener-champ-relat(n-r) ->
   info-fich(n-r,c-c,l-e)
   entier-infeg(j-c,c-c)
   info-numchamp(n-r,j-c,n-c,t-c,l-c)
   gener-champ-compo(n-r,n-c,t-c,l-c)
   info-index(n-r,n-c,t-i)
   ajout(<type-index(n-r,n-c,t-i),nil>)
   ajout(<champ-relat(n-c,n-r),nil>)
   echec;
gener-champ-relat(n-r) ->;


gener-champ-compo(n-r,n-c,comptype,c-c) ->
   entier-infeg(j-c,c-c)
   info-composite(n-r,n-c,j-c,n-c',x,y)
   val(add(j-c,1),j-c')
   ajout(<champ-compo(n-c,c-c,j-c',n-c'),nil>) echec;
gener-champ-compo(n-r,n-c,t-c,c-c) ->;
```

Dans l'interface de niveau 3 une demande se formule à l'aide du pédicat SELECTION. Ce prédicat se présente sous deux formes . La première ne comprend pas de paramètres et se contente d'afficher à l'écran l'ensemble des réponses obtenues. La seconde ajoute en outre les prédicats (n-r) qui contiendront les n-uplets de la réponse. Les deux formes appellent le prédicat SELECTION à 2 arguments n-r et l (n-r reste libre dans le cas de la première forme). l contient une liste acquise par le prédicat ENTRER-SELECTION qui contient les éléments de la requète .Aprés compilation cette liste est transmise à l'interface de niveau 2 par l'intermédiaire de SELECTION'.

```
selection ->
   fermer-tout
   entrer-selection(l)
   selection(n-r,l);

selection(n-r) ->
   fermer-tout
   garder-solution(n-r)
   entrer-selection(l)
   selection(n-r,l);

garder-solution(n-r) -> ssi(tete(n-r),
   garder-solution'(n-r));

garder-solution'(n-r) -> ligne
   exl("Cette relation existe déjà , faut-il :")
   exl("- 1 - supprimer.")
   exl("- 2 - ajouter.")
   exl("- 3 - donner un autre nom.")
   in-entier(i)
   garder-solution'(i,n-r);

garder-solution'(1,n-r) -> / ote-regle(n-r);
garder-solution'(2,n-r) -> /;

selection(n-r,l) ->
   configurer(l,ll)
   traiter-selection(l,ll,l')
   commencer-relation(l')
   ligne
   selection'(n-r,l,l');
```

ENTRER-SELECTION saisit une liste de n-uplets terminée par le caractére ".". Chaque n-uplet concerne un élément de la requète et correspond à une sélection dans la base. Cette sélection qui porte sur une entité (n-c) est définie par un opérateur (o) et une valeur de comparaison (v-c). Le quatrième élément du n-uplet est une variable (v-t) qui sera ultérieurement unifiée à la valeur trouvée dans la base. IN-CHAMP saisit le nom de l'entité, IN-OPER l'opérateur, IN-VAL la valeur de comparaison.

```
entrer-selection(<n-c,o,v-c,v-t>.l) ->
   in-champ(n-c)
   in-oper(o)
   in-val(v-c)
   / entrer-selection(l);
entrer-selection(nil) ->;
```

IN-CHAMP permet l'entrée d'un nom d'entité. Il en vérifie l'existence à l'aide de champ-relat et arrète la

saisie à la rencontre du caractère point par mise en échec.

```
in-champ(n-c) -> car-apres'(".") / vider-ligne echec;
in-champ(n-c) ->
   in-chaine(n-c)
   champ-relat(n-c,x) /;
in-champ(n-c) -> ligne
   exm("Ce champ n'existe pas dans la base.") /
   vider-ligne in-champ(n-c);
```

Les opérateurs admis par IN-OPER sont "<", ">", "<=", ">=", "=". Chacun de ces opérateurs est transmis par le paramètre (o) sous la forme d'un des identificateurs petit, grand, pteq, gdeq, egal . Le caractére "?" peut être utilisé dans le cas où l'on ne souhaite pas fournir un opérateur. Ce cas correspond à une absence de sélection pour l'entité concerné (toute les valeurs trouvées seront retenues), le paramétre (o) reste libre.

```
in-oper(o) -> in-oper'(o) /;
in-oper(o) -> ligne exm("Operateur inconnu.")
   vider-ligne / in-oper(o);

in-oper'(o) -> car-apres'("<") /
   in-car'("<") inf-apres(o);
in-oper'(o) -> car-apres'(">") /
   in-car'(">") sup-apres(o);
in-oper'(egal) -> car-apres'("=") / in-car'("=");
in-oper'(o) -> in-car'("?") /;

sup-apres(gdeq) -> car-apres("=") / in-car("=");
sup-apres(grand) -> /;

inf-apres(pteq) -> car-apres("=") / in-car("=");
inf-apres(petit) -> /;
```

Les valeurs admises par IN-VAL sont : une chaine, un entier ou le caractère "?" si l'on ne souhaite pas transmettre de valeurs de comparaison (l'argument v du prédicat reste libre dans ce cas).

```
in-val(v) -> in-chaine(v) /;
in-val(v) -> in-entier(v) /;
in-val(v) -> in-car'("?") /;
in-val(v) -> ligne exm("Valeur incorrecte.") /
   vider-ligne in-val(v);
```

CONFIGURER transforme une liste des requètes relatives à des entités en ajoutant des marques (prs ou lbr) aux éléments opérateur (o) et valeur de comparaison (v-c). La marque prs est utilisée si l'élément correspondant est défini sinon on appose la marque lbr. Le choix de la marque appropiée est effectué par le prédicat MARQUE-ELEM. Le marquage joue un

rôle lors de la définition de la sélection type et lors de la réalisation de la demande.

```
configurer(<n-c,o,v-c,v-t>.l,
   <n-c,<o,m>,<v-c,m'>,v-t>.l') ->
   pris(n-c)
   marque-elem(o,m) marque-elem(v-c,m')
   / configurer(l,l');
configurer(nil,nil) ->;

marque-elem(x,prs) -> prisl(x) /;
marque-elem(x,lbr) ->;
```

TRAITER-SELECTION examine la liste (l) des sélections et la traduit en la complétant en une liste (l') qui peut être comprise par l'interface de niveau 2. La liste marquée (ll) est utilisée pour rechercher dans la base de faits si une compilation comparable n'a pas déjà été effectuée. SELECTION-TYPE produit alors immédiatement le résultat.

```
traiter-selection(l,ll,l') ->
   selection-type(ll,l') /;
traiter-selection(l,ll,l') ->
   considerer-selection(l,l2,a)
   mettre-fin-liste(l2,l2)
   terminer-selection(l2,l')
   ligne
   exm("Faut-il définir un prototype")
   exm(" pour cette selection ? (o/n) : ")
   ssi(in-car'("o"),ajout-selection-type(ll,l',a'));
```

Chaque élément de la liste des demandes doit être mis en rapport avec les éléments précédents. Des liens doivent être constatés et établis entre une nouvelle requète et les requètes antérieures. Ce travail est réalisé par le prédicat CONSIDERER-SELECTION. A partir de la liste initiale (l) ce prédicat construit la liste compilée (l') en utilisant un arbre de constituants (a). La liste l est une liste de demandes qui ne mentionne que le nom d'une entité, un opérateur et une valeur de comparaison. A chaque entité peuvent naturellement correspondre plusieurs constituants qui appartiennent à des relations différentes. Il faut déterminer quelle est la relation et le constituant concerné (DEFINIR-RELATION) et réajuster la liste résultat (l') (METTRE-SELECTION ou ETABLIR-SELECTION).

```
considerer-selection(s.l,l',a) ->
   trace-selection(s)
   definir-relation(s,l',s',l-s)
   editer-relation(s,s')
   si-sinon(pris(l-s),
     mettre-selection(s',l-s,a),
     etablir-selection(s',l',a))
```

```
      / considerer-selection(l,l',a);
   considerer-selection(nil,l',a) -> /;
```

TRACE-SELECTION génère un texte rappelant le contenu de la sélection examinée.

```
   trace-selection(<n-c,o,v-c,v-t>) ->
      ligne ligne ligne
      exm(
         "On examine la selection qui porte sur le champ ")
      exm(n-c)
      trace-selection'(o,v-c);

   trace-selection'(o,v-c) -> ou(libre(v-c),libre(o)) /;
   trace-selection'(o,v-c) -> ligne
      exm("Les valeurs selectionnées doivent être ")
      edit-oper(o)
      edit-val(v-c);

   edit-oper(petit) -> exm("plus petites que ") /;
   edit-oper(pteq) -> exm("égales ou plus petites que ") /;
   edit-oper(egal) -> exm("égales à ") /;
   edit-oper(gdeq) -> exm("égales ou plus grandes que ") /;
   edit-oper(grand) -> exm("plus grandes que ") /;

   edit-val(v-c) -> chaine(v-c) / exm(v-c);
   edit-val(v-c) -> ex(v-c);
```

A un nom d'entité (n-c-r) on fait correspondre un nom de relation (n-r) et un nom de constituant (n-c). On propose dans ce but une série de solutions :

- Une relation où l'on trouve un constituant lié à l'entité qui a déja été choisie lors de l'examen des requètes précédentes. Ce choix pour être entériné doit être validé par BON-CHOIX-RELAT.

- Une nouvelle relation et un nouveau constituant déterminé par le prédicat CHOIX-RELATION.

DEFINIR-RELATION donne une liste l-s relative aux requètes déja compilées pour la relation choisie. Dans le cas d'une nouvelle relation cette liste est vide.

```
   definir-relation(<n-c-r,o,v-c,v-t>,l,
      <n-r,<n-c,o,v-c,v-t>>,l-s) ->
      libre(l) / choix-relation(n-c-r,o,v-c,n-r,n-c);
   definir-relation(<n-c-r,o,v-c,v-t>,<n-r,l-s>.l,
      <n-r,<n-c,o,v-c,v-t>>,l-s) ->
      lien-reference(n-c-r,n-r,n-c)
      bon-choix-relat(n-c-r,n-r,n-c) /;
   definir-relation(s,e.l,s',l-s) -> /
      definir-relation(s,l,s',l-s);
```

LIEN-REFERENCE établit l'existence d'une association entre une entité et un couple relation-constituant.

```
lien-reference(n-c-r,n-r,n-c) -> champ-lie(n-c-r,n-c)
   champ-relat(n-c,n-r);

bon-choix-relat(n-c-r,n-r,n-c) ->
   relation-unique(n-c-r,n-r,n-c) ligne/;
bon-choix-relat(n-c-r,n-r,n-c) ->
   ligne
   exm("La relation choisie est la relation des ")
   exr(n-r) ligne
   exm("qui contient un champ du même type.") ligne
   exm("Ce choix doit-il être retenu ? (o/n) : ")
   in-car'("o");

relation-unique(n-c-r,n-r,n-c) ->
   lien-reference(n-c-r,n-r,n-c)
   lien-reference(n-c-r,n-r',n-c')
   dif(n-r',n-r) / echec;
relation-unique(n-c-r,n-r,n-c) ->
   lien-reference(n-c-r,n-r,n-c);
```

Le choix d'une relation à associer à une entité se pose s'il existe dans la base plusieurs constituants de relations différentes qui correspondent à l'entité. Si RELATION-UNIQUE est mis en échec CHOIX-RELATION propose d'une part :

- Une liste de relations possibles fournie par LIEN-REFERENCE. Par le prédicat ENTREE-CHOIX-RELATION l'utilisateur donne sa réponse.

- La détermination d'une relation optimale effectuée par le prédicat CHOIX-RELATION' à l'aide du prédicat CLASSEMENT. Les critères utilisés pour obtenir cet optimum peuvent être des critères généraux ou dépendre de la structure d'une base particulière. Ils doivent être établis par l'utilisateur qui généralement indique par ce moyen ses préférences pour des relations où les constituants associés à l'entité sont dotés de propriétés (indexation) qui favorisent les recherches ultérieures.

```
choix-relation(n-c-r,o,v-c,n-r,n-c) ->
   relation-unique(n-c-r,n-r,n-c) ligne /;
choix-relation(n-c-r,o,v-c,n-r,n-c) ->
   ligne
   exm("Il faut déterminer quelle est")
   exm(" la relation associée au champ ")
   exl(n-c-r)
   exm("Ce choix peut-il être laissé")
   exm(" à l'initiative du système ? (o/n) : ")
   in-car'("n")
```

```
      exl("Dans la liste suivante")
      exm(" il faut sélectionner une relation")
      entree-choix-relat(n-c-r,n-r,n-c) /;
   choix-relation(n-c-r,o,v-c,n-r,n-c) ->
      ligne
      exl("La détermination va être réalisée")
      exm(" en prenant la relation optimale :")
      exl("celle qui permet la sélection la plus précise.")
      affecter(optirelat,999)
      choix-relation'(n-c-r,o,v-c)
      val(numrelat,j-r)
      info-numfich(j-r,n-r,x,y)
      lien-reference(n-c-r,n-r,n-c)
      exr(n-r) exm(" est la relation choisie ,")
      exm(" êtes vous d'accord (o/n) : ")
      in-car'("o") /;
   choix-relation(n-c-r,o,v-c,n-r,n-c) ->
      / choix-relation(n-c-r,o,v-c,n-r,n-c);

   entree-choix-relat(n-c-r,n-r,n-c) ->
      lien-reference(n-c-r,n-r,n-c)
      ligne
      exm("Relation des ") exr(n-r)
      echec;
   entree-choix-relat(n-c-r,n-r,n-c) ->
      ligne
      exm("Quel est le nom de la relation choisie : ")
      inr(n-r)
      lien-reference(n-c-r,n-r,n-c) /;
   entree-choix-relat(n-c-r,n-r,n-c) ->
      ligne exm("Cette relation n'existe pas dans la base")
      vider-ligne /
      entree-choix-relat(n-c-r,n-r',n-c');

   choix-relation'(n-c-r,o,v-c) ->
      lien-reference(n-c-r,n-r,n-c)
      classement(c,n-r,n-c,o,v-c)
      val(inf(c,optirelat),1) affecter(optirelat,c)
      obtient-num-relat(n-r,j-r)
      affecter(numrelat,j-r) echec;
   choix-relation'(n-c-r,o,v-c) ->;

   editer-relation(<n-c-r,o,v-c,v-t>,<n-r,s>) -> ligne
      exm("Le champ ") exm(n-c-r)
      ligne exm("est un champ de la relation des ")
      exr(n-r);
```

L-S est une liste de sélections relative à la relation n-r, s est une nouvelle sélection à ajouter à cette liste. Il s'agit d'un 4-uplet <n-c,o,v-c,v-t> : constituant, opérateur, valeur de comparaison, valeur à trouver. Le prédicat VOIR-LIAISON a pour objet d'unifier entre elles toute les occurences des variables et des valeurs figurant dans les

4-uplets relatifs à une même entité. VOIR-LIAISON utilise pour réaliser ce travail un arbre ordonné par noms d'entités (a) dont les feuilles sont des triplets <n-c-r,v-c,v-t> : nom d'entité, valeur de comparaison, valeur à trouver.

```
mettre-selection(<n-r,s>,l-s,a) ->
   voir-liaison(s,a)
   dans(s,l-s);
```

Si la sélection concerne une nouvelle relation il faut compléter et réactualiser la liste des sélections déja compilées (l'). Entre cette nouvelle relation et les précédentes peuvent en effet exister des rapports dont il faut tenir compte. Ces liens se traduisent par l'existence , dans des relations différentes, de constituants liés à une même entité. Cela peut amener à introduire des 4-uplets sélection supplémentaires : ceux qui sont induits par la présence de deux relations liées par une chaine d'entités. ETABLIR-LIEN complète ainsi la liste l' en utilisant la nouvelle sélection s', la liste initiale l'' et l'arbre des entités a.

Le prédicat LIAISON donne dans la liste l-l les éléments qui lient entre elles les relations n-r' et n-r. Ce prédicat appartient à la base de faits et doit être préalablement défini par l'admnistrateur de la base. Les listes sont constituées de couples <nom de relation, constituant>. Les constituants qui figurent dans deux couples succéssifs de la liste sont rattachés à une même entité et font correspondre deux realtions différentes où figurent dans une même relation.

COMPLETER-LIAISON utilise les éléments de la liste liaison l-l pour mettre à jour la liste l' des sélections.

```
etablir-selection(s',l',a) ->
   mettre-fin-liste(l',l'')
   etablir-lien(s',l'',l',a);

etablir-lien(<n-r,<n-c,o,v-c,v-t>>,<n-r',l-s>.l,l',a) ->
   liaison(n-r',n-r,l-l)
   completer-liaison(l-l,s',l',l-s,a)
   / etablir-lien(<n-r,<n-c,o,v-c,v-t>>,l,l',a);
etablir-lien(s,e.l,l',a) -> / etablir-lien(s,l,l',a);
etablir-lien(<n-r,s>,nil,l,a) ->
   dans(<n-r,l-s>,l) completer-relation(s,l-s,a);
```

Le premier argument de COMPLETER-LIAISON : <n-r,n-c'> est un couple de la liste liaison entre deux relations. Ce couple permet la définition d'une sélection à ajouter a celles de la relation n-r.

Le deuxième argument du prédicat COMPLETER-LIAISON correspond à l'élément liaison précédent dans la liste l-l. Cet élément a été complété par un opérateur, une valeur de comparaison et une valeur à trouver de manière à définir une

nouvelle sélection. Cette nouvelle sélection est ajoutée à la liste l-s des sélections pour la relation n-r par le prédicat COMPLETER-RELATION.

L'examen de la liste liaison l-l peut conduire à changer de relation. Le prédicat DANS est alors utilisé pour aller rechercher dans la liste l des sélections, la sous liste l-s' qui correspond à la nouvelle relation.

Lorsque l'on change de relation la nouvelle sélection est définie à l'aide des éléments de la sélection précédente puisque par hypothèse les constituants correspondants dépendent de la même entité.

Lorsque l'on reste dans la relation on complète par l'opérateur egal et une valeur de comparaison définie sous la forme d'une variable libre v-c'. Lors de la recherche de la réponse cette variable v-c' peut être unifiée à une valeur trouvée dans la base.

```
completer-liaison(<n-r,n-c'>.l-l,<n-r,<n-c,o,v-c,v-t>>,
   l,l-s,a) ->
   completer-relation(<n-c,o,v-c,v-t>,l-s,a)
   / completer-liaison(l-l,<n-r,<n-c',egal,v-c',v-t'>>,
       l,l-s,a);
completer-liaison(<n-r',n-c'>.l-l,<n-r,<n-c,o,v-c,v-t>>,
   l,l-s,a) ->
   completer-relation(<n-c,o,v-c,v-t>,l-s,a)
   dans(<n-r',l-s'>,l)
   / completer-liaison(l-l,<n-r',<n-c',o,v-c,v-t>>,l,
       l-s',a);
completer-liaison(nil,<n-r,s>,l,l-s,a) ->
   completer-relation(s,l-s,a);
```

COMPLETER-RELATION met à jour la liste l-s des requètes pour une relation. On utilise VOIR-LIAISON pour unifier les variables et valeurs qui dépendent d'une même entité.

```
completer-relation(s,s.l-s',a) -> voir-liaison(s,a) /;
completer-relation(s,l-s,a) -> libre(l-s) / echec;
completer-relation(s,s'.l-s,a) ->
   / completer-relation(s,l-s,a);
```

VOIR-LIAISON utilise l'arbre des entités (a) pour unifier les éléments d'une nouvelle sélection par rapport à ceux des sélection précédentes. On tient compte de l'existence de constituants de nature complexe où interviennent plusieurs composants élémentaires.

```
voir-liaison(<n-c,o,v-c,v-t>,a) ->
   champ-compo(n-c,x,y,z) /
   decompose-champ(<n-c,v-c,v-t>,a);
voir-liaison(<n-c,o,v-c,v-t>,a) ->
   champ-lie(n-c-r,n-c)
   deja-vu(<n-c-r,v-c,v-t>,a) /;
voir-liaison(<n-c-r,o,v-c,v-t>,a) ->
```

```
      champ-lie(n-c-r,n-c)
      deja-vu(<n-c-r,v-c,v-t>,a) /;
```

DECOMPOSE-CHAMP extrait les composants élémentaires d'un constituant complexe.

```
   decompose-champ(e,a) ->
      decompose-champ'(1,e,a) /;
   decompose-champ(<n-c,<v-c,m>,v-t>,a) ->
      decompose-champ''(1,<n-c,v-c,v-t>,a);

   decompose-champ'(i,<n-c,v-c.1-c,v-t.1-t>,a) ->
      champ-compo(n-c,c-c,i,n-c')
      champ-lie(n-c-r,n-c')
      deja-vu(<n-c-r,v-c,v-t>,a)
      val(add(i,1),il) /
      decompose-champ'(il,<n-c,1-c,1-t>,a);
   decompose-champ'(i,<n-c,nil,nil>,a) ->;

   decompose-champ''(i,<n-c,v-c.1-c,v-t.1-t>,a) ->
      champ-compo(n-c,c-c,i,n-c')
      champ-lie(n-c-r,n-c')
      deja-vu(<n-c-r,<v-c,m>,v-t>,a)
      val(add(i,1),il) /
      decompose-champ''(il,<n-c,1-c,1-t>,a);
   decompose-champ''(i,<n-c,nil,nil>,a) ->;
```

DEJA-VU gère l'arbre ordonné des entités. Il recherche ou ajoute l'élément relatif à l'entité correspondant au constituant examiné.

```
   deja-vu(e,<e,g,d>) -> /;
   deja-vu(e,<e',g,d>) -> aller-a-gauche(e,e') /
      deja-vu(e,g);
   deja-vu(e,<e',g,d>) -> deja-vu(e,d);

   aller-a-gauche(<n-c,v-c,v-t>,<n-c',v-c',v-t'>) ->
      val(inf(n-c,n-c'),1);
```

TERMINER-SELECTION achève la consitution de la liste compilée des requètes en ajoutant aux opérateurs et aux valeurs de comparaison les marques lbr si l'élément est une variable libre ou prs dans l'autre cas.

```
   terminer-selection(<n-r,1-s>.1,<n-r,1-s'>.1') ->
      configurer(1-s,1-s')
      / terminer-selection(1,1');
   terminer-selection(nil,nil) ->;
```

La constitution de la liste des requètes à fournir à l'interface de niveau 2 est une opération coûteuse qui nécéssite en outre l'intervention ponctuelle de l'utilisateur. Le résultat de ce travail peut être rangé dans

une base de connaissances sous une forme générale où sont mentionnés les noms d'entités, relations et constituants, la valeur des opérateurs et les caractéristiques (prs ou lbr) .Les valeurs de comparaison elles mêmes sont remplacées par des variables libres qui peuvent être unifiées aux valeurs d'une requète comparable . La forme générale est constituée par un couple de listes : forme générale de la requète initiale, forme générale de la requète compilée correspondante.

```
ajout-selection-type(ll,l',a) ->
   ajout-selection-typel(ll,ll',a)
   ajout-selection-typel(l',l'',a)
   ajout(<selection-type(ll',l''),nil>);

ajout-selection-typel(<n-c,<o,m>,<v-c,m'>,v-t>.l,
   <n-c,<o,m>,<v-c',m'>,v-t>.l',a) ->
   voir-liaison(<n-c,<o,m>,<v-c',m'>,v-t>,a)
   / ajout-selection-typel(l,l',a);
ajout-selection-typel(nil,nil,a) ->;

ajout-selection-typel(<n-r,l-s>.l,<n-r,l-s'>.l',a) ->
   ajout-selection-typel(l-s,l-s',a) /
   ajout-selection-typel(l,l',a);
ajout-selection-typel(nil,nil,a) ->;
```

Les prédicats qui suivent permettent la réalisation de la demande : on ouvre les relations et on recherche l'ensemble des solutions à l'aide du prédicat RECHERCHE-MULT de l'interface de niveau 2.

```
commencer-relation(<n-r,l-s>.l) -> ouvrir-fich(n-r)
   / commencer-relation(l);
commencer-relation(nil) ->;

selection'(n-r,l,l') ->
   editer-entete(l) rectifier-selection(l',ll)
   recherche-mult(ll)
   ranger-solution(n-r,l)
   editer-solution(l) echec;
selection'(n-r,l,l') ->
   terminer-relation(l');

rectifier-selection(<n-c,<o,m>,<v-c,prs>,v-t>.l-s,
                    <n-c,<o,m>,<v-c,prs>,v-t>.l-s-r) ->
   / rectifier-selection(l-s,l-s-r);
rectifier-selection(<n-c,<o,m>,<v-c,m-c>,v-t>.l-s,
                   <n-c,<o,m>,<v-c,m-c'>,v-t>.l-s-r) ->
   marque-elem(v-c,m-c')
   / rectifier-selection(l-s,l-s-r);
rectifier-selection(nil,nil) ->;

ranger-solution(n-r,l) -> pris(n-r) /
```

```
      tampon-neuf(ajout-solution(n-r,l));
   ranger-solution(n-r,l) ->;

   ajout-solution(n-r,l) -> bas sortie("tampon")
      ex(n-r) exm("(")
      ajout-solution'(l)
      sortie("console")
      entree("tampon") inserer entree("console");

   ajout-solution'(<n-c,o,v-c,v-t>.nil) ->
      ex(v-t) exm(") ->;;") ligne /;
   ajout-solution'(<n-c,o,v-c,v-t>.l) ->
      ex(v-t) exm(",") /
      ajout-solution'(l);


   editer-entete(l) ->
      ligne editer-entetel(l,l) ligne;

   editer-entetel(p,<n-c,o,v-c,v-t>.l) ->
      position-suivante(p,pl,p',n-c) pos(pl) exm(n-c)
      / editer-entetel(p',l);
   editer-entetel(p,nil) ->;

   editer-solution(l) ->
      ligne editer-solutionl(l,l);

   editer-solutionl(p,<n-c,o,v-c,v-t>.l) ->
      pris(v-t)
      position-suivante(p,pl,p',n-c)
      pos(pl) edit-val(v-t)
      / editer-solutionl(p',l);
   editer-solutionl(p,<n-c,o,v-c,v-t>.l) ->
      position-suivante(p,pl,p',n-c)
      / editer-solutionl(p',l);
   editer-solutionl(p,nil) ->;

   position-suivante(p,p-dep,p',n-c) ->
      champ-lie(n-c,n-c') champ-relat(n-c',n-r)
      info-champ(n-r,n-c',t-c,l-cl)
      strlen(n-c,l-c2) max2(l-cl,l-c2,l-c)
      val(add(p,l-c),pl) val(add(pl,l),p2)
      valid-pos(p2,p,p-dep,p');

   valid-pos(p2,p,p,p2) -> val(inf(p2,130),l) /;
   valid-pos(p2,p,l,p') -> ligne
      val(sub(p2,p),pl) val(add(pl,l),p');
```

OBTIENT-NUM-CHAMP permet d'obtenir le numéro d'un constituant de relation mais aussi d'obtenir l'ensemble des noms de constituants d'une relation si le paramètre correspondant n-c est libre au départ.

```
obtient-num-champ(n-r,n-c,j-c) -> info-fich(n-r,c-c,z)
   entier-infeg(j-c,c-c)
   info-numchamp(n-r,j-c,n-c,x,y);
```

La même remarque s'applique au prédicat suivant relatif cette fois aux relations d'une base.

```
obtient-num-relat(n-r,j-f) -> nbre-fich(c-f)
   entier-infeg(j-f,c-f)
   info-numfich(j-f,n-r,x,y);
```

TERMINER-RELATION ferme l'ensemble des relations utilisées pour une demande.

```
terminer-relation(<n-r,l-s>.l) -> fermer-fich(n-r)
   / terminer-relation(l);
terminer-relation(nil) ->;
```

Les prédicats suivants sont des prédicats "utilitaires". Ils interviennent dans les définitions précédentes et se comprennent facilement.

```
exr(n-r) -> nom-relation(n-r,n) exm(n);

inr(n-r) -> in-chaine(n) nom-relation(n-r,n);

entier-infeg(i,n) -> val(add(n,1),n') entier-inf(i,n');

entier-inf(i,0) -> / echec;
entier-inf(i,n) -> val(sub(n,1),i);
entier-inf(i,n) -> val(sub(n,1),n') entier-inf(i,n');

ou(x,y) -> x;
ou(x,y) -> y;

ssi(c,a) -> c / a;
ssi(c,a) ->;

si-sinon(c,a,b) -> c / a;
si-sinon(c,a,b) -> b;

dans(x,x.l) ->;
dans(x,y.l) -> dans(x,l);

prisl(v) -> libre(v) / echec;
prisl(x.l) -> / pris(x) prisl(l);
prisl(x) ->;

mettre-fin-liste(l,nil) -> libre(l) /;
mettre-fin-liste(nil,nil) -> /;
mettre-fin-liste(x.l,x.l') -> mettre-fin-liste(l,l');

ote-regle(r) -> tete(r)
```

```
   supprimer(l) / ote-regle(r);
ote-regle(r) ->;

max2(x,y,y) -> val(inf(x,y),l) /;
max2(x,y,x) ->;

vider-ligne -> car-apres(c)  no-car(c,10) /;
vider-ligne -> in-car(c) vider-ligne;

fermer-tout -> nom-relation(n-r,x) fermer-fich(n-r)
   echec;
fermer-tout ->
```

L'INTERFACE PROLOG_INFORMIX DE NIVEAU 2

L'interface de niveau 2 utilise les prédicats de l'interface de niveau prédédent pour réaliser, en grande partie, les fonctions habituelles des langages d'interrogation des S.G.B.D relationnels. On retrouve en particulier les opérations de produit, de sélection, de projection et d'intersection qui sont parmi les plus pratiquées. D'autres opérations : union, différence ... peuvent être facilement programmées en PROLOG avec un effort minimum.

Les accés à la base sont réalisés à l'aide d'un seul prédicat : RECHERCHE-MULT, qui comporte un seul argument : une liste de requètes à différentes relations de la base. L'exploitation de cette liste est séquentielle, l'ordre de rangement des éléments est donc important et conditionne les performances de réalisation. Dans un élément de cette liste on regroupe toutes les demandes relatives à une relation.Il se présente sous la forme d'un couple :

- nom de relation (n-r),

- liste de demandes pour cette relation (l-s).

L-S est elle même une liste de 4-uplets <n-c,<o,m-o>,<v-c,m-c>,v-t>.

- n-c est le nom d'un constituant de la relation,

- o est un opérateur de comparaison qui se présente sous la forme d'un identificateur : egal, gedq, grand, pteq, petit ou d'une variable libre.

- m-o et m-c sont des attributs qui sont unifiés aux valeurs d'identificateur : prs ou lbr.

- v-c est une valeur de comparaison ou une variable libre.

- v-t est unifiée à une valeur trouvée dans la base.

Une demande particulière concernant la recherche du constituant référence d'espèce des n-uplets de la relation IDENT dont le constituant référence de profil "id_ref" vaut 234 et le constituant probabilité "id_prob" est > 900 peut se présenter sous la forme :

```
<"ident",
<"id_prob",<grand,prs>,<900,prs>,v-prob>.
<"id_ref",<egal,prs>,<234,prs>,v-ref>.
<"id_comp",<o,lbr>,<v-c,lbr>,v-esp>.nil>>
```

La liste l-s comporte 2 sélections, la première porte sur le choix de n-uplets qui ont une probabilité > 900, la seconde sur le choix de n-uplets qui ont 234 comme référence de profil. La requète globale correspond à l'intersection des 2 relations. On obtient la relation résultat par projection de l'intersection sur les constituants probabilité (v-prob), référence de profil (v-ref), reférence d'espèce (v-esp).

D'une manière générale une requète relative à une relation sera toujours considérée comme l'intersection de relations produites par séléction dans la relation initiale.

L'intersection est réalisée de manière progressive en retenant à chaque fois parmi l'ensemble des n-uplets produits par les sélections précédentes ceux qui répondent à la nouvelle sélection.

Dans cette liste certaines demandes sont plus sélectives que d'autres. La demande relative à la référence de profil est à cet égard meilleure que la demande relative aux probabilités puisqu'elle conduira certainement au choix d'un moins grand nombre de n-uplets. Pour optimiser la réalisation des intersections successives on a donc intéret à commencer par la demande la plus précise, les autres demandes ne servant en fin de compte qu'à éliminer des n-uplets parmi ceux initialement choisis.

Le prédicat CHOIX-CHAMP réalise, en fonction de critères qui sont décrits ensuite et qui peuvent dépendre de la structure particulière d'une base, le choix de la sélection à mettre en oeuvre en premier lieu. Ce choix est effectué lors de la première rencontre d'un ensemble de sélection. Le résultat correspondant est alors rangé dans la base de faits relative à la B.D sous la forme d'un prédicat STRATEGIE-RECHERCHE.

RECHERCHE réalise la première sélection et RECHERCHE-MULT' les intersections succésives.

```
recherche-mult(<n-r,l-s-r>.l) ->
   si-sinon(strategie-recherche
            (n-r,l-s-r,<n-c,<o,m-o>,<v-c,m-c>,v-t>.l-s'),
      vrai,
      choix-champ(n-r,l-s-r,
                  <n-c,<o,m-o>,<v-c,m-c>,v-t>.l-s'))
   recherche(n-r,n-c,o,v-c,v-t)
   recherche-mult'(n-r,l-s')
   recherche-mult(l);
recherche-mult(nil) ->;
```

CHOIX-CHAMPS(n-r,l-s,l-s') détermine pour la relation n-r la stratégie d'utilisation l-s'de la liste de requètes l-s. Ce prédicat choisit dans l-s la demande la plus précise

qu'il place en tête de l-s'. Cette demande provoque la sélection dans la base d'un ensemble de n-uplets qui sont ou acceptés ou rejetés en fonction des requètes restantes. Les critères qui permettent le choix de la première sélection sont définis par le prédicat CLASSEMENT de la base de faits realtive au S.G.B.D. Ces critères dépendent de la nature et de la valeur des opérateurs et valeurs de sélection.

```
choix-champ(n-r,l-s,l-s') ->
   affecter(optichamp,999)
   choix-champ'(n-r,l-s,l)
   val(numchamp,j-c)
   permuter-champs(j-c,l-s,l-s')
   ajout-strategie-recherche(n-r,l-s,l-s');

choix-champ'(n-r,<n-c,<o,m-o>,<v-c,m-c>,v-t>.l-s,i) ->
   classement(c,n-r,n-c,o,v-c)
   val(inf(c,optichamp),l) / affecter(optichamp,c)
   affecter(numchamp,i)
   val(add(i,l),il)
   choix-champ'(n-r,l-s,il);
choix-champ'(n-r,e.l-s,i) ->
   val(add(i,l),il)
   choix-champ'(n-r,l-s,il);
choix-champ'(n-r,nil,i) ->;

permuter-champs(j-c,l-s,e.l-s') ->
   acces-liste-num(j-c,l,l-s,e,l-s');

acces-liste-num(j-c,j-c,e.l-s,e,l-s) -> /;
acces-liste-num(j-c,j,e.l-s,e',e.l-d) ->
   val(add(j,l),jl)
   acces-liste-num(j-c,jl,l-s,e',l-d);
```

La mise en forme précédente est ajoutée à la base de faits par l'intermédiaire du prédicat AJOUT STRATEGIE RECHERCHE. La liste de requètes (l-s) est généralisée (l-sl) en remplaçant les valeurs de comparaison (v-c) par des variables libres, l'attribut (prs ou lbr) est conservé. On obtient ainsi une forme qui peut être unifiée à toute demande comparable : mêmes constituants, mêmes opérateurs, même attribut de présence (prs) ou d'absence (lbr) des valeurs de comparaison. La liste mise en forme l-s' est généralisée de la même façon sous la forme l-sl'. Le prédicat STRATEGIE-RECHERCHE(n-r,l-sl,l-sl') est ajouté à la base de faits.

```
ajout-strategie-recherche(n-r,l-s,l-s') ->
   ajout-strategie-recherche'(l-s,l-s',l-sl,l-sl')
   unifie-strategie(l-sl,l-sl')
   ajout(<strategie-recherche(n-r,l-sl,l-sl'),nil>);

ajout-strategie-recherche'(<n-c,<o,m-o>,
```

```
      <v-c,m-c>,v-t>.l-s,<n-c',<o',m'>,<v-c',m-c'>,v-t'>.
      l-s',<n-c,<o,m-o>,<v-cl,m-c>,v-tl>.l-sl,
      <n-c',<o',m'>,<v-cl',m-c'>,v-tl'>.l-sl') ->
      ajout-strategie-recherche'(l-s,l-s',l-sl,l-sl');
   ajout-strategie-recherche'(nil,nil,nil,nil) ->;

   unifie-strategie(e.l-s,l-s') -> dans(e,l-s') /
      unifie-strategie(l-s,l-s');
   unifie-strategie(nil,l-s') ->;
```

Le prédicat RECHERCHE organise la sélection de n-uplets dans une relation (n-r). Elle repose sur la valeur (v-t) d'un constituant (n-c) de la relation que l'on compare (o) à une valeur de comparaison (v-c). Plusieurs cas sont à distinguer en fonction des valeurs et caractéristiques respectives de n-c,o et v-c.

```
   recherche(n-r,n-c,o,v-c,v-t) ->
      si-sinon(ou(non(prisl(v-c)),libre(n-c)),
         parcours-seq(n-r,n-c,v-t),
         recherche-val(n-r,n-c,o,v-c,v-t))
      valeur-compatible(v-t,o,v-c);
```

Un parcours séquentiel est à effectuer pour la relation (n-r) si la valeur de comparaison n'est pas précisée ou si le nom du constituant n'est pas mentionné. Dans ce dernier cas on effectue un parcours séquentiel de la relation en fonction de l'ordre de rangement des n-uplets dans la base.La réalisation de l'autre cas dépend de la nature du constituant : indexé ou non indexé. Si le constituant est indexé l'ordre de parcours est celui imposé par l'index sinon il s'agit d'un parcours séquentiel comme précédemment.

```
   parcours-seq(n-r,n-c,v-t) ->
      si-sinon(ou(non(prisl(n-c)),
         type-index(n-r,n-c,nonindex)),
         init-seq(n-r,sequentiel),
      init-ind(n-r,n-c))
      trouver-seq(n-r,premier)
      parcours-valeur(n-r,n-c,v-c,suivant,v-t);

   parcours-valeur(n-r,n-c,v-c,s,v-t) ->
      parcours-suite(n-r,s)
      vue(n-r,n-c.nil,v-t.nil);

   parcours-suite(n-r,s) ->;
   parcours-suite(n-r,s) ->
      trouver-seq(n-r,s)
      parcours-suite(n-r,s);
```

RECHERCHE-VAL concerne les cas où l'on doit retrouver un ensemble de valeurs en fonction d'un critère. On dispose d'une valeur comparaison (v-c) pour un constituant (n-c) de

la relation (n-r).

Si le constituant n'est pas indexé il faut réaliser la sélection à partir d'un parcours séquentiel des n-uplets de la relation : PARCOURS-SELECTION. On rejète ou on accepte les n-uplets trouvés par le prédicat VALEUR-COMPATIBLE.

Si le constituant est indexé on utilise cet index pour mettre en oeuvre la sélection : RECHERCHE-INDEX.

```
recherche-val(n-r,n-c,o,v-c,v-t) ->
   type-index(n-r,n-c,t-i)
   si-sinon(val(t-i,nonindex),
      parcours-selection(n-r,n-c,o,v-c,v-t),
      recherche-index(n-r,n-c,t-i,o,v-c,v-t));

parcours-selection(n-r,n-c,o,v-c,v-t) ->
   parcours-seq(n-r,n-c,v-t)
   valeur-compatible(v-t,o,v-c);
```

Pour une recherche indexée on utilise le constituant et la valeur de comparaison pour se positionner directement sur le premier n-uplet compatible : RECHERCHE-DEB. L'obtention éventuelle des n-uplets suivants dépend de la nature de l'opérateur (o) et du type (t-i) de l'index. PARCOURS-VALEUR et RECHERCHE-EGAL réalisent cette recherche.

```
recherche-index(n-r,n-c,t-i,o,v-c,v-t) ->
   sens-recherche(o,s)
   init-ind(n-r,n-c)
   recherche-deb(n-r,n-c,o,v-c)
   si-sinon(et(dif(o,egal),dif(o,comparaison)),
      parcours-valeur(n-r,n-c,v-c,s,v-t),
      ssi(val(t-i,dup),
         recherche-egal(n-r,n-c,o,v-c,v-t)));

recherche-deb(n-r,n-c,petit,v-c) ->
   trouver-cle(n-r,n-c,egal,v-c) /
   trouver-seq(n-r,precedent);
recherche-deb(n-r,n-c,petit,v-c) ->
   / recherche-deb(n-r,n-c,pteq,v-c);
recherche-deb(n-r,n-c,pteq,v-c) ->
   trouver-cle(n-r,n-c,grand,v-c) /
   trouver-seq(n-r,precedent);
recherche-deb(n-r,n-c,pteq,v-c) ->
   trouver-seq(n-r,dernier) /;
recherche-deb(n-r,n-c,o,v-c) ->
   trouver-cle(n-r,n-c,o,v-c);

recherche-egal(n-r,n-c,o,v-c,v-t) ->
   vue(n-r,n-c.nil,v-t.nil)
   recherche-egal'(n-r,n-c,o,v-c,v-t);

recherche-egal'(n-r,n-c,o,v-c,v-t) ->;
recherche-egal'(n-r,n-c,o,v-c,v-t) ->
```

```
      trouver-seq(n-r,suivant)
      vue(n-r,n-c.nil,v-t'.nil)
      valeur-compatible(v-t',o,v-c)
      recherche-egal'(n-r,n-c,o,v-c,v-t');

   sens-recherche(pteq,precedent) -> /;
   sens-recherche(petit,precedent) -> /;
   sens-recherche(x,suivant) -> /;
```

Lorsqu'un n-uplet est obtenu par le prédicat RECHERCHE son contenu est examiné par RECHERCHE-MULT'. Les sélections restantes sont prises en compte sequentiellement par le prédicat VALEUR-COMPATIBLE, y compris celles qui concernent les constituants structurés en champs élémentaires (les valeurs trouvées sont dans ce cas organisées sous forme de liste).

```
   recherche-mult'(n-r,<n-c,<o,m-o>,<v-c,m-c>,v-t>.l-s) ->
      vue(n-r,n-c.nil,v-t.nil)
      valeur-compatible(v-t,o,v-c)
      recherche-mult'(n-r,l-s);
   recherche-mult'(n-r,nil) -> ;

   valeur-compatible(v-t.l-t,o,v-c.l-c) ->
      / valeur-compatible1(v-t,o,v-c)
      valeur-compatible(l-t,o,l-c);
   valeur-compatible(nil,o,nil) -> /;
   valeur-compatible(v-t,o,v-c) ->
      valeur-compatible1(v-t,o,v-c);

   valeur-compatible1(v-t,egal,v-t) -> /;
   valeur-compatible1(v-t,o,v-c) ->
      ou(libre(o),non(pris1(v-c))) /;
   valeur-compatible1(v-t,egal,v-c) -> /
      si-sinon(chaine(v-c),
         strcmp(v-c,v-t,2),
         val(v-c,v-t));
   valeur-compatible1(v-t,comparaison,v-c) -> /
      si-sinon(chaine(v-c),
         strcmp(v-t,v-c,2),
         val(v-c,v-t));
   valeur-compatible1(v-t,gdeq,v-c) -> /
      si-sinon(chaine(v-c),
         ou(strcmp(v-t,v-c,2),strcmp(v-t,v-c,3)),
         val(inf(v-t,v-c),0));
   valeur-compatible1(v-t,grand,v-c) -> /
      si-sinon(chaine(v-c),
         strcmp(v-t,v-c,3),
         val(inf(v-c,v-t),1));
   valeur-compatible1(v-t,pteq,v-c) -> /
      si-sinon(chaine(v-c),
         ou(strcmp(v-t,v-c,2),strcmp(v-t,v-c,1)),
         val(inf(v-c,v-t),0));
```

```
valeur-compatiblel(v-t,petit,v-c) -> /
   si-sinon(chaine(v-c),
      strcmp(v-t,v-c,l),
      val(inf(v-t,v-c),l));
```

PRISL est une généralisation du prédicat PRIS de l'interpréteur PROLOG II qui peut s'appliquer à une liste d'éléments. Il est vrai si et seulement si tous les éléments sont pris (unifiés à une valeur au cours de la résolution).

```
prisl(v) -> libre(v) / echec;
prisl(x.l) -> / pris(x) prisl(l);
prisl(x) ->;
```

L'INTERFACE PROLOG_INFORMIX DE NIVEAU 1

Les prédicats prédéfinis de PROLOG II sont nommés dans un espace de travail spécifique (le monde SUPERVISEUR). Ils utilisent souvent les prédicats évaluables que l'on mentionne avec la syntaxe /?N où N représente le numéro interne du prédicat dans l'interpréteur (114 correspond au cas 14 de la fonction AUTRESPREDICATS).

Les prédicats prédéfinis de l'interface PROLOG INFORMIX ont été ajoutés à cet espace de travail. On les trouve en général sous deux formes :

- Une forme primitive qui comporte un argument de retour : a , qui correspond à la valeur de la fonction C équivalente. Cette valeur est égale à 0 si la réalisation s'est effectuée correctement. On obtient sinon un code erreur qu'il faut interpréter. Le prédicat est toujours évalué à vrai si les paramètres en entrée sont définis et du type correct.

- La deuxième forme est définie à partir de la première. Elle est privée de l'argument a. Le prédicat correspondant est évalué à vrai si le code retour est 0, faux sinon.

On utilise la syntaxe suivante dans le cadre des définitions :

i-d : identificateur,
i : entier,
n-b : nom de base de donnée,
c-f : nombre de relations d'une base,
n-f : nom de relation (fichier),
j-f : numéro d'ordre d'une relation,
c-c : nombre de composants d'un n-uplet de relation (de champs d'un enregistrement),
l-e : longueur d'un n-uplet de relation (taille d'un enregistrement),
j-c : numéro d'ordre d'un composant de n-uplet (d'un champ d'enregistrement),
n-c : nom d'un composant,
t-c : type d'un composant,
l-c : taille d'un composant,
t-i : type d'un index (unique, non unique, absent),
v : valeur d'un composant,
l-v : liste de valeurs de composants,

o : opérateur de comparaison,
l-n-c : liste de noms de composants.

La présentation suivante rappelle les définitions introduites dans le cadre de l'annexe PREDICATS EVALUABLES qui font le lien avec l'interface INFORMIX-langage C . Les prédicats prédéfinis sont ensuite présentés sous leur deux aspects.

DBSELECT(indic,nom)
DBSELECT(x-indic,x-nom,x-retf)

```
select(i,n-b,a) -> entier(i) chaine(n-b) /?114;
select(i-d,n-b) ->
   base-cle(i-d,i)
   select(i',n-b,a)
   eg(a,0);

base-cle(ouvredb,1) -> /;
base-cle(fermedb,2) -> /;
base-cle(ouvrefich,3) -> /;
base-cle(fermefich,4) -> /;
```

DBSELECT(DBOPEN,nombase)
OUVRIR-BD(x-nombase,x-retf))

```
ouvrir-bd(n-b,a) -> chaine(n-b) /?115;
ouvrir-bd(n-b) ->
   ouvrir-bd(n-b,a)
   eg(a,0);
```

DBSELECT(DBCLOSE,nombase)
FERMER-BD(x-nombase,x-retf)

```
fermer-bd(n-b,a) -> chaine(n-b) /?116;
fermer-bd(n-b) ->
   fermer-bd(n-b,a)
   eg(a,0);
```

NB-FICH(x-nbrel)
donne le nombre de relations dans la base

```
nbre-fich'(c-f) -> /?130;

nbre-fich(c-f) -> nbre-fich'(c-f') eg(c-f,c-f');
```

DBSELECT(FILEOPEN,nomfich)
OUVRIR-FICH(x-nomfich,x-retf)
et on définit une vue sur l'ensemble des composants

```
ouvrir-fich(n-f,a) -> chaine(n-f) /?117;
```

```
      ouvrir-fich(n-f) ->
         ouvrir-fich(n-f,a)
         eg(a,0);

DBSELECT(FILECLOSE,nomfich)
FERMER-FICH(x-nomfich,x-retf)


      fermer-fich(n-f,a) -> chaine(n-f) /?118;
      fermer-fich(n-f) ->
         fermer-fich(n-f,a)
         eg(a,0);

DBNFILE
  (&numfich,nomfich,&nbrchamp,&tailenr,&perms)
INFO-NUMFICH
  (x-numfich,x-retf,x-nomfich,x-nbrchamp,x-tailenr)

      info-numfich(j-f,a,n-f,c-c,l-e) -> entier(j-f) /?119;
      info-numfich(j-f,n-f,c-c,l-e) ->
         info-numfich(j-f,a,n-f',c-c',l-e')
         eg(n-f',n-f)
         eg(c-c',c-c)
         eg(l-e',l-e)
         eg(a,0);

DBNFIELD
  (numchamp,nomfich,nomchamp,typ,long,perms)
INFO-NUMCHAMP
  (x-nomfich,x-numchamp,x-retf,x-nomchamp,x-typ,x-long)

      info-numchamp(n-f,j-c,a,n-c,t-c,l-c) ->
         chaine(n-f)  entier(j-c) /?120;
      info-numchamp(n-f,j-c,n-c,t-c,l-c) ->

         info-numchamp(n-f,j-c,a,n-c',t-c,l-c)
         eg(n-c',n-c)
         eg(a,0);

DBNCOMPOSITE
  (numchamp,nomfich,nomchamp,nom,typ,long,perms)
INFO-COMPOSITE
  (x-nomfich,x-nomchamp,x-numchamp,
   x-retf,x-nom,x-typ,x-long)

      info-composite(n-f,n-c,j-c,a,n-c',t-c,l-c) ->
         chaine(n-f) chaine(n-c) entier(j-c) /?133;
      info-composite(n-f,n-c,j-c,n-c',t-c,l-c) ->
         info-composite(n-f,n-c,j-c,a,n,t-c,l-c)
         eg(n-c',n)
         eg(a,0);

DBFIELD(nomfich,nomchamp,&typ,&lgu,&perms)
```

INFO-CHAMP(x-nomfich,x-nomchamp,x-retf,x-typ,x-lgu)

```
info-champ(n-f,n-c,a,t-c,l-c) -> chaine(n-f) chaine(n-c)
   /?121;
info-champ(n-f,n-c,t-c,l-c) ->
   info-champ(n-f,n-c,a,t-c',l-c')
   eg(a,0)
   type-champ(t-c,t-c')
   eg(l-c',l-c);

type-champ(cartype,0) -> /;
type-champ(enttype,1) -> /;
type-champ(longtype,2) -> /;
type-champ(serietype,258) -> /;
type-champ(datetype,514) -> /;
type-champ(comptype,10) -> /;
```

DBFILE(nomfich,&numchamp,&tailenr,&perms)
INFO-FICH(x-nomfich,x-retf,x-numchamp,x-tailenr)

```
info-fich(n-f,a,c-c,l-e) -> chaine(n-f) /?122;
info-fich(n-f,c-c,l-e) ->
   info-fich(n-f,a,c-c',l-e')
   eg(a,0)
   eg(c-c',c-c)
   eg(l-e',l-e);
```

DBALIAS(nom,ancnom)
RENOMME-FICH(x-nom,x-ancnom,x-retf)

```
renomme-fich(n-f,n-f',a) -> chaine(n-f) chaine(n-f')
   /?126;
renomme-fich(n-f,n-f') ->
   renomme-fich(n-f,n-f',a)
   eg(a,0);
```

DBINDEX(nomfich,nomchamp,&typindex)
INFO-INDEX(x-nomfich,x-nomchamp,x-retf,x-typindex)

```
info-index(n-f,n-c,a,t-i) -> chaine(n-f) chaine(n-c)
   /?127;
info-index(n-f,n-c,t-i) ->
   info-index(n-f,n-c,a,t-i')
   eg(a,0)
   index-cle(t-i,t-i');

index-cle(nonindex,0) -> /;
index-cle(nondup,1) -> /;
index-cle(dup,2) -> /;
```

DBADDINDEX(nomfich,nomchamp,indic)
ADD-INDEX(x-nomfich,x-nomchamp,x-indic,x-retf)

```
      add-index(n-f,n-c,t-i,a) -> chaine(n-f) chaine(n-c)
         /?128;
      add-index(n-f,n-c,t-i) ->
         index-cle(t-i,t-i')
         add-index(n-f,n-c,t-i',a)
         eg(a,0);
```

DBDELINDEX(nomfich,nomchamp)
DEL-INDEX(x-nomfich,x-nomchamp,x-retf)

```
      del-index(n-f,n-c,a) -> chaine(n-f) chaine(n-c) /?129;
      del-index(n-f,n-c) ->
         del-index(n-f,n-c,a)
         eg(a,0);
```

DBSELFIELD(nomfich,NULL,indic)
INIT-SEQ(x-nomfich,x-indic,x-retf)

```
      init-seq(n-f,o,a) -> chaine(n-f) /?132;
      init-seq(n-f,o) ->
         acces-cle(o,o')
         init-seq(n-f,o',a)
         eg(a,0);

      acces-cle(primaire,1) -> /;
      acces-cle(sequentiel,2) -> /;
```

DBSELFIELD(nomfich,nomchamp,ACCKEYED)
INIT-IND(x-nomfich,x-nomchamp,x-retf)

```
      init-ind(n-f,n-c,a) -> chaine(n-f) chaine(n-c) /?131;
      init-ind(n-f,n-c) ->
         init-ind(n-f,n-c,a)
         eg(a,0);
```

DBFIND(nomfich,operateur,val,long,ptenregis)
TROUVER-SEQ(x-nomfich,x-operateur,x-retf)

```
      trouver-seq(n-f,o,a) -> chaine(n-f) /?134;
      trouver-seq(n-f,o) ->
         oper-cle(o,o')
         trouver-seq(n-f,o',a)
         eg(a,0);

      oper-cle(comparaison,1) -> /;
      oper-cle(premier,2) -> /;
      oper-cle(dernier,3) -> /;
      oper-cle(suivant,4) -> /;
      oper-cle(precedent,5) -> /;
      oper-cle(egal,6) -> /;
      oper-cle(gdeq,7) -> /;
      oper-cle(grand,8) -> /;
      oper-cle(courant,9) -> /;
```

DBFIND(nomfich,operateur,val,long,ptenregis)
TROUVER-CLE
(x-nomfich,x-nomchamp,x-operateur,x-val,x-retf)

```
trouver-cle(n-f,n-c,o,v,a) -> /?135;
trouver-cle(n-f,n-c,o,l-v) ->
   oper-cle(o,o')
   info-champ(n-f,n-c,comptype,c-c)
   / trouver-cle-comp(n-f,n-c,o',0,c-c,l-v);
trouver-cle(n-f,n-c,o,v) ->
   chaine(n-f)
   chaine(n-c)
   ent-chaine(v)
   oper-cle(o,o')
   trouver-cle(n-f,n-c,o',v,a)
   eg(a,0);

trouver-cle-comp(n-f,n-c,o,c-c,c-c,nil) ->
   / trouver-cle(n-f,n-c,o,v,a) eg(a,0);
trouver-cle-comp(n-f,n-c,o,i-c,c-c,v.l-v) ->
   info-composite(n-f,n-c,i-c,n-c',t-c,l-c)
   entrer-champ(n-f,n-c',v)
   val(add(i-c,1),i-c')
   / trouver-cle-comp(n-f,n-c,o,i-c',c-c,l-v);

ent-chaine(v) -> chaine(v) /;
ent-chaine(v) -> entier(v) /;
```

CHERCHE-CHAMP(x-nomfich,x-nomchamp,x-v,x-retf)
donne la valeur d'un composant de relation

```
cherche-champ(n-f,n-c,v,a) -> chaine(n-f) chaine(n-c)
   /?136;
cherche-champ(n-f,n-c,l-v) ->
   info-champ(n-f,n-c,comptype,c-c)
   / cherche-liste-champ(n-f,n-c,0,c-c,l-v);
cherche-champ(n-f,n-c,v) -> cherche-champ(n-f,n-c,v',a)
   eg(v',v) eg(a,0);

cherche-liste-champ(n-f,n-c,c-c,c-c,nil) -> /;
cherche-liste-champ(n-f,n-c,i-c,c-c,v.l-v) ->
   info-composite(n-f,n-c,i-c,n-c',t-c,l-c)
   cherche-champ(n-f,n-c',v',a) eg(v',v) eg(a,0)
   val(add(i-c,1),i-c') /
   cherche-liste-champ(n-f,n-c,i-c',c-c,l-v);

vue(n-f,nil,nil) ->/;
vue(n-f,x.l-n-c,y.l-v) -> cherche-champ(n-f,x,y) /
   vue(n-f,l-n-c,l-v);
vue(n-f,x.l-n-c,y.l-v) -> vue(n-f,l-n-c,l-v);
```

DBADD(nomfich,ptenregis)

ADD-ENR'(x-nomfich,x-vl,...,x-vn,x-retf)
les valeurs des composants sont dans x-vl...x-vn

```
add-enr'(n-f,vl,a) -> /?137;
add-enr'(n-f,vl,v2,a) -> /?137;
add-enr'(n-f,vl,v2,v3,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,v5,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,v5,v6,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,v8,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,v8,v9,a) -> /?137;
add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,v8,v9,vl0,a) -> /?137;

add-enr(n-f,vl) ->
   add-enr'(n-f,vl,a)
   eg(a,0);
add-enr(n-f,vl,v2) ->
   add-enr'(n-f,vl,v2,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3) ->
   add-enr'(n-f,vl,v2,v3,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4) ->
   add-enr'(n-f,vl,v2,v3,v4,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4,v5) ->
   add-enr'(n-f,vl,v2,v3,v4,v5,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4,v5,v6) ->
   add-enr'(n-f,vl,v2,v3,v4,v5,v6,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4,v5,v6,v7) ->
   add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4,v5,v6,v7,v8) ->
   add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,v8,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4,v5,v6,v7,v8,v9) ->
   add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,v8,v9,a)
   eg(a,0);
add-enr(n-f,vl,v2,v3,v4,v5,v6,v7,v8,v9,vl0) ->
   add-enr'(n-f,vl,v2,v3,v4,v5,v6,v7,v8,v9,vl0,a)
   eg(a,0);
```

DBUPDATE(nomfich,ptenregis)
MODIF-ENR(x-nomfich,x-retf)

```
modif-enr(n-f,a) -> chaine(n-f) /?138;
modif-enr(n-f) -> modif-enr(n-f,a) eg(a,0);
```

ENTRER-CHAMP(x-nomfich,x-nomchamp,x-v,x-retf)
permet de redéfinir la valeur d'un composant de relation

pour mise à jour ultérieure

```
      entrer-champ(n-f,n-c,v,a) ->
         chaine(n-f)
         chaine(n-c)
         ent-chaine(v)
         /?139;
      entrer-champ(n-f,n-c,v) ->
         entrer-champ(n-f,n-c,v,a)
         eg(a,0);
```

DBDELETE(nomfich)
DEL-ENR(x-nomfich,x-retf)

```
      del-enr(n-f,a) -> chaine(n-f) /?123;
      del-enr(n-f) -> del-enr(n-f,a) eg(a,0);
```

LES PREDICATS EVALUABLES DE L'INTERPRETEUR

PROLOG II est un langage interprété qui utilise un certain nombre de prédicats prédéfinis qui permettent, entre autre, le réalisation de certaines fonctions relatives à l'environnement de programmation (entrees-sorties, gestion des énoncés, trace ....). Certains de ces prédicats sont réalisés en PROLOG, d'autres, plus primitifs, sont définis dans l'interpréteur lui-même : les prédicats évaluables. Une des caractéristiques essentielle de PROLOG II est de permettre à un utilisateur compétent de modifier ou d'ajouter des prédicats évaluables à ceux qui existent déjà. On peut ainsi étendre l'environnement de programmation et se tailler un PROLOG sur mesure qui répond à des besoins spécifiques. PROLOG II fournit dans ce but une fonction écrite en langage C : AUTRESPREDICATS.

Dans cette fonction chaque prédicat évaluable est identifié par un numéro. On communique avec le reste de l'interpréteur par l'intermédiaire de certaines variables et fonctions externes. Les variables sont du type ENTM ou ENTC, ENTM coorespond à un mot de 32 bits et ENTC à un mot de 16 bits.

A l'entrée de la procédure la variable externe, de type entc, R1 contient le numéro de prédicat appelé lors de la résolution PROLOG.

La fonction GETZ permet d'obtenir la valeur du paramètre courant du prédicat à évaluer. S'il s'agit d'un entier : getz(entier), la variable externe de type entm, RR1 permet d'en acquérir la valeur (un entier PROLOG est codé sur 32 bits, les 8 bits de gauche forment un préfixe qui donne le type de la constante, les 24 bits de droite contiennent la valeur proprement dite). S'il s'agit d'une chaine de caractères PROLOG: getz(chaine), on dispose dans RR1 d'un pointeur sur cette chaine. La transformation en chaine C est réalisée en utilisant la fonction CHAINEC (le codage n'est pas le même).

La fonction RANGER permet la transmission d'une valeur au prédicat à évaluer. Ranger comporte trois arguments:

- un pointeur (entm) sur le paramètre du prédicat (SUBST),

- une valeur (entm) à transmettre (entier PROLOG ou chaine PROLOG),

-une indication du type : chaine ou entier.

Dans le cas d'une chaine, on peut utiliser préalablement la fonction CHAINEP pour transformer une chaine C en chaine PROLOG.

Pour passer d'un paramètre au paramètre suivant on utilise la fonction PARAMETRE_SUIVANT qui met à jour la variable externe SUBST.

Ces facilités on été utilisées pour définir une extension de l'interpréteur PROLOG II qui réalise un interface avec le système de géstion de base de données relationnel INFORMIX. On dispose d'origine avec INFORMIX d'une interface avec le langage C fournie sous la forme d'un ensemble de fonctions. Elles réalisent les accès et les mises à jour dans une base sous une forme trés primitive. On trouvera dans la notice INFORMIX [INFOR 84] le détail relatif à l'utilisation de chaque fonction. L'interface avec PROLOG est définie par un ensemble de prédicats évaluables ajoutés à la fonction AUTRESPREDICATS de l'interpréteur qui correspondent sensiblement à chacune des fonctions de l'interface C.

Un prototype de l'interface avait été écrit par C.OLIVE dans le cadre d'un mémoire de DEA. Ce travail a été repris pour donner naissance à cette version définitive

Le fichier PREDCL.H contient toutes les déclarations relatives à l'interpréteur PROLOG, DBIO.H celles relatives à l'interface C-INFORMIX. Certaines de ces déclarations sont utilisées dans la fonction AUTRESPREDICATS.

Pour chaque cas on trouvera en commentaire le nom et les arguments de la fonction C de l'interface INFORMIX et le nom et les arguments du prédicat. Les noms des paramètres PROLOG corespondent aux noms des paramètres des fonctions C, ils sont simplement préfixés par "x-" pour obéir à la syntaxe PROLOG II. Le paramètre x-retf correspond quant-à lui à la valeur de la fonction proprement dite, cette valeur est dans la majorité des cas un code retour.

```
#include "predcl.h"
#include "dbio.h"
#define lgenrmax 500
#define lgchaine 81
#define nbchamps 20

autrespredicats()
{
int i,n,retf,operateur;
entm pp,ival;
entc  typindex,lgu,perms,
      tailenr,numfich,typ,j;
entc  indic,numchamp,nbc;
```

```
char c,ancnom[lgchaine],nom[lgchaine],nombase[lgchaine],
     nomchamp[lgchaine],nomfich[lgchaine],
     val[lgchaine],*ptr;

static short nf;

char listchamp[nbchamps][lgchaine],*ptrchamp[nbchamps];

switch(rl){
   .
   .
   .
case 14:

  /* DBSELECT(indic,nom)
     SELECT(x-indic,x-nom,x-retf)
  ******************************/

  getz(entier);
  indic=rrl & 0x00ffffffL;
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nom);
  retf=dbselect(indic,nom);
  retf=1;
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 15:

  /* DBSELECT(DBOPEN,nombase)
     OUVRIR-BD(x-nombase,x-retf))
  ******************************/

  getz(chaine);
  chainec(rrl,nombase);
  retf=dbselect(DBOPEN,nombase);
  for (nf=0;!dbnfile(nf,nom,&nbc,&tailenr,&perms);nf++);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 16:

  /* DBSELECT(DBCLOSE,nombase)
     FERMER-BD(x-nombase,x-retf)
  ******************************/

  getz(chaine);
```

```
  chainec(rrl,nombase);
  retf=dbselect(DBCLOSE,nombase);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 17:

  /* DBSELECT(FILEOPEN,nomfich)
     OUVRIR-FICH(x-nomfich,x-retf)
  et on définit une vue sur l'ensemble des composants
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  retf=dbselect(FILEOPEN+EXCLUSIVE,nomfich);
  if (retf==0) {
    for (j=0;
      !dbnfield(j,nomfich,listchamp[j],&typ,&lgu,&perms)
      ;j++)
    ptrchamp[j]=listchamp[j];
    retf=dbsetfileview(nomfich,ptrchamp,(int)j);
  }
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;

case 18:

  /* DBSELECT(FILECLOSE,nomfich)
     FERMER-FICH(x-nomfich,x-retf)
  ******************************/

  getz(chaine);
  chainec(rrl,nomfich);
  retf=dbselect(FILECLOSE,nomfich);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 19:

  /* DBNFILE
       (&numfich,nomfich,&nbrchamp;&tailenr,&perms)
     INFO-NUMFICH
       (x-numfich,x-retf,x-nomfich,x-nbrchamp,x-tailenr)
  ************************************************/

  getz(entier);
  numfich=rrl & 0x00ffffffL ;
  retf=dbnfile(numfich,nomfich,&numchamp,&tailenr,&perms);
```

```
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  if (retf==0) {
    parametre_suivant;
    chainep(nomfich,&pp);
    ranger(subst,pp,chaine);
    parametre_suivant;
    ranger(subst,(entm)numchamp,entier);
    parametre_suivant;
    ranger(subst,(entm)tailenr,entier);
  }
  break;


case 20:

  /* DBNFIELD
       (numchamp,nomfich,nomchamp,typ,long,perms)
     INFO-NUMCHAMP
       (x-nomfich,x-numchamp,x-retf,x-nomchamp,x-typ,x-long)
  ************************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(entier);
  numchamp=rrl & 0x00ffffffL;
  retf=dbnfield (numchamp,nomfich,nomchamp,&typ,&lgu,&perms);
  if (typ<0) typ=10;
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  if (retf==0) {
    parametre_suivant;
    chainep(nomchamp,&pp);
    ranger(subst,pp,chaine);
    parametre_suivant;
    ranger(subst,(entm)typ,entier);
    parametre_suivant;
    ranger(subst,(entm)lgu,entier);
  }
  break;


case 21:

  /* DBFIELD(nomfich,nomchamp,&typ,&lgu,&perms)
     INFO-CHAMP(x-nomfich,x-nomchamp,x-retf,x-typ,x-lgu)
  ************************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
```

```
  chainec(rrl,nomchamp);
  parametre_suivant;
  retf=dbfield(nomfich,nomchamp,&typ,&lgu,&perms);
  if (typ<0) typ=10;
  ranger(subst,(entm)retf,entier);
  if (retf==0) {
    parametre_suivant;
    ranger(subst,(entm)typ,entier);
    parametre_suivant;
    ranger(subst,(entm)lgu,entier);
  }
  break;


case 22:

  /* DBFILE(nomfich,&numchamp,&tailenr,&perms)
     INFO-FICH(x-nomfich,x-retf,x-numchamp,x-tailenr)
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  retf=dbfile (nomfich,&numchamp,&tailenr,&perms);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  if (retf==0) {
    parametre_suivant;
    ranger(subst,(entm)numchamp,entier);
    parametre_suivant;
    ranger(subst,(entm)tailenr,entier);
  }
  break;


case 23:

  /* DBDELETE(nomfich)
     DEL-ENR(x-nomfich,x-retf)
  ***************************/

  getz(chaine);
  chainec(rrl,nomfich);
  retf=dbdelete (nomfich);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 24:

  /* DBLOCK(nomfich)
     VERROUILLE-FICH(x-nomfich,x-retf)
  ***********************************/
```

```
  printf("Non implante sur IBM PC");
  break;


case 25:

  /* DBUNLOCK(nomfich)
     DEVERROUILLE-FICH(x-nomfich,x-retf)
  ************************************/

  printf("Non implante sur IBM PC");
  break;


case 26:

  /* DBALIAS(nom,ancnom)
     RENOMME-FICH(x-nom,x-ancnom,x-retf)
  ************************************/

  getz(chaine);
  chainec(rrl,nom);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,ancnom);
  retf=dbalias (nom,ancnom);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 27:

  /* DBINDEX(nomfich,nomchamp,&typindex)
     INFO-INDEX(x-nomfich,x-nomchamp,x-retf,x-typindex)
   ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  retf=dbindex (nomfich,nomchamp,&typindex);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  if (retf==0) {
    parametre_suivant;
    ranger(subst,(entm)typindex,entier);
  }
  break;
```

```
case 28:

  /* DBADDINDEX(nomfich,nomchamp,indic)
     ADD-INDEX(x-nomfich,x-nomchamp,x-indic,x-retf)
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  parametre_suivant;
  getz(entier);
  indic=rrl & 0x00ffffffL;
  retf=dbaddindex (nomfich,nomchamp,indic);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 29:

  /* DBDELINDEX(nomfich,nomchamp)
     DEL-INDEX(x-nomfich,x-nomchamp,x-retf)
  ***************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  parametre_suivant;
  retf=dbdelindex (nomfich,nomchamp);
  ranger(subst,(entm)retf,entier);
  break;

case 30:

  /* NB-FICH(x-nbrel)
     donne le nombre de relations dans la base
  ******************************************/

  rrl=nf;
  ranger(subst,rrl,entier);
  break;

case 31:

  /* DBSELFIELD(nomfich,nomchamp,ACCKEYED)
     INIT-IND(x-nomfich,x-nomchamp,x-retf)
  ***************************************/

  getz(chaine);
```

```
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  retf=dbselfield(nomfich,nomchamp,ACCKEYED);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 32:

  /* DBSELFIELD(nomfich,NULL,indic)
     INIT-SEQ(x-nomfich,x-indic,x-retf)
  ************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(entier);
  operateur=rrl & 0x00ffffffL;
  retf=dbselfield(nomfich,0,operateur);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;

case 33:

  /* DBNCOMPOSITE
       (numchamp,nomfich,nomchamp,nom,typ,lcng,perms)
     INFO-COMPOSITE
       (x-nomfich,x-nomchamp,x-numchamp,
        x-retf,x-nom,x-typ,x-long)
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  parametre_suivant;
  getz(entier);
  numchamp=rrl & 0x00ffffffL;
  nom[0]='\0';
  retf=dbncomposite
         (numchamp,nomfich,nomchamp,nom,&typ,&lgu,&perms);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  if (retf==0) {
    chainep(nom,&pp);
    parametre_suivant;
    ranger(subst,pp,chaine);
    parametre_suivant;
```

```
    ranger(subst,(entm)typ,entier);
    parametre_suivant;
    ranger(subst,(entm)lgu,entier);
  }
  break;


case 34:

  /* DBFIND(nomfich,operateur,val,long,ptenregis)
     TROUVER-SEQ(x-nomfich,x-operateur,x-retf)
  ***********************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(entier);
  operateur=rrl & 0x00ffffffL;
  retf=dbfind(nomfich,operateur,0,&lgu,red);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 35:

  /* DBFIND(nomfich,operateur,val,long,ptenregis)
     TROUVER-CLE
        (x-nomfich,x-nomchamp,x-operateur,x-val,x-retf)
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  retf=dbfield(nomfich,nomchamp,&typ,&lgu,&perms);
  parametre_suivant;
  getz(entier);
  operateur=rrl & 0x00ffffffL;
  parametre_suivant;
  if (retf==0)
    switch (typ) {
      case CHARTYPE:
        getz(chaine);
        chainec(rrl,val);
        for (j=strlen(val);j<lgu;j++) val[j]=' ';
        val[lgu]='\0';
        retf=dbfind(nomfich,operateur,val,&lgu,red);
        break;
      case COMPOSTYPE:
        retf=dbfind(nomfich,operateur,0,&lgu,red);
        break;
```

```
      case INTTYPE:
        getz(entier);
        n=rrl & 0x00ffffffL;
        retf=dbfind(nomfich,operateur,&n,&lgu,red);
        break;
      default:
        getz(entier);
        rrl=rrl & 0x00ffffffL;
        retf=dbfind(nomfich,operateur,&rrl,&lgu,red);
    }
   parametre_suivant;
   ranger(subst,(entm)retf,entier);
  break;


case 36:

  /* CHERCHE-CHAMP(x-nomfich,x-nomchamp,x-v,x-retf)
  donne la valeur d'un composant de relation
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  retf=dbfield(nomfich,nomchamp,&typ,&lgu,&perms);
  parametre_suivant;
  if (retf==0) {
    ptr=red;
    for (numchamp=0;
      (dbnfield(numchamp,nomfich,nom,&typ,&lgu,&perms)==0)&&
      (rstrcmp(nom,nomchamp)!=0);
      numchamp++)
      if (typ!=COMPOSTYPE) ptr+=lgu;
    switch (typ) {
      case INTTYPE:
        rbytecpy(ptr,&j,2);
        ranger(subst,(entm)j,entier);
        break;
      case CHARTYPE:
        rbytecpy(ptr,val,(int)lgu);
        val[lgu]='\0';
        chainep(val,&pp);
        ranger(subst,pp,chaine);
        break;
      case COMPOSTYPE:
        retf=-1;
        break;
      default:
        rbytecpy(ptr,&ival,4);
        ranger(subst,ival,entier);
    }
```

```
  }
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 37:

  /* DBADD(nomfich,ptenregis)
     ADD-ENR'(x-nomfich,x-v1,...,x-vn,x-retf)
  les valeurs des composants sont dans x-v1...x-vn
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  retf=dbfile(nomfich,&nbc,&tailenr,&perms);
  if (retf!=0) nbc=-1;
  ptr=red;
  for(numchamp=0;numchamp<nbc;numchamp++) {
    retf=dbnfield
           (numchamp,nomfich,nomchamp,&typ,&lgu,&perms);
    if (retf==0)
      switch (typ) {
        case CHARTYPE:
          parametre_suivant;
          getz(chaine);
          chainec(rrl,val);
          n=strlen(val);
          for(j=n;j<lgu;j++) val[j]=' ';
          rbytecpy(val,ptr,(int)lgu);
          ptr=ptr+lgu;
          break;
        case INTTYPE:
          parametre_suivant;
          getz(entier);
          j=rrl & 0x00ffffffL;
          rbytecpy(&j,ptr,2);
          ptr+=2;
          break;
        case COMPOSTYPE:
          break;
        case SERIALTYPE:
          ptr+=4;
          break;
        default:
          parametre_suivant;
          getz(entier);
          ival=rrl & 0x00ffffffL;
          rbytecpy(&ival,ptr,4);
          ptr+=4;
      }
  }
  if (retf==0) retf=dbadd(nomfich,red);
```

```
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 38:

  /* DBUPDATE(nomfich,ptenregis)
     MODIF-ENR(x-nomfich,x-retf)
  ****************************/

  getz(chaine);
  chainec(rrl,nomfich);
  retf=dbupdate(nomfich,red);
  parametre_suivant;
  ranger(subst,(entm)retf,entier);
  break;


case 39:

  /* ENTRER-CHAMP(x-nomfich,x-nomchamp,x-v,x-retf)
  permet de redéfinir la valeur d'un composant de relation
  pour mise à jour ultérieure
  ************************************************/

  getz(chaine);
  chainec(rrl,nomfich);
  parametre_suivant;
  getz(chaine);
  chainec(rrl,nomchamp);
  parametre_suivant;
  retf=dbfield(nomfich,nomchamp,&typ,&lgu,&perms);
  if (retf==0) {
    ptr=red;
    for (numchamp=0;
      (dbnfield(numchamp,nomfich,nom,&typ,&lgu,&perms)==0)&&
      (rstrcmp(nom,nomchamp)!=0);
      numchamp++)
      if (typ!=COMPOSTYPE) ptr+=lgu;
    switch (typ) {
      case CHARTYPE:
        getz(chaine);
        chainec(rrl,val);
        n=strlen(val);
        for(j=n;j<lgu;j++) val[j]=' ';
        rbytecpy(val,ptr,(int)lgu);
        break;
      case INTTYPE:
        getz(entier);
        j=rrl & 0x00ffffffL;
        rbytecpy(&j,ptr,2);
        break;
```

```
        case COMPOSTYPE:
          retf=-1;
          break;
        case SERIALTYPE:
          retf=-1;
          break;
        default:
          getz(entier);
          ival=rrl & 0x00ffffffL;
          rbytecpy(&ival,ptr,4);
      }
    }
    parametre_suivant;
    ranger(subst,(entm)retf,entier);
    break;
}
avancer();
if ( bretoux==true ) return(0);
cond=true;return(0);
}
```

REFERENCES BIBLIOGRAPHIQUES

[DYB 68] W. DYBOWSKI, D.A. FRANKLIN
"Conditional Probability and the identification of bacteria : A pilot study"
Journal of general Microbiology 54, pp. 215-229
1968

[FRIED 73] R.B. FRIEDMAN, D. BRUCE, J. MAC LOWRY, VEE BRENER
"Computer assisted identification of bacteria"
American Journal of Clinical Pathology 60,
pp. 315-403; 1973

[LAP 73] S.P. LAPAGE, B. BASCOMB, W.R. WILLCOX, M.A. CURTIS
"Identification of bacteria by computer : general aspects and perspectives"
Journal of General Microbiology 77, pp. 273-290, 1973

[TER 76] M. TERRENOIRE, D. TOUNISSOUX
"Inequalities using Batthacharya distance and applications to decision process"
Proceeding of 3rd International joint Conference on Pattern Recognition, CORONADO, California, p. 242,
1976

[GAV1 76] F. GAVINI, B. LEFEBVRE
"Positions taxonomiques d'enterobacteries H_2S^- par rapport au genre CITROBACTER"
Annales de Microbiologie (Institut Pasteur). 127 A,
pp. 275-295, 1976

[GAV2 76] F. GAVINI, C. FERRAGUT, B. LEFEBVRE, H. LECLERC
"Etude taxonomique d'enterobactéries appartenant ou apparentées au genre ENTEROBACTER"
Annales de Microbiologie (Institut Pasteur), 127 B,
pp. 317-335, 1976

[GAV 77] F. GAVINI, H. LECLERC, B. LEFEBVRE, C. FERRAGUT, D. IZARD
"Etude taxonomique d'enterobactéries appartenant ou apparentées au genre KLEBSIELLA"
Annales de Microbiologie (Institut Pasteur), 128 B,
pp. 45-49, 1977

[GAV1 80] F. GAVINI, C. FERRAGUT, D. IZARD, P.A. TRINEL, H. LECLERC, B. LEFEBVRE, D.A.A. MOSSEL
"SERRATIA FONTICOLA, a new species from water"
International Journal of Systemic Bacteriology, 292, pp. 92-101, 1980

[GAV2 80] F. GAVINI, B. LEFEBVRE,
"Etude d'un nouveau système multitest pour l'identification des enterobactéries des aliments"
Actes du Colloque International de Microbiologie alimentaire, LILLE, 1980

[TOUN 80] D. TOUNISSOUX
"Processus séquentiels adaptatifs de reconnaissance des formes pour l'aide au diagnostic"
Thèse de Doctorat ès Sciences Mathématiques
LYON, 1980

[GAV1 81] F. GAVINI, D. IZARD, P.A. TRINEL, B. LEFEBVRE, H. LECLERC
"Etude taxonomique d'entérobactéries appartenant ou apparentées à l'espèce ESCHERICHIA COLI"
Canadian Journal of Microbiology, 1, p. 98, 1981

[GAV 82] F. GAVINI, C. OGER, B. LEFEBVRE, D. IZARD, H. LECLERC
"Développement of a computer identification system for Coliform strains"
Journal of Applied Bacteriology, 52, pp. 329-332, 1982

[LEF1 82] B. LEFEBVRE, F. GAVINI
"Theory and programming of a computer identification system for Coliform strains"
Journal of Applied Becteriology, 52, pp. 325-328, 1982

[LEF2 82] B. LEFEBVRE
"Un système de sélection et d'identification. Application à la reconnaissance des souches bactériennes"
Proceedings of the IASTED International Symposium Modeling Identification and Control, DAVOS, pp. 91-94, Mars 1982

[DEL 82] C. DELOBEL, M. ADIBA
"Bases de Données et systèmes relationnels"
DUNOD, Novembre 1982

[NIC 83] J.M. NICOLAS, K. YAZDANIAN
"Un aperçu de Bdgen : Un SGBD déductif"
Conférence IFIP, 1983

[MARC 83] A. MARCOUX, C. PERCEBOIS, P. BAZEX
"Querylog : Un système de bases de données qui utilise la déduction"
Proc. Journées BIGRE, CAP D'ADGE, 1983

[LEF 83] B. LEFEBVRE, F. GAVINI
"Système expert en identification bactériologique"
Actes Colloques de l'INSERM : Les bacilles à Gram -, Taxonomie, Identification, Applications, LILLE, 114, pp., 441-447, Mars 1983

[INFOR 84] "INFORMIX : Système de gestion de bases de données relationnel"
Manuel utilisateur, 1984

[YAZ 84] K. YAZDANIAN
"Déduction dans les bases de données relationnelles, fondements logiques et mise en oeuvre"
Proc. Séminaire Bases de Données, Sécurité et Intelligence"
C.N.A.M., Paris 1984

[DEV 84] Ph. DEVIENNE
"PROLOG comme langage du SGBD de type réseau IDS II"
Actes des Journées Bases de données, sécurité et Intelligence, AFCET-C.N.A.M., Paris, Octobre 1984

[COND 86] M. CONDILLAC
"PROLOG, Fondements et Applications"
DUNOD, Mars 1986.

B I B L I O G R A P H I E

[DEL 82] C. DELOBEL, M. ADIBA
"Bases de Données et Systèmes Relationnels"
DUNOD, Novembre 1982

[NIC 83] J.M. NICOLAS, K, YAZDANIAN
"Un aperçu de Bdgen : Un SGBD déductif"
Conférence IFIP, 1983

[MARC 83] A. MARCOUX, C. PERCEBOIS, P. BAZEX
"Querylog : Un système de bases de données qui utilise la déduction"
Proc. Journées BIGRE, CAP D'ADGE, 1983

[YAZ 84] Y. YAZDANIAN
"Déduction dans les bases de données relationnelles, fondements logiques et mise en oeuvre"
Proc. Séminaire Bases de Données, Sécurité et Intelligence
C.N.A.M., PARIS 1984

[DEV 84] Ph. DEVIENNE
"PROLOG comme langage du SGBD de type réseau IDS II"
Actes des Journées Bases de Données, Sécurité et Intelligence. AFCET-C.N.A.M., PARIS, Octobre 1984

[FARR 85] H. FARRENY
"Les Systèmes Experts. Principes et Exemples."
CEPADUES, 1985

S O M M A I R E

pages