Facultés universitaires
Notre-Dame de la Paix

INSTITUT D'INFORMATIQUE

Namur

SYSTÈME DE CONCEPTION ET D'EXPLOITATION DE BASES DE DONNEES

DEUXIÈME PARTIE : MANUEL DE REFERENCE DES LANGAGES

PROJET DE RECHERCHE C.I.P.S. N° I.2/15.

Responsable du projet : F. BODART, professeur à l'Institut d'Informatique

Chercheurs de l'équipe "Grands Fichiers" : S. COLUSSI (1976-1977)
C. DEHENEFFE (1971-1977)
D. GUILLEBAUD (1977)
J.L. HAINAUT (1971-1977)
H. HENNEBERT (1971-1976)
B. LECHARLIER (1973-1977)
W. PAULUS (1973-1977)

Collaborateurs : H. LEROY, professeur à l'Institut d'Informatique
P. VAN BASTELAER, professeur à l'Institut d'Informatique
J. BERLEUR, professeur à l'Institut d'Informatique
C. CHERTON, professeur à l'Institut d'Informatique

Avec la participation de J.P. LALOUX
Y. DELVAUX

AVERTISSEMENT

Cette brochure décrit avec précision la syntaxe, les fonctions et les règles d'utilisation des langages de bases de données DDL, DML et NUL. La lecture de la première brochure intitulée "Modèles et langages" constitue un préalable nécessaire à son utilisation.
D'autre part, la mise en oeuvre des processeurs de ces langages est décrite dans la brochure "Architecture et utilisation du système SPHINX".

Dans la mesure du possible, les exemples des différentes expressions des langages concernent la base de données CONFIGUR décrite dans la brochure "Exemples de bases de données et d'application".

°

° °

D.D.L.

1. INTRODUCTION DDL.2.
2. SYMBOLES ATOMIQUES DDL.3.
3. CONVENTIONS UTILISEES POUR DEFINIR LA SYNTAXE DDL.5.
4. DECLARATION DE LA RACINE DDL.9.
5. DECLARATION D'UN DOMAINE DE VALEURS SIMPLE DDL.10.
6. DECLARATION D'UN OBJET ELEMENTAIRE ATTRIBUT DDL.13.
7. DECLARATION D'UN OBJET ELEMENTAIRE DIRECTEMENT RELIE A UN OBJET COMPLEXE DDL.15.
8. DECLARATION D'UN OBJET COMPLEXE ET DES OBJETS ELEMENTAIRES DESCENDANTS DDL.18.
9. DECLARATION D'UNE RELATION ENTRE OBJETS COMPLEXES DDL.19.
10. PROGRAMME DDL DDL.21.
11. REMARQUES SUR L'INTRODUCTION D'UN PROGRAMME DDL EN MACHINE DDL.22.
12. MESSAGES DU COMPILATEURS DDL.23.

D.M.L.

NOTION GENERALES

1. LE PROGRAMME DML DML.2.
2. NOTION DE BLOC OU DE CONTEXTE DML.2.
3. NOTION DE BOUCLE D'ACCES DML.2.
4. LES AUTRES BLOCS DML.3.
5. ROLES DE L'ETIQUETTE DML DML.3.
6. NOTION DE BLOC DE BASE DML.3.
7. LA ZONE DE COMMUNICATION DML.3.
8. NOTION DE VARIABLE DE TRAVAIL DML.4.
9. LES COMPTEURS ASSOCIES A UNE REALISATION D'OBJET COMPLEXE DML.4.
10. LE TRAITEMENT DES ERREURS DML.5.

REGLES D'INTERFERENCES DML/COBOL

1. STRUCTURE D'UN PROGRAMME DML DML.8.
2. ORDRES PERFORM ET IF DML.8.

LIMITATIONS DUES AU COMPILATEUR DML

1. IDENTIFICATEURS RESERVES DML.9.
2. REPERAGE LEXICAL DES ORDRES DML DML.9.
3. LIMITES DE LA STRUCTURE D'UN PROGRAMME DML.9.
4. UTILISATION DE SOUS-PROGRAMMES DML.10.
5. UNICITE DES IDENTIFICATEURS DML.10.
6. NOMBRE D'ORDRES ET DE BLOCS DML DML.10.
7. PARAGRAPHE D'ERREUR GRAVE DML.10.

DESCRIPTION SYNTAXIQUE DES ORDRES DML

1. LES ELEMENTS DU LANGAGE DML.11.
2. LES METASYMBOLES DML.12.
3. LES EXPRESSIONS SYNTAXIQUES DML.13.

LES ORDRES DML

DECLARATION D'UNE ETIQUETTE (@ALLOCATE) DML.15.

- CREATION D'OCCURRENCES DE RELATIONS (@ATTACH) DML.17.
- VIDAGE DE VARIABLES DE TRAVAIL (@CLEAR) DML.19.
- CLOTURE DU CONTEXTE D'UNE BASE DE DONNEES (@CLOSE) DML.20.
- CREATION D'UNE OCCURRENCE D'OBJET COMPLEXE (@CREATE) DML.22.
- SUPPRESSION D'OCCURRENCES DE RELATIONS (@DETACH) DML.25.
- CLOTURE DE CONTEXTE ET ITERATION (@END) DML.27.
- SORTIE PREMATUREE D'UN CONTEXTE (@EXIT) DML.29.
- MODIFICATION DES RELATIONS RELATIVES AUX OBJETS ELEMENTAIRES (@MODIFY) DML.30.
- COPIE ENTRE VARIABLES DE TRAVAIL (@MOVE) DML.32.
- ITERATION FORCEE D'UNE BOUCLE D'ACCES (@NEXT) DML.33.
- ACCES A UNE BASE DE DONNEES (@OPEN) DML.34.
- INITIALISATION AVANT CREATION (@PREPARE) DML.35.
- ACCES A DES REALISATIONS D'OBJETS COMPLEXES (@REACH) DML.37.
- SAUVETAGE D'UNE REFERENCE DE BOUCLE (@SAVE) DML.42.
- SUPPRESSION D'UNE REALISATION D'UN OBJET COMPLEXE (@SUPPRESS) DML.43.
- TEST ET COMPARAISON D'ETIQUETTES ET VARIABLES (@TEST) DML.45.
- REMPLACEMENT D'OCCURRENCES DE RELATIONS (@TRANSFER) DML.46.

LES MESSAGES DU COMPILATEUR DML

1. LES COMPOSANTS DU MESSAGE DML.48.
2. LES NUMEROS DE VERBES ET DE FONCTIONS DML.48.
3. LES MESSAGES GLOBAUX DML.49.
4. LES MESSAGES GENERAUX DML.49.
5. LES MESSAGES SPECIALISES DML.50.

LES CODES D'ERREUR DU SYSTEME SPHINX

1. LES CODES D'ANOMALIES GRAVES DML.52.
2. LES CODES D'ANOMALIES LEGERES DML.54.
3. ANALYSE DES ERREURS DML.57.

o

o o

N.U.L.

1. INTRODUCTION NUL.2.
2. LE MODE OUVERTURE NUL.4.
3. LE MODE CONTEXTE NUL.5.
   3.1. Définition de contexte NUL.5.
   3.2. Structure d'un contexte NUL.7.
   3.3. Critères simples NUL.8.
   3.4. Critères complexes NUL.10.
   3.5. Qualificatif de critère complexe NUL.12.
4. LE MODE COMMANDE (IMPRESSION) NUO.13.
5. TRAITEMENT DES ERREURS NUL.17.
6. LES COMMANDES DE CONTROLE NUL.19.
7. LES FONCTIONS DE CONTROLE DU SYSTEME NUL.20.
8. SYNTAXE DE NUL DANS LE FORMALISME BNF NUL.21.

o

o o

D.D.L.

LANGAGE DE DESCRIPTION DE DONNEES

1. INTRODUCTION

Le présent chapitre a pour objet de décrire les différents éléments du langage de définition de données (DDL) du système SPHINX. Ce DDL permet de déclarer des schémas de bases de données qui pourront alors être exploitées par les langages de manipulation de données (DML et NUL). Il repose sur les concepts du modèle d'accès tel qu'il a été défini dans la brochure : "Modèles et langages".
La compréhension de ce chapitre nécessite donc la lecture préalable des deux premiers chapitres de cette brochure.

On donnera ici une description complète du DDL implémenté dans le système SPHINX. Les restrictions imposées par cette implémentation sont signalées dans la mesure du possible.

Le chapitre commence par la définition des symboles atomiques, constituants élémentaires des phrases du langage et par la description de quelques conventions de description syntaxique.

Sont décrits ensuite les divers types de déclarations et la notion de programme DDL.

Enfin, après quelques remarques sur l'introduction d'un programme en machine, on trouvera, en fin de chapitre, la liste des messages du compilateur DDL.

Les procédures exactes permettant l'introduction en machine d'un programme DDL, sa compilation et l'initialisation de la base de données correspondante sont décrites dans la brochure : "Système SPHINX".

o

o o

2. SYMBOLES ATOMIQUES

Un symbole atomique est formé d'un caractère ou d'une chaîne de caractères considérée comme une entité indécomposable. On peut répartir les symboles atomiques en trois catégories principales : les mots, les entiers et les caractères spéciaux.

2.1. Les mots

Un *mot* est une chaîne finie de caractères alphanumériques ou de "-". Les mots se répartissent en deux catégories principales: les mots-clés et les noms d'objets ou de relations.

Les *mots-clés* constituent un ensemble de mots bien déterminés permettant de distinguer les différents types de déclarations.

Les *noms d'objets et de relations* ont une longueur limitée à 15 caractères (9 pour le nom de racine) et doivent respecter les règles relatives aux noms de données COBOL. En particulier, ils doivent différer des mots réservés COBOL.

Exemples :

mots-clés : ROOT NAME COMPLEX-OBJECT RELATION
noms : A PERSONNE NOM CONJOINT

2.2. Les entiers

Un *entier* est une *chaîne* de 1 à 5 chiffres. Nous appellerons de plus *valeur d'un entier*, le *nombre* entier dont il est la représentation décimale.

Exemple :

Les entiers 1, 01, 001, 0001, 00001 (chaînes numériques) ont pour valeur commune le nombre 1.

Les caractères spéciaux :

Les caractères spéciaux suivants sont utilisés par le DDL :

".", ",", ":", ";", "(", ")"	sont des délimiteurs,
"*"	représente la valeur ∞,
"#"	précède une ligne de commentaires.

o

o o

3. CONVENTIONS UTILISEES POUR DEFINIR LA SYNTAXE

Tout programme DDL se décompose en un certain nombre de *clauses*. Une clause est une suite "significative" de symboles atomiques permettant de déclarer un objet, une relation d'accès ou certaines de leurs propriétés. Elle se compose de mots-clés et de caractères spéciaux, bien déterminés, permettant de reconnaître de quel type de clause il s'agit et, aussi, de symboles variables (noms, entiers, "*") distingant une clause particulière des autres du même type. Une clause se décompose souvent elle-meme en d'autres clauses plus simples.

Exemples de clauses :

a. ROOT NAME IS CONFIGUR.

Dans cet exemple "ROOT", "NAME", "IS" sont des mots-clés propres au type de clause (déclaration de racine). "." est un caractère spécial marquant la fin de la clause. "CONFIGUR" est un nom permettant de distinguer cette déclaration de racine des déclarations antérieures.

b. COMPLEX-OBJECT NAME IS VOITURE ;
ELEMENTARY-OBJECT NAME IS :
1 IMMATRICULATION PICTURE X(6).

Cette clause permet de déclarer l'objet complexe VOITURE et son unique descendant. Elle contient la clause

1 IMMATRICULATION PICTURE X(6)

de déclaration de l'objet élémentaire IMMATRICULATION.
Cette dernière clause contient elle-même la clause

PICTURE X(6)

permettant de déclarer le domaine de valeurs de cet objet.

Nous désignerons un type de clauses par un nom mis entre "⟨" et "⟩" (métasymbole). Exemple : ⟨déclaration-racine⟩.
Par la suite, pour alléger l'exposé, nous utiliserons le terme "clause" au lieu de "type de clauses", lorsqu'aucune confusion ne sera à craindre.

Nous fixerons la syntaxe de chaque clause par une suite de mots-clés; de caractères spéciaux et de métasymboles appelée "règle syntaxique". Un métasymbole permet de désigner soit une clause, soit un nom, soit un entier.
Les métasymboles ⟨nom⟩, ⟨nom-1⟩, ⟨nom-2⟩, ⟨nom-3⟩, ⟨nom-4⟩ représentent des noms.
Les métasymboles ⟨entier-1⟩, ⟨entier-2⟩, ⟨entier-3⟩, ⟨entier-4⟩ représentent des entiers. Les autres métasymboles, placés dans une règle syntaxique, représentent des valorisations de clauses.

Le texte d'une règle syntaxique doit se lire de gauche à droite et de haut en bas. Dans chaque valorisation de la règle, tout mot-clé, tout symbole spécial doit apparaître obligatoirement là où il est spécifié ; de même, tout métasymbole doit être valorisé.

Toutefois, les symboles "[" et "]" échappent à cette règle et jouent un rôle particulier dans la définition de la syntaxe :
Désignons par règle-1, ..., règle-n une série de n règles syntaxiques.

a. $\begin{bmatrix} \text{règle-1} \\ \vdots \\ \text{règle-n} \end{bmatrix}$ désigne soit la chaîne vide, soit une valorisation d'une des règles règle-1, ..., règle-n.

b. $\begin{bmatrix} \text{règle-1} \\ \vdots \\ \text{règle-n} \end{bmatrix}_i^j$ représente l'une quelconque des règles obtenues par concaténation de k règles choisies parmi règle-1, ..., règle-n ($i \leqslant k \leqslant j$). (chaque règle peut être choisie plusieurs fois)

<u>Remarques</u> :

- Les règles [règle-1] et $[\text{règle-1}]_0^1$ sont équivalentes.
- Les symboles "[" et "]" n'apparaissent donc jamais dans la valorisation d'une règle syntaxique.

<u>Exemples</u> :

Règle	Exemple de valorisation
$\begin{bmatrix} \langle\text{nom}\rangle \\ \langle\text{entier}\rangle \\ \text{A} \end{bmatrix}$ B	B LEGRAIN B 128 B A B

$\begin{bmatrix} \langle \text{nom} \rangle \\ \langle \text{entier} \rangle \\ A \end{bmatrix}_{o}^{1}$ B	B LEGRAIN B 128 B A B
[A [B] C] D	D A C D A B C D
$\begin{bmatrix} A \\ B \end{bmatrix}_{o}^{1}$ D	D A D B D
$\begin{bmatrix} A \\ B \end{bmatrix}_{2}^{2}$	A A A B B A B B
$[A]_{1}^{4}$	A A A A A A A A A A

Enfin, nous utiliserons des métasymboles indicés de manière à définir certains ensembles de clauses liées entre elles par une règle algébrique.

<u>Exemple</u> :

Posons $\langle$ suite-croissante $\rangle_1$ égal à 1
Et définissons $\langle$ suite-croissante $\rangle_\alpha$ par la règle ci-dessous, valable pour $\alpha \geqslant 2$:

$\langle$ suite-croissante $\rangle_{\alpha-1}$ α

On obtient ainsi

$\langle$ suite-croissante $\rangle_2$ égal à 1 2
$\langle$ suite-croissante $\rangle_3$ égal à 1 2 3
⋮
$\langle$ suite-croissante $\rangle_\alpha$ égal à 1 2 3 ... α

Certaines règles, d'ordre purement syntaxique, ne peuvent être exprimées simplement par le formalisme que nous venons de définir. Nous les regrouperons sous le titre "restrictions syntaxiques".

o

o o

4. DECLARATION DE LA RACINE

Syntaxe (clause ‹ déclaration-racine›) :

ROOT NAME IS ‹ nom ›.

Restriction syntaxique :

‹ nom› désigne un nom d'au plus 9 caractères.

Sémantique :

Un nouvel objet racine de nom ‹ nom› est déclaré.
Si la compilation du programme se termine sans erreur, ‹ nom› permettra également de désigner le nouveau schéma et la nouvelle base de données ainsi définis.
De plus, s'il existait précédemment un schéma de même nom, celui-ci sera remplacé par le nouveau schéma. Par contre, la base de données correspondante ne sera pas modifiée tant que la procédure d'initialisation de la nouvelle base de données n'aura pas été exécutée.

Exemple :

ROOT NAME IS CONFIGUR.

Remarques :

Définir un schéma dont le nom existe déjà correspond généralement à "modifier" un ancien schéma. Dans ce cas, il faut prendre garde de recompiler tous les programmes DML travaillant sur cette base de données, même s'ils ne manipulent que des objets et des relations déclarés de la même façon dans les deux "versions" du schéma. Sans cela, le résultat de leur exécution sera vraisemblablement une erreur grave.

o

o o

5. DECLARATION D'UN DOMAINE DE VALEURS SIMPLE

Syntaxe de la clause ‹déclaration-domaine-simple› :

Première forme : chaînes de caractères

PICTURE X[(‹entier-1›)]

Seconde forme : nombres

PICTURE [S] [[9[(‹entier-2›)]] V] 9[(‹entier-3›)] $\begin{bmatrix}\text{BINARY}\\ \text{PACKED}\end{bmatrix}_{o}^{1}$

Restrictions syntaxiques :

- ‹entier-1› est un entier compris entre 1 et 1280.
- Etant donnée une clause de la seconde forme,

  posons x = o si la clause est de la forme [S] [V] 9[(‹entier-3›)]
  = 1 si elle est de la forme [S] 9V9[(‹entier-3›)],
  = ‹entier-2› si elle est de la forme [S] 9(‹entier-2›)V9[(‹entier-3›)] ;
  posons y = 1 si la clause est de la forme [S][[9[(‹entier-2›)]] V] 9,
  = ‹entier-3› si elle est de la forme [S][[9[(‹entier-2›)]] V] 9(‹entier-3›);
  on doit avoir

  $1 \leqslant x + y \leqslant 18.$

  En particulier, ‹entier-2› désigne un entier compris entre 1 et 17
  ‹entier-3› désigne un entier compris entre 1 et 18.

Sémantique :

La clause ‹déclaration-domaine-simple› permet de déclarer le domaine de valeurs d'un objet élémentaire simple .

Si la clause est utilisée sous sa première forme, soit l la valeur de ‹entier-1› (si ‹entier-1› n'est pas spécifié, l=1). Le domaine de valeurs ainsi déclaré est l'ensemble des chaînes de caractères EBCDIC de longueur l.

Si la clause est utilisée sous sa seconde forme, elle déclare un ensemble de nombres ; considérons de plus les nombres x et y définis plus haut.

- Si la lettre S est spécifiée, la clause définit un ensemble de nombres positifs ou négatifs ; dans le cas contraire elle définit un ensemble de nombres positifs.

- Si la lettre V est spécifiée, l'ensemble est constitué des nombres décimaux possédant x chiffres avant la virgule et y chiffres après la virgule.

- Si la lettre V n'est pas spécifiée, l'ensemble est constitué des nombres entiers de y chiffres.

- Si le mot-clé BINARY est spécifié, chaque nombre de l'ensemble est codé en mémoire sous forme binaire ; si le mot-clé PACKED est spécifié, il est codé sous forme décimale condensée (2 chiffres par byte) ; sinon, il est codé sous forme décimale étendue (1 chiffre par byte).

Nous appellerons *longueur externe* ou simplement longueur d'une chaîne de caractères le nombre l de caractères de cette chaîne et nous appellerons longueur l d'un nombre le nombre x+y de chiffres de ce nombre. Toute clause ⟨déclaration-domaine-simple⟩ définit un ensemble de valeurs de même longueur. On aura toujours $1 \leqslant l \leqslant 1280$ pour les chaînes de caractères,

$1 \leqslant l \leqslant 18$ pour les nombres.

Nous appellerons *longueur interne* d'une valeur le nombre l^i de bytes utilisés pour représenter cette valeur en mémoire. Ce nombre intervient dans certaines formules limitant la longueur des objets élémentaires. (cfr. déclaration d'un objet élémentaire directement relié à l'objet complexe).

On a : $l^i = l$ pour les chaînes de caractères ;

$l^i = l$ pour les nombres dont la déclaration ne comprend aucun des mots-clés "BINARY" et "PACKED" ;

$l^i = E(\frac{l+2}{2})$ pour les nombres dont la déclaration contient le mot-clé "PACKED" ; (*)

$l^i = 2$ si $1 \leqslant l \leqslant 4$,

4 si $5 \leqslant l \leqslant 9$,

8 si $10 \leqslant l \leqslant 18$ pour les nombres dont la déclaration contient le mot-clé "BINARY".

(*) La lettre E désigne ici la fonction "partie entière" d'un nombre décimal. On a par exemple : $E(\frac{3}{2}) = 1$, $E(\frac{4}{2}) = 2$, $E(\frac{5}{2}) = 2$, etc.

Exemples :

Chaînes de caractères :

PICTURE	X	$l = 1$	$l^i = 1$
PICTURE	X(25)	$l = 25$	$l^i = 25$

Nombres :

PICTURE	S9(8)V9(2)	PACKED	$l = 10$	$l^i = 6$
PICTURE	S9(8)V9	BINARY	$l = 9$	$l^i = 4$
PICTURE	S9V9(2)	PACKED	$l = 3$	$l^i = 2$
PICTURE	S9V9		$l = 2$	$l^i = 2$
PICTURE	SV9(2)		$l = 2$	$l^i = 2$
PICTURE	SV9		$l = 1$	$l^i = 1$
PICTURE	S9(2)		$l = 2$	$l^i = 2$
PICTURE	S9		$l = 1$	$l^i = 1$
PICTURE	9(8)V9(2)	PACKED	$l = 10$	$l^i = 6$
PICTURE	9(8)V9	BINARY	$l = 9$	$l^i = 4$
PICTURE	9V9(2)	PACKED	$l = 3$	$l^i = 2$
PICTURE	9V9		$l = 2$	$l^i = 2$
PICTURE	V9(2)		$l = 2$	$l^i = 2$
PICTURE	V9		$l = 1$	$l^i = 1$
PICTURE	9(2)		$l = 2$	$l^i = 2$
PICTURE	9		$l = 1$	$l^i = 1$

o

o o

6. DECLARATION D'UN OBJET ELEMENTAIRE ATTRIBUT (ou objet élémentaire de niveau supérieur à 1)

Syntaxe des clauses ⟨déclaration-attribut⟩$_\alpha$. (α=2,...,47) :

⟨entier-1⟩ ⟨nom⟩ [OCCURS ⟨entier-2⟩]

$$\left[\begin{array}{l}\langle\text{déclaration-domaine-simple}\rangle \\ \quad [\,,\langle\text{déclaration-attribut}\rangle_{\alpha+1}]^{1280}_{1}\end{array}\right]^{1}_{1}$$

Restrictions syntaxiques :

- ⟨nom⟩ désigne un nom d'objet élémentaire, de niveau α, attribut d'un objet élémentaire de niveau α-1. Ce nom doit respecter les règles relatives aux listes de désignation d'objets élémentaires. (cfr. brochure "Modèles et langages" : page DDL.2.) ;

- ⟨entier-1⟩ désigne un entier de 1 ou 2 chiffres, de valeur α, appelé numéro de niveau. (Il est permis d'écrire 02, 03, ..., 09 au lieu de 2, 3, ..., 9.) Le numéro de niveau d'un objet élémentaire attribut doit donc être compris entre 2 et 47. (cfr. syntaxe) ;

- ⟨entier-2⟩ désigne un entier compris entre 1 et 1280 ;

- une déclaration d'objet élémentaire de niveau 47 ne peut contenir de déclaration d'objet élémentaire attribut. Un tel objet élémentaire est donc toujours simple.

Sémantique :

La clause ⟨déclaration-attribut⟩ permet de déclarer un objet élémentaire attribut de nom ⟨nom⟩ et de niveau α (α=2,...,47) Nous dirons qu'un objet élémentaire est de niveau α s'il est attribut d'un objet élémentaire de niveau α-1. Le niveau des objets élémentaires directement reliés à l'objet complexe est fixé à 1. La déclaration d'un objet élémentaire attribut de niveau α est donc toujours contenue dans la déclaration d'un objet élémentaire composé de niveau α-1. La clause "OCCURS ⟨entier-2⟩"fixe le nombre de valeurs de l'objet attribut contenues dans chaque valeur de l'objet composé. Ce nombre est appelé *facteur de répétition* de l'objet attribut. Si la clause "OCCURS ⟨entier-2⟩" est omise, le facteur de répétition vaut 1.

Si une déclaration d'objet élémentaire attribut contient une clause ‹déclaration-domaine-simple›, c'est un objet simple et son domaine de valeurs est fixé par cette clause. La longueur interne l^i de ces valeurs est aussi appelée *longueur interne de l'objet élémentaire.*

Si une déclaration d'objet élémentaire de niveau α contient une ou plusieurs clauses ‹déclaration-attribut›$_{\alpha+1}$, c'est un objet élémentaire composé. Soient $OE_1, \ldots, OE_n$ les objets élémentaires déclarés par ces clauses, $J_1, \ldots, J_n$ leurs facteurs de répétition et $l_1^i, \ldots, l_n^i$ leurs longueurs internes.
Toute valeur de l'objet de niveau α est formée de la concaténation de J_1 valeurs de OE_1, J_2 valeurs de OE_2, ..., J_n valeurs de OE_n, non nécessairement distinctes, prises dans cet ordre. La longueur interne de l'objet est donnée par l'égalité :

$$l^i = \sum_{k=1}^{n} l_k^i * J_k.$$

<u>Exemples</u> :

a.
```
2   PRIX   PICTURE 9(8)
```
$l^i = 8$

b.
```
3   ADRESSE,
    4   NUMERO    PICTURE 9(4),
    4   RUE       PICTURE X(18),
    4   LOCALITE  PICTURE X(18)
```
$l^i = 40$

c.
```
36  A,
    37   B   OCCURS 2 PICTURE 9(9) BINARY,
    37   C   ,
         38  D   OCCURS 3 PICTURE X(24),
         38  E   OCCURS 2,
             39  F   PICTURE 9(1) PACKED,
             39  G   OCCURS 2,
                 40  H   PICTURE 9(4) BINARY
```
$l^i = 90$

o

o o

7. DECLARATION D'UN OBJET ELEMENTAIRE DIRECTEMENT RELIE A UN OBJET COMPLEXE (ou objet élémentaire de niveau 1)

Syntaxe de la clause <déclaration-objet-élémentaire-niv1> :

Première forme : objet élémentaire simple

[o]1 ⟨ nom ⟩ [OCCURS (⟨ entier-1⟩ , ⟨ entier-2⟩)]

⟨ déclaration-domaine-simple⟩ $\begin{bmatrix}\text{IDENTIFIER}\\ \text{ACCESS-KEY}\end{bmatrix}_{o}^{1}$

Seconde forme : objet élémentaire composé

[o]1 ⟨ nom ⟩ [OCCURS (⟨ entier-1⟩ , ⟨ entier-2⟩)]

$\begin{bmatrix}\text{IDENTIFIER}\\ \text{ACCESS-KEY}\end{bmatrix}_{o}^{1}$ [, ⟨ declaration-attribut⟩$_2$] 1280 / 1

Restrictions syntaxiques :

- ⟨ nom ⟩ est un nom qui doit être différent pour chaque déclaration d'objet élémentaire de niveau 1 contenue dans une même déclaration d'objet complexe;
- ⟨ entier-1 ⟩ est l'un des nombres o ou 1. Toutes les déclarations d'objets élémentaires de niveau 1 pour lesquelles ⟨ entier-1 ⟩ = o doivent être placées après celles pour lesquelles ⟨ entier-1 ⟩ = 1 ou pour lesquelles la clause "OCCURS" est omise;
- ⟨ entier-2 ⟩ est un entier compris entre 1 et 1280 (inclus);
- si le mot-clé IDENTIFIER est spécifié, on doit avoir ⟨ entier-1 ⟩ = ⟨ entier-2⟩ = 1 ou la clause "OCCURS" doit être omise. Ce mot-clé ne peut apparaître que dans la première clause ⟨déclaration-objet-élémentaire-niv 1⟩ incluse dans une même clause ⟨ déclaration-objet-complexe ⟩ ;
- si le mot-clé ACCESS-KEY est spécifié, on doit avoir ⟨ entier-2 ⟩ = 1 ou la clause "OCCURS" doit être omise.

Sémantique :

La clause ⟨déclaration-objet-élémentaire-niv1⟩ permet de déclarer un objet élémentaire, de nom ⟨nom⟩, cible d'une relation issue de l'objet complexe dont la déclaration englobe la présente clause.

Les caractéristiques I et J de cette relation sont fixées par les nombres ⟨ entier-1 ⟩ et ⟨ entier-2 ⟩ , respectivement. L'absence de clause "OCCURS" équivaut à "OCCURS (1,1)".
La caractéristique L de la relation vaut 1 si le mot-clé IDENTIFIER est spécifié, sinon L=∞. De plus, K=o, dant tous les cas.
Si l'un des mots clés IDENTIFIER ou ACCESS-KEY est spécifié, la relation d'accès inverse existe (de l'objet élémentaire vers l'objet complexe).

La première forme de la clause ⟨déclaration-objet-élémentaire-niv1 ⟩ permet de déclarer le domaine des valeurs d'un objet élémentaire simple par une clause ⟨déclaration-domaine-simple ⟩ . Cette dernière clause définit un ensemble de chaînes de caractères ou de nombres ayant tous même longueur interne l^i. Soit alors J la valeur de ⟨ entier-2⟩, on doit avoir : $l^i * J \leqslant 1280$. Ce nombre doit être réduit à 256 si le mot-clé "ACCESS-KEY" est spécifié, à 27 si "IDENTIFIER" est spécifié.

La seconde forme de la clause ⟨déclaration-objet-élémentaire-niv1⟩ permet de déclarer le domaine des valeurs d'un objet élémentaire composé, grâce à une liste de déclarations d'objets élémentaires de niveau 2 (clauses ⟨déclaration-attribut⟩$_2$). Soient $OE_1,\ldots,OE_n$ ces objets élémentaires et $J_1,\ldots,J_n$ les facteurs de répétitions spécifiés dans la clause "OCCURS" de leur déclaration.
Toute valeur de l'objet élémentaire composé est formée de la concaténation de J_1 valeurs de OE_1,J_2 valeurs de $OE_2,\ldots,J_n$ valeurs de OE_n, non nécessairement distinctes, prises dans cet ordre. De plus, si $l_1^i,\ldots,l_n^i$ sont les longueurs internes de ces valeurs, la condition suivante doit être vérifiëe : $(\sum_{k=1}^{n} l_k^i * J_k) * J \leqslant 1280$ (256 si "ACCESS-KEY" est spécifié, 27 si "IDENTIFIER" est spécifié).
L'ordre d'accès aux valeurs, via une relation, n'est pas déterminé par le DDL. Il sera défini dans le DML.
L'ordre d'accès aux réalisations cibles, via une relation d'objet élémentaire vers objet complexe, n'est pas déterminé.
Si une déclaration d'objet complexe contient une déclaration d'objet élémentaire où apparaît le mot-clé "IDENTIFIER", chaque réalisation de l'objet complexe est identifiée par une valeur de l'objet élémentaire. L'ordre d'accès aux réalisations de l'objet complexe, par la relation d'accès issue de la racine, sera l'ordre croissant de ces valeurs (ordre lexicographique). Dans le cas contraire, l'ordre d'accès via cette relation n'est pas déterminé.

Exemples :

```
-  01    TAILLE-MEM    OCCURS (0,1)    PICTURE  9(5)

-   1    NUMERO   IDENTIFIER,
         2   GAMME  PICTURE   X(10),
         2   MODELE PICTURE   X(10)

-   1    PRENOM    OCCURS  (1,10)    PICTURE   X(25)

-   1    A    ACCESS-KEY,
         2    B   OCCURS  2    PICTURE  9(9)   BINARY,
         2    C,
              3    D   OCCURS   3   PICTURE  X(24),
              3    E   OCCURS   2,
                   4   F   PICTURE  9   PACKED,
                   4   G   OCCURS   2,
                       5   H   PICTURE   9(4)   BINARY

-   1    CHAINE   OCCURS  (0,1280)   PICTURE  X
```

°

° °

8. DECLARATION D'UN OBJET COMPLEXE ET DES OBJETS ELEMENTAIRES DESCENDANTS

Syntaxe de la clause ⟨déclaration-objet-complexe⟩

```
[ C-O          ]1
[COMPLEX-OBJECT]1   NAME IS ⟨nom⟩

[ ; [ E-O          ]1  NAME [ IS  ]1  : ⟨déclaration-objet-élémentaire-niv1⟩
    [ELEMENTARY-OBJECT]1    [S  ARE]1
                                  [,⟨déclaration-objet-élémentaire-niv1⟩]319
                                                                        0
  ] .
```

Restriction syntaxique :

⟨nom⟩ est un nom qui doit être différent pour chaque clause ⟨déclaration-objet-complexe⟩ contenue dans un même programme DDL.

Sémantique :

La clause ⟨déclaration-objet-complexe⟩ permet de déclarer un objet complexe de nom ⟨nom⟩ ainsi que ses éventuels descendants. Si l'objet complexe n'est relié à aucun objet élémentaire, un point suivra immédiatement le nom de l'objet complexe. Sinon, les descendants de l'objet complexe sont déclarés par une ou plusieurs clauses ⟨déclaration-objet-élémentaire-niv1⟩.

Le nombre maximum de clauses ⟨déclaration-objet-élémentaire-niv1⟩ est limité par l'implémentation à 320. Ce maximum doit de plus être réduit du nombre de relations entre objets complexes issues de l'objet et du nombre de relations entre objets complexes, sans inverses, ayant l'objet pour cible.
L'objet complexe ainsi déclaré est cible d'une relation issue de la racine, de quadruplet 0-∞, 1-1.

Exemple :

```
C-O  NAME  IS CPU;
E-O  NAMES ARE :
            1 NUMERO IDENTIFIER,
              2 GAMME        PICTURE  X(10),
              2 MODELE       PICTURE  X(10),
            1 MOT            PICTURE  9(2),
            1 MEMOIRE,
              2 MINI         PICTURE  9(5),
              2 MAXI         PICTURE  9(8),
            1 TRANSFERT-IO   PICTURE  9(5),
            1 NBR-PROC       PICTURE  9(2),
            1 COUT           OCCURS (0,1)
              2 PRIX         PICTURE  9(8),
              2 MEMOIRE-ST   PICTURE  9(5).
```

9. DECLARATION D'UNE RELATION ENTRE OBJETS COMPLEXES

Syntaxe de la clause ⟨déclaration-relation⟩

```
RELATION  [NAME IS ⟨nom-1⟩]1
          [WITHOUT NAME    ]1
          FROM ( ⟨entier-1⟩ ,[⟨entier-2⟩]1 )  ⟨nom-2⟩
                             [    *     ]1
          TO   ( ⟨entier-3⟩ ,[⟨entier-4⟩]1 )  ⟨nom-3⟩
                             [    *     ]1
          [INVERTED   [BY ⟨nom-4⟩]] .
```

Restrictions syntaxiques et conventions :

- ⟨nom-2⟩ et ⟨nom-3⟩ sont des noms d'objets complexes déclarés avant la présente clause.
- ⟨entier-1⟩ et ⟨entier-3⟩ sont des entiers compris entre o et 61 (inclus). Soient K et I leurs valeurs respectives. On doit avoir I=o ou K=o.
- ⟨entier-2⟩ et ⟨entier-4⟩ ,s'ils sont spécifiés, sont des entiers strictement positifs. Posons L = ⟨entier-2⟩ si ⟨entier-2⟩ est spécifié,
  ∞ si * est spécifié après "FROM";
  J = ⟨entier-4⟩ si ⟨entier-4⟩ est spécifié,
  ∞ si * est spécifié après "TO".

Sémantique :

La clause ⟨déclaration-relation⟩ permet de déclarer une relation ayant pour origine l'objet complexe ⟨nom-2⟩ et pour cible l'objet complexe ⟨nom-3⟩. Cette relation est caractérisée par le quadruplet I-J, K-L défini ci-dessus. Si les deux mots-clés "WITHOUT NAME" sont spécifiés, la relation ne possède pas de nom; dans le cas contraire, elle a pour nom ⟨nom-1⟩.

Si le mot-clé "INVERTED" n'est pas spécifié, la relation est déclarée sans inverse. S'il est spécifié et suivi immédiatement de ".", elle possède une inverse sans nom. Enfin, si ce mot clé est suivi de "BY ⟨nom-4⟩", la relation inverse a pour nom ⟨nom-4⟩.

L'ensemble des déclarations de relations entre objets complexes, contenues dans un programme DDL, doit vérifier la règle de non-existence de circuits de relations fortes et la règle d'unicité des désignations. Ces règles ont été énoncées et justifiées dans le fascicule : "Modèles et Langages" (pp. MOD 43 et DDL 2). Rappelons-les brièvement.

1. Non-existence de circuits de relations fortes

Etant donnés n+1 objets complexes $O_o, O_1, \ldots, O_n$
et n+1 relations $R_o, R_1, \ldots, R_n$ tels que

- ou bien O_α est origine de R_α,
  $O_{\alpha+1 (\text{mod } n+1)}$ est cible de R_α,
  R_α possède le quadruplet $I_\alpha - J_\alpha$, $K_\alpha - L_\alpha$;
- ou bien O_α est cible de R_α,
  $O_{\alpha+1 (\text{mod } n+1)}$ est origine de R_α,
  R_α possède le quadruplet $K_\alpha - L_\alpha$, $I_\alpha - J_\alpha$; $(\alpha = o, \ldots, n)$

il existe β, γ tels que $0 \leqslant \beta, \gamma \leqslant n$ et $I_\beta = K_\gamma = 0$.

2. Unicité des désignations

Etant donnés deux objets complexes non nécessairement distincts, il existe, au plus, une relation sans nom ayant pour origine un de ces objets et pour cible l'autre. De plus, deux relations nommées de même origine et de même cible ont des noms distincts.

Une conséquence directe de cette deuxième règle, dans le cas des relations d'un objet vers lui-même (⟨nom-2⟩ = ⟨nom-3⟩), est que deux relations inverses ne peuvent pas posséder le même nom (⟨nom-1⟩ = ⟨nom-4⟩) ni être simultanément sans nom (présence de "WITHOUT NAME" et "INVERTED.").

Cette interdiction doit cependant être levée, dans le cas important où I = K = 0 et J = L car alors, la clause ⟨déclaration-relation⟩ permet de déclarer *une seule* relation, *symétrique*.

L'ordre d'accès aux réalisations cibles, à partir d'une même origine n'est que partiellement défini : tant qu'aucune occurrence de la relation n'a été supprimée pour cette origine, l'ordre d'accès est l'ordre chronologique de création des occurrences. Ensuite, il n'est plus défini.

10. PROGRAMME DDL

Syntaxe

$$\langle \text{déclaration-racine} \rangle [\langle \text{déclaration-objet-complexe} \rangle]_{1}^{999} [\langle \text{déclaration-relation} \rangle]_{1}^{500} \text{END.}$$

Sémantique :

Un programme DDL définit un nouveau schéma de base de données. Il comprend une déclaration de racine (clause ‹déclaration-racine›), une ou plusieurs déclarations d'objets complexes (clause ‹déclaration-objet-complexe›) et zéro, une ou plusieurs déclarations de relations entre objets complexes (clause ‹déclaration-relation›).

Le nombre maximum de déclarations d'objets complexes et de relations est limité par l'implémentation, mais la limite exacte est difficile à définir, car elle dépend de beaucoup de facteurs différents. Elle est de l'ordre de quelques centaines.

Remarque :

La compilation d'un programme DDL produit un certain nombre de fichiers permettant l'initialisation de la base de données correspondant au schéma.
Les procédures d'initialisation et de compilation sont exposées dans le chapitre "utilisation du système SPHINX" du fascicule consacré au système SPHINX.

11. REMARQUES SUR L'INTRODUCTION D'UN PROGRAMME DDL EN MACHINE (*)

Pour être manipulable par le compilateur DDL, un programme DDL doit être écrit dans un fichier et donc découpé en enregistrements.
Cette découpe n'est pas significative pour le compilateur : le texte DDL est considéré par lui comme une suite continue de symboles atomiques, éventuellement séparés par un nombre quelconque de blancs (ou espaces).

Précisons toutefois que :

- un symbole atomique ne peut s'étendre sur plusieurs enregistrements;
- si un enregistrement commence par le caractère spécial "#", son contenu n'est pas analysé par le compilateur qui se contentera de le recopier sur le listing de sortie (commentaire);
- les mots et les entiers doivent être séparés, entre eux, au moins par un blanc. Les caractères spéciaux peuvent être contigus à n'importe quel symbole.

Actuellement, le compilateur DDL traite des enregistrements de longueur variable inférieure ou égale à 120 caractères.

De plus, il reconnaît les mots-clés en se basant sur les trois premiers caractères. Autrement dit, tout mot-clé peut être remplacé par ses trois premiers caractères.

(*) On trouvera tous les autres renseignements techniques dans le fascicule : "Utilisation du système SPHINX".

12. LES MESSAGES DU COMPILATEUR

On doit distinguer deux catégories de messages du compilateur :

- les messages de contrôle,
- les messages d'erreur.

12.1. Les messages de contrôle

Ces messages sont affichés au terminal si la compilation se déroule en conversationnel; sinon, ils apparaîtront sur le listing associé à la tâche batch. Ils jalonnent l'exécution et donnent quelques renseignements sommaires sur la validité du programme.

Leur liste et leur signification sont données par le tableau ci-dessous.

Libellé du message	Signification
***** D.D.L. COMPILATION INITIATED *****	Ce message marque le début de la compilation.
***** D.D.L. COMPILATION ABORTED *****	Le programme DDL contient une erreur grave empêchant la détermination du nom de racine.La compilation se termine sans analyser le reste du texte DDL.
***** D.D.L. COMPILATION COMPLETED WITH ERRORS *****	Le texte du programme DDL a été analysé entièrement par le compilateur, mais certaines erreurs ont été détectées. Un fichier d'erreur est généré. L'état des schémas et bases de données connus du système avant la compilation demeure inchangé.

Libellé du message	Signification
***** D.D.L. COMPILATION COMPLETED WITHOUT ERRORS *****	Le compilateur a reconnu un programme DDL valide définissant un nouveau schéma. Les fichiers nécessaires à l'initialisation de la nouvelle base de données sont générés. S'il existait précédemment un schéma de même nom, celui-ci est supprimé et remplacé par le nouveau. Dès la réception de ce message, il est permis de *compiler* des programmes DML travaillant sur le nouveau schéma. Il ne pourront *s'exécuter* qu'après initialisation de la base de données. Il en est de même pour NUL.

12.2. Les messages d'erreur

Ces messages sont placés dans un fichier et renseignent le programmeur sur les erreurs détectées par le compilateur dans son programme. Elles sont précédées, dans le fichier d'erreurs, du numéro de la ligne erronée. Ces erreurs peuvent être de plusieurs types : syntaxiques, déclarations invalides ou incompatibles au sens du modèle d'accès implémenté, dépassement de capacité du compilateur ou du système ...

En voici la liste, classée par types de déclaration.

12.2.1. Erreurs associées au programme DDL

Libellé du message	Signification
END MISSING IS SUPPLIED BY THE SYSTEM THE PROCESS CAN CONTINUE IF ONLY THIS ERROR EXISTS.(1)	Fin incorrecte du programme. La fin de fichier a été détectée prématurément (1) ou le "END" de fin de programme est suivi d'autres symboles (2) ou d'autres enregistrements dans le fichier (3). Si aucune autre erreur n'est détectée, précédant celle-ci, dans le programme DDL, la compilation sera considérée comme correcte.
SYMBOLS ARE PRESENT AFTER "END" OR "END." THE PROCESS MAY CONTINUE IF ONLY THIS ERROR EXISTS. (2)	
THIS LINE IS IGNORED BECAUSE IT COMES AFTER THE END. (3)	
THE SYMBOL "⟨symbole⟩" IS WRONG .	La chaîne de caractères ⟨symbole⟩ ne constitue pas un symbole reconnu par le DDL ou occupe une position incorrecte dans le programme.
THE INTEGER HAS MORE THAN 5 DIGITS	Un entier ne peut comporter plus de 5 chiffres (Cfr. page DDL.3.)

Erreur dans la déclaration de l'objet racine

Libellé du message	Signification
ALL THE DB NUMBERS WERE ASSIGNED. THE DDL TEXT CANNOT BE COMPILED.	32 schémas différents sont déjà connus du système. Ce nombre ne peut être dépassé. Le présent programme ne peut donc être compilé.
ERROR IN THE ROOT DEFINITION. D.D.L. COMPILATION ABORTED.	La clause définissant l'objet racine contient une erreur. La compilation s'arrête prématurément : la suite du texte DDL n'est pas analysée. L'état du système (bases de données, schémas internes, fichiers d'initialisation) demeure inchangé.
THE FIRST CHARACTER OF THE ROOT NAME IS NUMERIC BUT IT MUST BE ALPHABETIC. D.D.L. COMPILATION ABORTED.	

12.2.2. Erreurs dans les déclarations d'objets complexes

Libellé de l'erreur	Signification
IF NO ERRORS EXIST IN THE FIRST C-O DEFINITION, YOU HAVE INTRODUCED A DDL WITHOUT C-O.	Le programme DDL ne contient aucune déclaration valide d'objet complexe.
COMPLEX-OBJECT LENGTH GREATER THAN 2040 CHARACTERS.	Le nombre de bytes nécessaire à l'implémentation de chaque réalisation de l'objet complexe est trop élevé. Causes possibles : objets élémentaires associés trop nombreux ou trop longs; l'objet complexe participe à trop de relations entre objets complexes.
MORE THAN 320 POINTERS FOR A C-O	L'objet complexe participe à un trop grand nombre de relations entre objets complexes. Le nombre maximum de relations auxquelles peut participer un objet complexe est limité par la formule ci-dessous : Soient : $N[j=1]$ et $N[j>1]$ le nombre de relations entre objets complexes,issues de l'objet telles que $j=1$ et $j>1$ (resp.), $N[k=1]$ et $N[k>1]$ le nombre de relations, entre objets complexes, sans inverses, dont l'objet est cible, telles que $k=1$ et $k>1$ (respect.). On doit avoir $N[j=1]+N[k=1]+4*(N[j>1]+N[k>1])\leq 320.$

Erreurs dans les déclarations d'objets complexes

Libellé de l'erreur	Signification
MORE THAN 320 COUNTERS FOR A C-O	Le nombre de relations entre l'objet complexe et d'autres objets (complexes ou élémentaires) est trop élevé. Ce nombre est limité par la formule suivante : Soit E le nombre d'objets élémentaires de niveau 1, associé à l'objet complexe et les nombres N[j=1], N[k=1], N[j>1] N[k>1] définis plus haut, on doit avoir $E + N[i=1] + N[k=1] + N[j>1] + N[k>1] \leqslant 320$
THE FIRST CHARACTER OF THE C-O IS NOT ALPHANUMERIC.	Le nom de l'objet complexe est invalide soit parce qu'il ne commence pas par une lettre, soit parce qu'il est long de plus de quinze caractères.
C-O NAME LENGTH GREATER THAN 15 CHARACTERS.	
WHEN THE OBJECT ⟨nom-1⟩ IS ANALYSED, THE OBJECT ⟨nom-2⟩ IS IN A CYCLE.	L'objet complexe ⟨nom-2⟩ participe à un circuit de relations fortes. (cfr. page DDL.20).
WHEN THE OBJECT ⟨nom-1⟩ IS ANALYSED, THERE ARE MORE THAN 320 ELEMENTS IN THE HIERARCHY.	La complexité de la structure en réseau des relations du schéma est telle que le compilateur DDL ne peut en venir à bout.
WHEN THE OBJET ⟨nom-1⟩ IS ANALYSED, THERE ARE MORE THAN 99 LEVELS IN THE STACK.	

Erreurs dans les déclarations d'objets complexes

Libellé de l'erreur	Signification
TWO C-O WITH THE SAME NAME.	Deux objets complexes sont déclarés avec le même nom. (cfr. page DDL.18); voir aussi la brochure "Modèles et Langages" page DDL2 § 2).

12.2.3. Erreurs dans les déclarations d'objets élémentaires

Libellé du message	Signification
IDENTIFIER GREATER THAN 27 CHARACTERS	La longueur d'un objet élémentaire, déclaré avec la clause IDENTIFIER, excède 27 ou est égale à 0 (cfr. page DDL.16).
IDENTIFIER LENGTH EQUATE TO 0	
PICTURE LENGTH OF THE ELEMENTARY-OBJECT GREATER THAN 1280.	La déclaration du domaine d'un objet élémentaire simple, spécifie un ensemble de valeurs dont la longueur excède 1280 (chaînes de caractères) ou 18 (nombres) ou est égale à 0 (cfr. page DDL.10).
PICTURE LENGTH OF AN E-O IS EQUATE TO 0	
NUMERIC PICTURE HAS MORE THAN 18 DIGITS	
PICTURE LENGTH IS NOT NUMERIC	Erreur syntaxique dans la déclaration d'un domaine de valeurs simples (cfr. page DDL.10).
THE ELEMENTARY OBJECT NAME IS GREATER THAN 15 CHARACTERS.	Le nom de l'objet élémentaire est invalide soit parce qu'il est long de plus de quinze caractères, soit parce qu'il ne commence pas par une lettre.
THE FIRST LETTER OF THE ELEMENTARY OBJET NAME IS NOT ALPHABETIC.	
(THE) SECOND PARAMETER OF CLAUSE OCCURS CORRESPONDING TO AN ELEMENTARY OBJECT OF LEVEL 01 IS NOT NUMERIC. (ou MUST BE DIFFERENT OF 0) (ou IS GREATER THAN 1280).	Erreur syntaxique dans la clause OCCURS d'une déclaration d'objet élémentaire de niveau 1. (cfr. page DDL.15).

Erreurs dans les déclarations d'objets élémentaires

Libellé du message	Signification
AN IDENTIFIER IS ASSOCIATED WITH AN ELEMENTARY OBJECT WICH IS NOT THE FIRST DECLARED.	Erreur syntaxique : le mot-clé IDENTIFIER apparaît dans une déclaration d'objet élémentaire qui ne suit pas immédiatement la phrase "E-O NAMES ARE : " (cfr. page DDL.15).
THE DECLARED IDENTIFIER (ACCESS-KEY) IS IN CONFLICT WITH THE CLAUSE OCCURS.	Erreur syntaxique : un des mots-clés IDENTIFIER ou ACCESS-KEY apparaît dans une déclaration d'objet élémentaire en même temps qu'une clause OCCURS différente de OCCURS (1,1) ou OCCURS (0,1) (valable seulement avec ACCESS-KEY) (cfr page DDL.15 et voir aussi la brochure "Modèles et Langages" : page DDL 6 §3).
THE CLAUSE OCCURS IS NOT NUMERIC.	Erreur syntaxique dans la clause OCCURS d'une déclaration d'objet élémentaire attribut (cfr page DDL.15).
A MANDATORY E-O COMES AFTER AN OPTIONAL.	Erreur syntaxique : les déclarations d'objets élémentaires nanties d'une clause OCCURS (0,J) doivent se trouver après toutes les autres déclarations d'objets élémentaires de niveau 1 (cfr. page DDL.15).

Erreurs dans les déclarations d'objets élémentaires

Libellé du message	Signification
NO UNIQUE QUALIFIER LIST FOR ELEM. OBJECTS ⟨OE-name⟩.	Il existe plusieurs objets élémentaires de même nom ⟨OE-name⟩ ne satisfaisant pas aux règles sur les listes de désignation énoncées à la page DDL 4 de la brochure "Modèles et Langages".
THE E-O ⟨OE-name⟩ IS A STRUCTURE BUT A PICTURE IS ASSOCIATED WITH IT.	La déclaration de l'objet élémentaire ⟨OE-name⟩ contient à la fois une clause PICTURE et une liste de déclarations d'objets élémentaires attributs, il est donc déclaré à la fois simple et composé, ce qui est impossible. (cfr pages DDL.13 et DDL.15 voir aussi la brochure "Modèles et Langages" page MOD 24 §c).
LEVEL-NUMBER NOT NUMERIC.	Erreur syntaxique : le numéro de niveau d'un objet élémentaire est invalide : soit parce qu'il est non numérique, soit parce qu'il ne répond pas aux règles définies aux pages DDL.13 et DDL.15.
FIRST E-O OF C-O HAVE NOT HIS LEVEL-NUMBER EQUATE TO 1.	
THE LEVEL NUMBER ⟨LEV-NBR⟩ IS OUT OF SEQUENCE.	

12.3.4. Erreurs dans les déclarations de relations entre objets complexes

Libellé du message	Signification
THE FIRST CHARACTER OF THE RELATION IS NOT ALPHABETIC.	Le nom de la relation est invalide parce qu'il ne commence pas par une lettre.
TWO RELATIONS WITHOUT NAME (ou TWO RELATIONS WITH THE NAME : ⟨ nom-REL ⟩) BETWEEN THE SAME ORIGIN AND TARGET.	Deux relations sans nom (ou de même nom : ⟨ nom-REL ⟩)sont déclarées ayant même origine et cible, ce qui est interdit. (cfr. page DDL.20 et voir aussi la brochure "Modèles et Langages" page DDL.2. §2).
THE RELATION ATTRIBUTE I (ou J ou K ou L) IS NOT NUMERIC.	Le paramètres I-J, K-L de la relation déclarée ne respectent pas une ou plusieurs des règles définies à la page DDL.19 et aux pages MOD.30 et MOD.43 de la brochure "Modèles et Langages".
THE RELATION ATTRIBUTE K > L (OU I > J).	
THE RELATION ATTRIBUTE J(ou L) IS NOT GREATER THAN ZERO.	
THE RELATION ATTRIBUTES I AND K ARE BOTH DIFFERENT OF ZERO.	
COMPLEX-OBJECT MENTIONNED IN THE RELATION DEFINITION IS NOT DEFINED.	Lors de la compilation d'une déclaration de relation, l'un au moins des objets complexes origine ou cible est non déclaré ou mal déclaré.
A RELATION EXISTS WHILE NO C-O ARE FOUND.	

Erreurs dans les déclarations de relations entre objets complexes

Libellé du message	Signification
THE CAPACITY OF RELATION DICTIONARY IS EXCEEDED. THERE ARE MORE THAN 500 RELATIONS.	Le nombre de déclarations de relations entre objets complexes contenues dans le programme DDL excède la capacité du compilateur (>500).

D.M.L.

LANGAGE DE MANIPULATION DE DONNEES POUR PROGRAMMEURS

NOTIONS GENERALES

1. LE PROGRAMME DML

Un programme DML est un programme COBOL dont la division Procédure contient des ordres DML.
Contrairement aux langages de manipulation de bases de données classiques, l'usage du DML impose au programme d'application une structure très stricte ; en effet, ce langage est basé sur la notion de boucle d'accès, dispositif qui contrôle non seulement les accès à la base de données, mais aussi l'ordre d'exécution des instructions COBOL.

2. NOTION DE BLOC OU DE CONTEXTE

Certains ordres DML ne peuvent apparaître dans un programme que suivis chacun d'un ordre de clôture END ou CLOSE, explicite ou implicite ; la séquence des instructions qui s'étend depuis cet ordre DML jusqu'à l'ordre de clôture qui lui correspond s'appellera un *bloc* (ou *contexte*). Un programme contiendra ainsi des blocs OPEN-CLOSE, REACH-END, ALLOCATE-END et PREPARE-END. Deux blocs distincts sont soit complètement disjoints, soit emboîtés ; il ne peuvent donc se chevaucher.
Un contexte est actif à un endroit déterminé si cet endroit se trouve dans ce contexte.

3. NOTION DE BOUCLE D'ACCES

Un bloc OPEN-CLOSE et un bloc REACH-END constituent une *boucle d'accès* à des réalisations d'un objet racine ou complexe ; un bloc OPEN-CLOSE contrôle l'accès à l'unique réalisation d'un objet racine, c'est-à-dire à une base de données ; un bloc REACH-END contrôle l'accès à certaines réalisations d'un objet complexe d'une base de données, ainsi aux valeurs associées d'objets élémentaires.
Les ordres OPEN et REACH qui constituent l'*entête d'une boucle* définissent le nom de l'objet et les modalités d'accès aux réalisations, tandis que les ordres CLOSE et END constituent la *clôture d'une boucle*.
Les instructions comprises entre l'entête et la clôture d'une boucle constituent le *corps de la boucle* ; ces instructions sont exécutées pour chaque réalisation sélectionnée par l'ordre d'entête de la boucle ; une boucle peut donc ne pas être exécutée si aucune réalisation ne peut être sélectionnée.

A chaque boucle d'accès correspond une *étiquette* distincte qui en particulier sera affectée à chaque *réalisation courante* de cette boucle, ce qui permet de la désigner dans les instructions du corps de boucle. Cette étiquette est inconnue en dehors de la boucle qui la définit.

4. LES AUTRES BLOCS

Le DML fait usage de deux types de blocs : le bloc ALLOCATE-END et le bloc PREPARE-END qui, s'il ne contrôlent aucun accès définissent cependant également une *étiquette* qui sera affectée à une éventuelle réalisation d'un objet complexe, courante pour ce bloc et connue dans lui seul.

5. ROLES DE L'ETIQUETTE DML

Toutes les étiquettes DML d'un programme étant distinctes,une étiquette permettra d'identifier un certain nombre de concepts.
On pourra désigner au moyen d'une étiquette DML :
- l'ordre DML OPEN, REACH, ALLOCATE, PREPARE qui la définit,
- le bloc (ou contexte ou boucle) dont cet ordre est l'entête,
- la réalisation courante éventuelle de ce bloc,
- la zone de communication éventuelle associée à ce bloc (voir ci-dessous).

6. NOTION DE BLOC DE BASE

On appellera *bloc*, ou *contexte*, ou *boucle de base* d'une instruction COBOL ou DML d'un programme, le bloc le plus emboîté dans lequel cette instruction est contenue, qu'il soit du type OPEN-CLOSE ou REACH-, ALLOCATE- ou PREPARE-END.
On appellera de même bloc OPEN-CLOSE de base, le bloc OPEN-CLOSE le plus imbriqué contenant l'instruction.

7. LA ZONE DE COMMUNICATION

Le programme d'application interagit avec le système de gestion de la base de données par l'intermédiaire de zones appartenant au programme et connues à la fois de celui-ci et du système de gestion ; la déclaration de ces zones qui sont appelées *zone de communication* est générée par le précompilateur dans le programme COBOL lui-même. La zone de communication contient trois types d'information :

- des variables par lesquelles le système indique la manière dont l'exécution d'un ordre DML s'est achevée et que le programme pourra consulter ;
- des paramètres d'exécution des ordres DML à destination du système et auxquels le programme ne peut avoir accès ;
- des zones dans lesquelles le système rangera les valeurs d'objets élémentaires demandées dans les ordres DML d'accès et accessibles par des ordres COBOL, et dans lesquelles seront rangées par des ordres COBOL des valeurs grâces auxquelles le système pourra modifier la base de données. Chaque ordre définissant une étiquette pourra provoquer la réservation d'une telle zone qui portera le nom de cette étiquette ; toute partie de cette zone qui correspond à un objet élémentaire dont on demande les valeurs ou devant faire l'objet d'une modification, sera désignée par le nom de cet objet élémentaire ; elle en aura également la découpe telle qu'elle aura été déclarée dans le schéma DDL (traduite dans les concepts COBOL).

La désignation des valeurs d'objets élémentaires liées à une réalisation courante se déduit immédiatement de la description ci-dessus puisque cette désignation se ramène à celle de zones COBOL ; elle obéit donc aux règles de désignation COBOL.

8. NOTION DE VARIABLE DE TRAVAIL

Le programmeur DML dispose de 99 variables dites *variables de travail*. Une telle variable peut être vide ; elle peut aussi contenir la référence à une réalisation d'un objet complexe. Dans ce dernier cas, elle possède également un type qui correspond à l'objet complexe de cette réalisation. En perdant sa valeur (par un ordre CLEAR), une variable perd aussi son type.
Le nom d'une variable de travail est constitué du caractère '$' suivi de deux chiffres différents de '00'.

9. LES COMPTEURS ASSOCIES A UNE REALISATION D'OBJET COMPLEXE

La désignation des différents compteurs s'effectuera sous la forme :

@COUNT ⟨nom cible⟩ FROM ⟨étiq⟩ [⟨dés.rel⟩]

où

⟨étiq⟩ est le nom d'une étiquette DML associée à un contexte actif à cet endroit.

⟨nom.cible⟩ est le nom d'un objet complexe ou élémentaire cible d'une relation dont l'origine est l'objet dont ⟨étiq⟩ est une réalisation.

⟨dés. rel⟩ est la désignation de la relation dont on spécifie le compteur. Une telle relation doit être déclarée dans le schéma. Si la relation porte le nom ⟨nom rel⟩ on spécifiera

VIA ⟨nom rel⟩ ;

Si la relation est sans nom, cette clause VIA sera omise.

Cette expression désigne la valeur du compteur associé à la relation ainsi décrite ; elle peut être employée partout où un identificateur de zone COBOL peut apparaître.

10. LE TRAITEMENT DES ERREURS

Toute exécution d'un ordre DML s'accompagne du positionnement d'indicateurs qui permettent d'analyser la façon dont cet ordre a été exécuté.
L'exécution d'un ordre peut se terminer selon trois modes :

- terminaison normale,
- terminaison avec anomalies légère, prévues ou non par le programmeur, mais qui n'entravent pas le déroulement correct du programme,
- terminaison avec anomalie grave empêchant le travail sur la base de données concernée.

Une anomalie est telle que son occurrence n'implique la perte d'intégrité ni du programme d'application, ni du système SPHINX, ni de la base de données concernée ; rentrent dans cette catégorie, une tentative d'accès à une base de données inexistante, la référence à des étiquettes DML non affectées (dans ce cas l'exécution se poursuit après le CLOSE), des tentatives de violation des caractéristiques I-J, K-L d'une relation, ... Cette anomalie peut être ignorée ou prise en compte par le programmeur.
Une anomalie grave résulte de la détection par le système d'une perte d'intégrité dans le programme d'application (zones paramètres détruites, non respect des règles relatives aux blocs, ...), dans le système de gestion de la base de données (système déchargé, erreur du système, saturation mémoire, erreur de communication) ou dans la base de données. On rangera également dans cette catégorie tout refus du système d'accepter un nouvel utilisateur (fin de session).

Lorsqu'une telle anomalie est détectée, les communications du programme avec SPHINX sont automatiquement clôturées ; ce programme disparaît des tables du système SPHINX ; sa séquence normale est rompue et son exécution se poursuit à un paragraphe de nom SYS-DUMP qui doit être rédigé par le programmeur.
Différentes variables, déclarées automatiquement par le système dans la zone de communication, renseignent le programmeur sur l'exécution d'un ordre DML:

```
SYS-ERROR       (en X(2))
SYS-STATUS      (en X(2))
SYS-MAC-NUM     (en X(4))
SYS-ERAREA
```

SYS-ERROR contient un code de retour qui est '00' en cas de terminaison normale, de '01' à '10' en cas d'anomalie légère et de '11' à '99' ou un code alphanumérique en cas d'anomalie grave.

SYS-STATUS indique au moyen d'un code le verbe DML ou l'opération dont l'exécution a provoqué l'anomalie, ce code est établi comme suit :

01	OPEN
02	CLOSE
03	REACH
04	EXIT
05	NEXT
06	END
07	ALLOCATE
08	PREPARE
09	CREATE
10	MODIFY
11	ATTACH
12	DETACH
13	TRANSFER
14	SUPPRESS
15	TEST
16	SAVE
17	MOVE
18	CLEAR
19	COUNT
20	Protection.

SYS-MAC-NUM contient le numéro séquentiel de l'ordre DML dans le programme ; cette numérotation est assurée par le compilateur DML et est indiquée sur le listing de compilation.

SYS-ERAREA : Cette zone fournit le détail des codes de retour dans le cas d'un ordre DML multiple : CREATE, ATTACH, DETACH, TRANSFER ; chaque code est relatif à une opération partielle selon des modalités qui seront précisées pour chacun de ces ordres. SYS-ERROR contient le maximum de ces codes.

SYS-ERAREA se décompose comme suit :

- SYS-ERNBR (en 9(3)COMP) : nombre de codes.
- SYS-ERLIST (en X(2) OCCURS) : liste des codes.

o

o o

REGLES D'INTERFERENCE DML/COBOL

Les principes du DML et de COBOL ainsi que certaines stratégies de compilation de ces deux langages imposent au programmeur le respect de certaines règles de construction des programmes DML.

1. STRUCTURE D'UN PROGRAMME DML

La structure de boucle (et de bloc en particulier) est prioritaire sur la structure de contrôle qui serait imposée par les ordres COBOL GOTO, PERFORM et USE. Du point de vue du programme COBOL, un bloc doit être considéré de l'extérieur comme une instruction unique (un GO TO extérieur à un bloc ne peut renvoyer à l'intérieur de ce bloc) ; d'autre part, un bloc doit être considéré comme une procédure fermée dont on ne sort que par l'ordre END ou CLOSE ou encore par un ordre DML EXIT (un GO TO intérieur à un bloc ne peut renvoyer à l'extérieur de ce bloc).
Un bloc a donc un seul point d'entrée, qui est l'ordre DML d'entête, et un seul point de sortie qui est l'ordre DML de clôture, exécuté directement ou par le biais d'un ordre EXIT. L'usage des ordres PERFORM devra également suivre cette règle.

Cette rigidité imposée peut cependant être assouplie car ces règles ne s'appliquent qu'à l'enchaînement statique (perçu à la compilation DML) et dynamique (perçu par SPHINX) des ordres DML uniquement. Par exemple, il est permis de placer dans un bloc un ordre PERFORM spécifiant des paragraphes extérieurs à ce bloc si ceux-ci ne contiennent aucun ordre DML.

2. ORDRES PERFORM ET IF

Lors de l'utilisation d'un ordre PERFORM relatif à un paragraphe contenant des blocs DML, on tiendra compte du fait que la traduction de ces derniers est susceptible d'engendrer des étiquettes COBOL.

D'autre part, seuls les ordres NEXT, EXIT et COUNT ont une traduction dans laquelle n'apparaît pas le signe de ponctuation "." ; eux seuls peuvent donc apparaître dans une branche (ou une condition) d'une instruction alternative IF-THEN-ELSE.

LIMITATIONS DUES AU COMPILATEUR DML

1. IDENTIFICATEURS RESERVES

Le précompilateur COBOL génère soit en division DATA soit en division PROCEDURE des identificateurs (nom de zones ou d'étiquettes) tous préfixés par SYS- ; le programmeur, prudent, évitera de déclarer des identificateurs de cette forme.

2. REPERAGE LEXICAL DES ORDRES DML

- Toute ligne de la division PROCEDURE dont le premier caractère non blanc est "@" est considérée comme contenant un verbe DML ; ce verbe est le premier mot qui suit ce caractère dans la ligne.

- Un ordre DML (sauf COUNT, NEXT et EXIT) s'étend depuis le caractère "@" jusqu'au premier "." suivant qui ne fait pas partie d'une constante ; cet ordre peut s'étendre sur plusieurs lignes.

- Lorsqu'un ordre DML (sauf COUNT), ou une partie seulement, apparaît sur une ligne, celle-ci ne peut contenir ni un autre ordre DML, ni une partie d'un tel ordre, ni un ordre COBOL.

- L'expression @COUNT peut apparaître partout où un nom de zone peut être utilisé en COBOL.

- Un ordre DML ou une partie de celui-ci doit être inscrit entre les positions 12 et 72 (celles-ci incluses).

- Un mot d'un ordre DML ne peut s'étendre sur plus d'une ligne ; en particulier, une constante alphanumérique citée dans un ordre REACH ne peut excéder 59 caractères.

3. LIMITES DE LA STRUCTURE D'UN PROGRAMME

- Un programme ne peut contenir plus de 99 blocs simultanément emboîtés ;
- Un programme ne peut contenir plus de 25 ordres REACH IF simultanément emboîtés ;
- Un programme et ses sous-programmes ne peuvent ouvrir plus de 26 bases de données.

4. UTILISATION DE SOUS-PROGRAMMES

Tout ordre DML relatif à l'accès ou à la modification de réalisations d'objets complexes doit être inscrit dans un bloc OPEN-CLOSE ; d'autre part, un processus (exécution d'un module assemblé) ne peut demander une seconde fois l'ouverture (OPEN) d'unebase de données s'il ne l'a déjà fermée.
Il en résulte que l'on ne peut appeler, de l'intérieur d'un bloc relatif à une base de données, un sous-programme travaillant sur cette même base de données. L'appel à un tel sous-programme ne se fera qu'après fermeture (CLOSE) de la base de données dans le programme appelant ; d'autre part, le programme appelé devra contenir son propre bloc OPEN-CLOSE.
On notera cependant que dans le système SPHINX l'exécution d'un ordre CLOSE est peu coûteuse, et qu'il en est de même d'un nouvel ordre OPEN à condition que celui-ci ne suive pas de trop loin le dernier CLOSE.

5. UNICITE DES IDENTIFICATEURS

La déclaration d'un étiquette dans un ordre DML OPEN, REACH, ALLOCATE ou PREPARE donne lieu à la génération dans la zone de communication d'une zone dont le nom est celui de l'étiquette et dont la découpe reprend entre autres choses la description des objets élémentaires éventuels demandés dans cet ordre. On en déduit aisément la règle d'unicité des étiquettes DML, celle relative aux mots réservés de COBOL ainsi que celle de la qualification des noms d'objets élémentaires cités dans les instructions COBOL.

6. NOMBRE D'ORDRES ET DE BLOCS DML

Dans la version actuelle, à tout ordre DML (sauf NEXT, EXIT et COUNT) correspond un appel au sous-programme d'interface ; d'autre part, tout bloc (sauf OPEN-CLOSE) entraîne la génération d'une redéfinition de la zone de communication (REDEFINES).

On vérifiera lors de l'exécution d'un gros programme que le compilateur COBOL peut supporter ces nombres correspondant de CALL et de REDEFINES.

7. PARAGRAPHE D'ERREUR GRAVE

Tout programme DML doit contenir un paragraphe d'erreur portant le nom standard SYS-DUMP.

o

o o

DESCRIPTION SYNTAXIQUE DES ORDRES DML

1. LES ELEMENTS DU LANGAGE

1.1. Symboles

@ = < > , ; . : * ( )

1.2. Mots

- *Verbes* : OPEN, CLOSE, REACH, EXIT, NEXT, END, ALLOCATE, PREPARE, CREATE, MODIFY, ATTACH, DETACH, TRANSFER, SUPPRESS, TEST, SAVE, MOVE, CLEAR, COUNT

- *Opérateurs logiques* : AND, OR

- *Opérateur de négation* : NOT

- *Mots fonctionnels* : FROM, WITH, GET, TO, VIA, BY, IN, IF, NEXT, EXIT

- *Valeur standard de paramètre* : ALL

- *Identificateurs* :
  - noms de zones COBOL,
  - noms d'éléments du schéma (relation et objets),
  - noms d'étiquettes DML : obéissent à la règle de formation des identificateurs COBOL, mais ont une longueur de 8 caractères au plus,
  - noms de variables de travail : construit par concaténation du symbole $ et de 2 chiffres différents de '00'.

- *constantes numériques et alphanumériques* : mêmes formats que ceux acceptés par le compilateur COBOL, chaînes de caractères limitées à 59 positions.

1.3. Règle de séparation des mots

Le précompilateur est capable de distinguer parmi deux éléments consécutifs d'une même ligne :
- un symbole d'un mot même s'ils sont contigus ;
- deux symboles, mêmes s'ils sont contigus ;
- deux mots, s'ils sont séparés par au moins un espace.

Hormis leur rôle de séparation, les espaces sont ignorés par le compilateur ; ainsi en est-il également des symboles "," et ";" dans certains ordres DML.

2. LES METASYMBOLES

L'expression de la syntaxe DML fait appel à plusieurs *métasymboles* dont le rôle est de désigner soit des classes d'identificateurs, soit des clauses qui devront être développées dans la suite ; il apparaîtrons entre ⟨ et⟩.

2.1. Classes d'identificateurs

- noms d'éléments du schéma : ⟨ nom rac⟩ mis pour : un nom de racine ;
  ⟨ nom OC⟩ mis pour : un nom d'objet complexe ;
  ⟨ nom OE⟩ mis pour : un nom d'objet élémentaire directement relié à un objet complexe ;
  ⟨ nom rel⟩ mis pour : un nom de relation entre objets complexes.

- noms d'étiquettes DML : ⟨ étiq⟩, ⟨ étiq1⟩, ⟨ étiq2⟩, ⟨ étiq3⟩, ⟨ étiq4⟩ mis pour : un nom d'étiquette.

- nom de variables de travail : ⟨ var.trav⟩, ⟨ var.trav1⟩, ⟨ var.trav2⟩, mis pour : un nom de variable de travail.

2.2. Clauses à développer

- ⟨ liste étiq⟩ mis pour : liste de une ou plusieurs étiquette ; deux étiquettes consécutives sont séparées par "," et un nombre arbitraire d'espaces (0, 1 ou plusieurs).

- ⟨ liste OE⟩ mis pour : liste de un ou plusieurs noms d'objets élémentaires ; les règles de séparation sont les mêmes que ci-dessus.

- ⟨ dés.valeur⟩ mis pour : soit une constante, soit le nom d'une zone COBOL.

- ⟨ liste dés.valeur⟩ mis pour : liste d'une ou plusieurs ⟨ dés.valeur⟩ ; les règles de séparation sont les mêmes que ci-dessus.

- ⟨dés.rel⟩ : mis pour : désignation d'une relation entre deux objets complexes ; son développement sera explicite là où elle sera utilisée.

- ⟨transfert partiel⟩ : mis pour : un transfert partiel ; son développement sera explicité là où il sera utilisé.

- ⟨condition⟩ : mis pour : une condition ; son développement sera explicité là où elle sera utilisée.

- ⟨op.comp1⟩, ⟨op.comp2⟩ : mis pour : un opérateur de comparaison ; leur développement sera explicité là où il sera utilisé.

2.3. Le métasymbole ⟨vide⟩

Ce métasymbole est mis pour : absence d'élément.

3. LES EXPRESSIONS SYNTAXIQUES

Les règles d'interprétation sont celles qui ont été définies dans la section DDL du présent volume.

o

o o

LES ORDRES DML

DECLARATION D'UNE ETIQUETTE (@ALLOCATE)

1. SYNTAXE

@ALLOCATE ⟨ nom OC ⟩ = ⟨ étiq1 ⟩ [FROM ⟨ étiq2 ⟩] [$\begin{bmatrix}\text{WITH}\\ \text{GET}\end{bmatrix}_1^1$ $\begin{bmatrix}\langle\text{ liste OE }\rangle\\ \text{ALL}\end{bmatrix}_1^1$].

2. REGLES SYNTAXIQUES

- ⟨ nom OC ⟩ est le nom d'un objet complexe de la base de données désignée par ⟨ étiq2 ⟩.
- ⟨ étiq1 ⟩ est le nom d'une étiquette de boucle , qui sera attachée à ce contexte ; ce nom ne peut pas avoir déjà été défini.
- ⟨ étiq2 ⟩ est le nom d'une étiquette relative à un bloc OPEN-CLOSE ; si la clause FROM est absente, le bloc OPEN-CLOSE de base est implicitement désigné.
- ⟨ liste OE ⟩ est une liste dont chaque élément est le nom d'un objet élémentaire directement accessible à partir de l'objet complexe de nom ⟨ nom OC ⟩.
- Cet ordre doit se trouver dans le contexte ⟨ étiq2 ⟩ si la clause FROM est spécifiée, ou dans un bloc OPEN-CLOSE sinon .
- A tout ordre @ALLOCATE doit correspondre un ordre @END, explicite ou non, qui clôture son contexte.

3. SEMANTIQUE

L'ordre ALLOCATE crée un contexte qui s'étend jusqu'au END qui le clôturera en aval dans le programme ; à ce contexte est associée une étiquette de nom ⟨ étiq1 ⟩ qui pourra dans la suite être affectée à différentes réalisations de l'objet complexe de nom ⟨ nom OC ⟩. Cette affectation s'effectuera grâce à des ordres REACH IN et PREPARE/CREATE situés dans le contexte de ⟨ étiq1 ⟩. Après l'exécution de l'ALLOCATE, et dans son contexte, l'étiquette ⟨ étiq1 ⟩ est connue et non affectée ; tout ordre (ou boucle) DML lui faisant référence sera correct mais non effectué.
La clause FROM, explicite ou non, indique au compilateur la base de données à laquelle l'objet ⟨ nom OC ⟩ appartient (rappelons qu'il n'y a pas de contrainte d'unicité de noms d'objets complexes entre plusieurs bases de données).

L'ordre ALLOCATE réserve dans la zone de communication de l'espace qui accueillera non seulement des données techniques, mais aussi, si l'utilisateur le désire, des valeurs d'objets élémentaires liées à la réalisation ⟨ étiq1 ⟩.

Le paramètre WITH(GET) permet au programmeur de spécifier les objets élémentaires dont il désire les valeurs ; la valeur standard ALL signifie que tous les objets élémentaires liés à l'objet ⟨nom oe⟩ sont désirés alors que l'absence de la clause WITH(GET) indique qu'aucune valeur n'est demandée.

Avant qu'une réalisation de ⟨nom OC⟩ ne soit affectée à ⟨étiq1⟩, les zones de réception de valeurs d'objets élémentaires contiennent des valeurs aléatoires éventuellement incompatibles avec le type déclaré de l'objet élémentaire ; on se gardera donc de consulter ces zones avant qu'elles ne soient effectivement garnies. On utilisera, en cas d'ambiguïté l'ordre TEST(⟨étiq1⟩) qui permettra de vérifier l'affectation de ⟨étiq1⟩.

4. EXEMPLES

```
@ALLOCATE   MEMOIRE = M.
@ALLOCATE   CPU = UC  FROM  CONF  WITH  (NUMERO,CONT).
```

°

° °

CREATION D'OCCURRENCES DE RELATIONS (@ATTACH)

1. SYNTAXE

@ATTACH [⟨ étiq1 ⟩] [TO ⟨ étiq2⟩ ⟨ dés.rel ⟩]$_1^{100}$

2. REGLES SYNTAXIQUES

- ⟨ étiq1⟩ est le nom d'une étiquette relative à un objet complexe O ; Si cette clause est absente, l'étiquette implicitement désignée est celle du bloc de base ;
- ⟨ étiq2 ⟩ est le nom d'une étiquette relative à un objet complexe C ;
- ⟨ dés.rel⟩ est la désignation d'une relation entre objets complexes ; les formes possibles sont les suivantes :
  - VIA ⟨ nom rel ⟩ : il y correspond la relation de nom ⟨ nom rel ⟩ de O vers C ;
  - VIA ⟨ nom rel ⟩*: il y correspond la relation de nom ⟨ nom rel ⟩ de C vers O ;
  - ⟨ vide ⟩ : il y correspond la relation sans nom de O vers C ;
  - VIA * : il y correspond la relation sans nom de C vers O.

  Il doit exister dans la base de données concernée une relation correspondant à cette désignation.
- Cet ordre doit se trouver dans les contextes de ⟨ étiq1⟩ et ⟨ étiq2⟩ ;
- Toutes les étiquettes citées (explicitement ou non) doivent être relatives à des ordres REACH, ALLOCATE ou PREPARE WITH déjà traité par un ordre CREATE ;
- Chaque relation désignée doit être du type I-J, K-L avec I>J et K>L.

3. SEMANTIQUE

L'ordre ATTACH commande la création d'occurrences de relations selon les modalités suivantes :

- soit r_o la réalisation correspondant à ⟨ étiq1⟩ et r_i la réalisation correspondant à ⟨ étiq2⟩;
- pour chaque clause TO ⟨ étiq2⟩ ⟨ dés.rel ⟩ , on demandera soit la création de l'occurrence (r_o,r_i) de la relation désignée par ⟨ dés.rel ⟩ si cette désignation n'est pas suffixée par "*", soit la création de l'occurrence (r_i,r_o) de la relation désignée par ⟨ dés.rel ⟩ si cette désignation est suffixée par "*" ;

- une création n'est effectuée que si :
  - r_o et r_i existent effectivement,
  - cette création ne viole pas le respect des paramètres J et L de la relation désignée,
  - l'occurrence n'existe pas encore ;
- si la relation désignée possède une inverse, une occurrence de cette dernière est également créée ;
- les créations sont commandées dans l'ordre où leurs clauses apparaissent dans l'ordre ATTACH.

4. EXEMPLES

```
- @ATTACH  MACH1  TO  SGBD1.
- @ATTACH  SYST-EXP  TO  CONSTR1,
                     TO LANGAGE1  VIA  ACCEPTE*,
                     TO LANGAGE2  VIA  SUPPORTE,
                     TO CONSTR2   VIA  *,
                     TO SGBD1     VIA  SUPPORTE,
                     TO SGBD2     VIA  SUPPORTE*.

- @ATTACH  TO  TERMIN1, TO  TERMIN2,  TO  TERMIN3.
```

o

o o

VIDAGE DE VARIABLES DE TRAVAIL (@CLEAR)

1. SYNTAXE

$$@CLEAR \left[\begin{array}{l} \text{ALL} \\ \langle \text{var.trav.1} \rangle \left[,\langle \text{var.trav.2} \rangle \right]_{0}^{98} \end{array} \right]_{1}^{1} .$$

2. REGLES SYNTAXIQUES

⟨var.trav.1⟩ et ⟨var.trav.2⟩ sont des noms de variable de travail.

3. SEMANTIQUE

- Si la clause ALL est spécifiée, toutes les variables de travail sont vidées de leur valeur et de leur type.
- Si la clause ⟨var.trav.1⟩ [,⟨var.trav.2⟩]$_{0}^{98}$ est spécifiée, les variables ainsi désignées sont vidées de leur valeur et de leur type.

4. EXEMPLES

```
@CLEAR  $11.
@CLEAR  $04, $01, $20.
@CLEAR  ALL.
```

o

o o

CLOTURE DU CONTEXTE D'UNE BASE DE DONNEES (@CLOSE)

1. SYNTAXE

@CLOSE [⟨étiq⟩].

2. REGLES SYNTAXIQUES

- ⟨étiq⟩ est le nom d'une étiquette connue jusqu'à cet endroit, et définir dans un ordre OPEN ; il ne correspond qu'un seul ordre CLOSE à tout ordre OPEN.
- si la clause ⟨étiq⟩ est absente, le CLOSE se réfère implicitement à l'ordre OPEN précédent le plus proche qui n'a pas encore été clôturé par un CLOSE à cet endroit.

3. SEMANTIQUE

L'ordre CLOSE définit la fin du contexte ouvert par l'ordre OPEN auquel il correspond. Après son exécution, la base de données disponible dans ce contexte est rendu indisponible.

L'ordre OPEN clôture également tous les contextes qui lui sont emboîtés et qui n'auraient pas encore été fermés à cet endroit ; ces contextes ont été ouverts par des ordres OPEN, REACH, ALLOCATE ou PREPARE. Pour ces contextes, l'ordre CLOSE définit une suite de clôtures END et CLOSE *implicites*.

4. EXEMPLES

- voir exemple de l'ordre OPEN.
- le texte :

```
        @OPEN  CONFIGUR = CO.
    @ALLOCATE  MEMOIRE = M.
               @REACH  CPU = C.
                       @OPEN PERSONNEL = P
                            @PREPARE EMPLOYE = E
@CLOSE CO
```

est équivalent à :

```
@OPEN  CONFIGUR = CO
       @ALLOCATE  MEMOIRE = M
                  @REACH  CPU = C
                          @OPEN  PERSONNEL = P
                                 @PREPARE  EMPLOYE = E

                                 @END
                          @CLOSE
                  @END
       @END
@CLOSE
```

o

o o

CREATION D'UNE OCCURRENCE D'OBJET COMPLEXE (@CREATE)

1. SYNTAXE

$$\text{@CREATE [}\langle\text{étiq1}\rangle\text{] [,}\langle\text{dés.rel}\rangle\text{ : } \left[\begin{array}{c}\langle\text{étiq2}\rangle\\ (\langle\text{liste étiq}\rangle)\end{array}\right]_1^1 \Big]_o^{25}.$$

2. REGLES SYNTAXIQUES

- ⟨étiq1⟩ est le nom d'une étiquette définie dans un ordre PREPARE.
- ⟨étiq2⟩ est le nom d'une étiquette définie dans un ordre ALLOCATE, REACH ou PREPARE déjà traité par un CREATE.
- ⟨liste étiq⟩ est une liste d'une ou plusieurs étiquettes définies par des ordres ALLOCATE, REACH ou PREPARE déjà traité par un CREATE ; ces étiquettes sont relatives à un même objet complexe.
- L'absence de la clause ⟨étiq1⟩ indique que l'ordre spécifie implicitement l'étiquette de son contexte de base, qui doit être celui d'un PREPARE.
- L'étiquette spécifiée explicitement ou non dans l'ordre CREATE doit être celle de son bloc de base ; la clause ⟨étiq1⟩ est donc toujours facultative.
- ⟨dés.rel⟩ est la désignation d'une relation entre objets complexes. Soit O l'objet correspondant au contexte de base, et C l'objet correspondant à ⟨étiq⟩ ou (⟨liste étiq⟩.). Les formes de désignation possibles sont les suivantes :
  - <nom rel> : il y correspond la relation de nom ⟨nom rel⟩ de O vers C ;
  - <nom rel>*: il y correspond la relation de nom ⟨nom rel⟩ de C vers O ;
  - <vide> : il y correspond la relation sans nom de O vers C ;
  - * : il y correspond la relation sans nom de C vers O.

  Il doit exister dans la base de données une relation ainsi désignée.
- Parmi les relations désignées doivent se trouver toutes celles qui ont O pour origine et telles que I > Ø ou leur inverse, mais l'une des deux seulement, ainsi que toutes celles qui ont O pour cible et telles que K > Ø et qui sont sans inverse.
- Pour chaque relation désignée dont O est origine, il faut spécifier au moins I étiquettes distinctes dans la clause ⟨étiq2⟩ ou (⟨liste étiq⟩) ; pour chaque relation désignée dont O est cible, il faut spécifier au moins K étiquettes distinctes dans cette même clause.

- L'étiquette du contexte de base ne peut apparaître dans la clause ‹étiq2› ou (‹liste étiq›).
- On ne peut écrire dans le contexte d'un PREPARE qu'un seul ordre CREATE qui lui est relatif.
- On ne peut trouver entre un PREPARE IN et son CREATE, d'ordre REACH IN spécifiant la même étiquette que ce PREPARE.
- L'ordre CREATE doit se trouver dans tous les contextes ‹étiq2› et (‹liste étiq›).

3. SEMANTIQUE

a. L'exécution de l'ordre CREATE crée une réalisation d'un objet complexe *pourvu que les conditions d'existence de cette réalisation soient réunies.*
Ces conditions sont précisées dans le texte DDL qui décrit la base de données ; elles s'expriment dans les caractéristiques des relations auxquelles participe l'objet dont on désire créer une réalisation.
D'une manière générale, pour toute relation (appelée R) dont l'objet est origine, il faut spécifier au moins I_R et au plus J_R occurrences à créer simultanément, et pour toute relation (appelée S) dont l'objet est cible, il faut spécifier au moins K_S et au plus L_S occurrences à créer simultanément.
Pratiquement ces conditions sont réalisées de la manière suivante :
- pour chaque objet élémentaire directement relié à l'objet complexe, il faut préalablement à l'exécution du CREATE avoir garni la zone de communication avec au moins I et au plus J valeurs, le nombre exact étant assigné au compteur de cet objet élémentaire (I et J sont les paramètres de la clause OCCURS (I,J) explicite ou non) ; on notera que l'initialisation due au PREPARE est considérée comme une valeur valide.
- si l'un des objets élémentaires est déclaré identifiant (IDENT) aucune réalisation de l'objet complexe ne doit déjà être associée à la valeur présente dans la zone de communication pour cet objet élémentaire.
- Pour chaque relation (appelée R) entre objets complexes dont l'objet concerné est origine, l'ordre CREATE doit spécifier au moins I_R et au plus J_R occurrences valides et distinctes à créer simultanément dans l'ordre CREATE.
- Pour chaque relation (appelée S) entre objets complexes dont l'objet concerné est cible, l'ordre CREATE doit spécifier au moins K_S et au plus L_S occurrences valides et distinctes à créer simultanément.
- L'exécution de l'ordre CREATE crée la réalisation définie ci-dessus *pourvu que les caractéristiques des relations dont on crée simultanément des occurrences ne soient pas violées pour les autres réalisations de la base de données* ; cependant, si la création d'une occurrence de relation est reconnue impossible, et si la présence de cette occurrence *n'est pas une condition*

d'existence de la réalisation à créer, cette dernière sera créée malgré l'échec de la création de l'occurrence de relation.

2. Simultanément à la création d'une réalisation d'objet complexe, l'exécution d'un CREATE crée les occurrences de relation qui sont spécifiées dans les clauses de la commande ; sont créées non seulement les occurrences qui constituent les conditions d'existence de la réalisation créée, mais aussi d'autres, pour autant que la règle du point précédent soit respectée.

3. Si les relations apparaissant dans les clauses du CREATE sont munies d'une inverse, les occurrences de ces dernières sont également créées.

4. L'exécution de l'ordre CREATE affecte la réalisation créée à l'étiquette du PREPARE correspondant, ainsi qu'à celle de l'ALLOCATE spécifié dans la clause IN de ce PREPARE.

4. EXEMPLE

4.1. Considérons la création d'une réalisation de SGBD. Celle-ci doit être reliée à au moins une réalisation de CONSTRUCTEUR, et une réalisation de NOM. Le programme suivant est donc valide :

```
@REACH CONSTRUCTEUR = C1 IF (...).
      @REACH CONSTRUCTEUR = C2 FROM CONF IF (...).
            @REACH SYSTEME-EXPLOIT = S1 FROM CONF IF (...).
                  @PREPARE SGBD = DB WITH (NOM, SIGLE).
                           MOVE Z1 TO NOM OF DB.
                           MOVE Z2 TO SIGLE OF DB.
                           MOVE 1 TO @COUNT NOM FROM DB,
                                     @COUNT SIGLE FROM DB.
                           @CREATE DB, :(C1,C2), SUPPORTE : S1
                  @END.
@END C1.
```

4.2.
```
@ CREATE CPU1, FOURNI : CONSTR1, : SYSTEXPL.
@ CREATE CPU2, DISTRIBUE* : CONSTR1, COMPATIBLE : CPU3, CPU1.
@ CREATE SGBD1, :(CONSTR1,CONSTR2), SUPPORTE* : SE1.
@ CREATE MEM, : CONSTR1, * : CPU1.
```

°

° °

SUPPRESSION D'OCCURRENCES DE RELATIONS (@DETACH)

1. SYNTAXE

@DETACH [⟨étiq1⟩] [FROM ⟨étiq2⟩ ⟨dés.rel⟩]$_1^{100}$.

2. REGLES SYNTAXIQUES

- ⟨étiq1⟩ est le nom d'une étiquette relative à un objet complexe O ; si cette clause est absente, l'étiquette implicitement désignée est celle du bloc de base ;
- ⟨étiq2⟩ est le nom d'une étiquette relative à un objet complexe C ;
- ⟨dés.rel⟩ est la désignation d'une relation entre objets complexes ; les formes possibles sont les suivantes :
  - VIA ⟨nom rel⟩ : il y correspond la relation de nom ⟨nom rel⟩ de O vers C ;
  - VIA ⟨nom rel⟩* : il y correspond la relation de nom ⟨nom rel⟩ de C vers O ;
  - ⟨vide⟩ : il y correspond la relation sans nom de O vers C ;
  - VIA * : il y correspond la relation sans nom de C vers O.

  Il doit exister dans la base de données concernée une relation correspondant à cette désignation.
- Cet ordre doit se trouver dans les contextes de ⟨étiq1⟩ et ⟨étiq2⟩ ;
- Toutes les étiquettes citées (explicitement ou non) doivent être relatives à des ordres REACH, ALLOCATE ou PREPARE WITH déjà traité par un ordre CREATE ;
- Chaque relation désignée doit être du type I-J, K-L avec I > J et K > L.

3. SEMANTIQUE

L'ordre DETACH commande la suppression d'occurrences de relations selon les modalités suivantes :

- soit r_o la réalisation correspondant à ⟨étiq1⟩ et r_i la réalisation correspondant à ⟨étiq2⟩ ;
- pour chaque clause FROM ⟨étiq2⟩ ⟨dés.rel⟩, on demandera soit la suppression de l'occurrence (r_o,r_i) de la relation désignée par ⟨dés.rel⟩ si cette désignation n'est pas suffixée par "*", soit la suppression de l'occurrence (r_i,r_o) de la relation désignée par ⟨dés.rel⟩ si cette désignation est suffixée par "*" ;

- une suppression n'est effectuée que si :
  - r_o et r_i existent effectivement,
  - cette suppression ne viole pas le respect des paramètres I et K de la relation désignée,
  - l'occurrence désignée existe ;
- Si la relation désignée possède une inverse, l'occurrence correspondante est également supprimée ;
- Les suppressions sont commandées dans l'ordre où leurs clauses apparaissent dans l'ordre DETACH.

4. EXEMPLES

```
@DETACH  MACH1  FROM  SGBD1.

@DETACH SYST-EXP FROM CONSTR1
                 FROM LANGAGE1 VIA ACCEPTE*
                 FROM CONSTR2 VIA *.

@DETACH  FROM TERMIN1, FROM TERMIN1, FROM TERMIN3.
```

o

o o

CLOTURE DE CONTEXTE ET ITERATION (@END)

1. SYNTAXE

@END [⟨étiq⟩].

2. REGLES SYNTAXIQUES

- ⟨étiq⟩ est le nom d'une étiquette connue jusqu'à cet endroit, et définie dans un ordre REACH, ALLOCATE ou PREPARE ; il ne correspond qu'un seul ordre END à tout ordre REACH, ALLOCATE et PREPARE ;
- Si la clause ⟨étiq⟩ est absente le END se réfère implicitement au dernier contexte ouvert qui n'a pas encore été clôturé à cet endroit (celui-ci ne peut donc avoir été ouvert par un OPEN).

3. SEMANTIQUE

- L'ordre END définit la fin du contexte ouvert par un ordre REACH, ALLOCATE ou PREPARE. Après son exécution l'étiquette de ce contexte et la réalisation courante qui lui était éventuellement associée sont rendues indiponibles.
- L'ordre END clôture également tous les contextes qui lui sont emboîtés et qui n'auraient pas encore été fermés à cet endroit ; ces contextes ont été ouverts par des ordres OPEN, REACH, ALLOCATE ou PREPARE. Pour ces contextes, l'ordre CLOSE definit une suite de clôtures END et CLOSE *implicites*.
- Lorsqu'un END, implicite ou explicite, est relatif à un ordre REACH, son exécution commande l'itération automatique de la boucle REACH-END ainsi constituée ; les instructions DML et COBOL situées entre l'ordre REACH et l'ordre END qui lui correspond constituent le corps de la boucle, et seront exécutés pour chaque réalisation obéissant aux spécifications de l'ordre REACH.

4. EXEMPLE

Le texte

```
@REACH  CPU = C.
        @REACH MACHINE = M.
               @PREPARE  MEMOIRE = MEM.
@END C.
```

est équivalent à :

```
@REACH  CPU = C
        @REACH  MACHINE = M
                @PREPARE MEMOIRE = MEM

                @END
         @END
@END
```

o

o o

SORTIE PREMATUREE D'UN CONTEXTE (@EXIT)

1. SYNTAXE

@EXIT [⟨ étiq ⟩] [.]

2. REGLES SYNTAXIQUES

- Cet ordre doit se trouver dans le contexte ⟨ étiq ⟩ ;
- ⟨ étiq ⟩ peut avoir été défini par un ordre OPEN, REACH, ALLOCATE ou PREPARE ;
- Si la clause ⟨étiq⟩ est absente, l'ordre EXIT se réfère implicitement à son contexte de base ;
- Un ordre EXIT peut constituer la dernière instruction d'une branche TRUE ou FALSE d'une alternative IF COBOL.

3. SEMANTIQUE

- L'ordre EXIT provoque, lors de son exécution, la sortie prématurée du contexte auquel il fait référence ; l'exécution du programme se poursuit à la première instruction (explicite ou implicite) qui suit le END ou le CLOSE (explicite ou implicite) du contexte que l'on quitte.
- Si l'ordre EXIT n'est pas relatif à son contexte de base, il provoque préalablement à l'effet décrit ci-dessus, l'abandon de tous les contextes actifs emboîtés dans le contexte ⟨ étiq ⟩ .
- Si l'ordre EXIT est relatif à un ordre REACH, il provoque par voie de conséquence l'arrêt prématuré des itérations de cette boucle d'accès, ainsi que de toutes celles qui lui seraient emboîtées et encore actives à cet endroit.

4. EXEMPLE

```
@OPEN  CONFIGUR = C.
        :
       @REACH  MEMOIRE = M  GET  ALL.
                :
                IF I = 0  THEN MOVE 1 TO I
                                    @EXIT
                          ELSE MOVE 2 TO I
                                    @EXIT C.
                :
       @END.
        :
@CLOSE.
```

o

o o

MODIFICATION DES RELATIONS RELATIVES AUX OBJETS ELEMENTAIRES (@MODIFY)

1. SYNTAXE

@MODIFY [⟨étiq⟩]$_0^{99}$.

2. REGLES SYNTAXIQUES

- Cet ordre doit se trouver dans le contexte ⟨étiq⟩.
- ⟨étiq⟩ est le nom d'une étiquette définie dans un ordre REACH, ALLOCATE ou PREPARE suivi de CREATE ;
- Si une clause ⟨étiq⟩ est absente, l'ordre MODIFY se réfère implicitement à son contexte de base ;
- Toutes les clauses ⟨étiq⟩ sont différentes ;
- Les différentes clauses ⟨étiq⟩ sont séparées par un ou plusieurs espaces, par , ou par ; .

3. SEMANTIQUE

- Pour chaque clause ⟨étiq⟩, l'exécution de l'ordre MODIFY va modifier la base de données en associant la réalisation ⟨étiq⟩ aux nouvelles valeurs d'objets élementaires qui ont été rangées dans la zone de communication affectée au contexte ⟨étiq⟩.
- Le programmeur doit avoir au préalable rangé dans cette zone de communication les nouvelles valeurs d'objets élémentaires ainsi que des compteurs y afférents lorsque le nombre de valeurs d'un objet élémentaire affecté d'une clause OCCURS a changé.
- Une tentative de modification d'un objet déclaré IDENT n'est pas prise en compte ; elle est ignorée mais les modifications concernant les autres objets élémentaires sont effectuées.
- Les objets élémentaires soumis à une telle modification sont ceux qui sont spécifiés dans la clause GET ou WITH de l'ordre ayant défini ⟨étiq⟩.
- Si à une étiquette n'est associée aucune réalisation, seule la modification qui la concerne ne sera pas exécutée.
- Les modifications sont exécutées dans l'ordre où elles sont spécifiées dans la commande MODIFY.

4. EXEMPLES

```
@MODIFY.
@MODIFY  CP1.
@MODIFY  UC1, MEM4, MACH.
```

o

o o

COPIE ENTRE VARIABLES DE TRAVAIL (@MOVE)

1. SYNTAXE

@MOVE ⟨var.trav1⟩ TO ⟨var.trav2⟩.

2. REGLES SYNTAXIQUES

- ⟨var.trav1⟩ et ⟨var.trav2⟩ sont des noms de variables de travail.

3. SEMANTIQUE

- Si la variable ⟨var.trav1⟩ est initialisée, l'exécution du MOVE affectera à la variable ⟨var.trav2⟩ la valeur et le type de la variable ⟨var.trav1⟩.
- Si la variable ⟨var.trav1⟩ n'est pas initialisée (vide), l'exécution du MOVE videra la variable ⟨var.trav2⟩ de sa valeur et de son type.
- La valeur et le type de ⟨var.trav2⟩ avant l'exécution du MOVE sont perdus.

4. EXEMPLE

```
@MOVE  $17  TO  $01.
```

o

o o

ITERATION FORCEE D'UNE BOUCLE D'ACCES (@NEXT)

1. SYNTAXE

@NEXT [⟨ étiq⟩] [.]

2. REGLES SYNTAXIQUES

- Cet ordre doit se trouver dans le contexte ⟨ étiq⟩.
- ⟨ étiq⟩ est le nom d'une étiquette définie par un ordre REACH.
- Si la clause ⟨ étiq⟩ est absente, l'ordre NEXT se réfère implicitement à son contexte de base (qui doit donc être ouvert par un REACH).
- Un ordre NEXT peut constituer la dernière instruction d'une branche TRUE ou FALSE d'une alternative IF COBOL.

3. SEMANTIQUE

- Si l'ordre NEXT est relatif à sa boucle de base, son exécution provoque l'abandon de la réalisation courante de cette boucle puis l'itération de cette dernière (l'entièreté du corps de la boucle est exécutée pour la réalisation suivante si elle existe).
- Si l'ordre NEXT n'est pas relatif à sa boucle de base, son exécution provoque d'abord la sortie prématurée de tous les blocs actifs à cet endroit et inclus dans la boucle de base (EXIT), ensuite l'itération forcée de cette boucle de base (NEXT).

4. EXEMPLE

```
@REACH  CPU = C.
        @REACH MEMOIRE = M.
                .
                :
        IF I = 0 THEN
                    @NEXT C
                ELSE
                    @NEXT.
                :
        C.
@END.
```

o

o o

ACCES A UNE BASE DE DONNEES (@OPEN)

1. SYNTAXE

@OPEN ⟨ nom rac ⟩ = ⟨ étiq ⟩.

2. REGLES SYNTAXIQUES

- ⟨ nom rac ⟩ est le nom de la racine d'une base de données ;
- ⟨ étiq ⟩ est un nom unique parmi les noms d'étiquettes du programme ;
- Cet ordre ne peut se trouver dans un bloc OPEN-CLOSE relatif à la même racine ⟨ nom rac ⟩ (c'est-à-dire : on ne peut accéder deux fois simultanément à la même racine) ;
- On ne peut construire dans un même programme plus de 26 blocs OPEN-CLOSE emboîtés l'un dans l'autre (c'est-à-dire : on ne peut travailler simultanément sur plus de 26 bases de données) ;
- A tout ordre @OPEN doit correspondre un ordre @CLOSE,explicite ou non, qui clôture son contexte.

3. SEMANTIQUE

L'ordre OPEN commande l'accès initial à une base de données, c'est-à-dire à sa racine ; après l'exécution de cet ordre, et jusqu'à l'instruction CLOSE correspondante, il sera permis de demander l'accès à des réalisations d'objets complexes de cette base de données.
Dans ce contexte OPEN-CLOSE, cette racine pourra être désignée par la citation de l'étiquette ⟨étiq⟩ à laquelle elle est associée ; une telle désignation sera utilisée en particulier dans la clause FROM d'un ALLOCATE ou d'un REACH, ou encore comme paramètre d'un EXIT.

4. EXEMPLE

```
@OPEN  CONFIGUR = C1.
       :
@CLOSE.
       :
@OPEN  PERSONNEL = P.
       @OPEN  CONFIGUR = C2
@CLOSE P.
```

o

o o

INITIALISATION AVANT CREATION (@PREPARE)

1. SYNTAXE

@PREPARE ⟨ nom OC ⟩ = ⟨ étiq1 ⟩ [FROM ⟨ étiq2 ⟩] [[WITH / GET]$_1^1$ [⟨ liste OE ⟩ / ALL]$_1^1$ IN ⟨ étiq3 ⟩] .

2. REGLES SYNTAXIQUES

- ⟨ nom OC ⟩ est le nom d'un objet complexe de la base de données désignée par ⟨ étiq2 ⟩ ou par le bloc OPEN-CLOSE de base.
- ⟨ étiq1 ⟩ est le nom d'une étiquette qui sera attachée au bloc que le PREPARE définit ; ce nom doit ne pas avoir déjà été défini.
- ⟨ étiq2 ⟩ est le nom d'une étiquette relative à un bloc OPEN-CLOSE ; si cette clause est omise, le bloc OPEN-CLOSE de base est implicitement désigné.
- ⟨ liste OE ⟩ est une liste dont chaque élément est le nom d'un objet élémentaire directement relié à l'objet complexe de nom ⟨ nom OC ⟩.
  Dans cette liste doivent apparaître les noms de tous les objets élémentaires obligatoires, c'est-à-dire ceux dont la clause OCCURS soit est absente, soit est caractérisée par (1,J) ; les noms d'autres objets élémentaires peuvent apparaître.
- ⟨ étiq3 ⟩ est le nom d'une étiquette définie par un ordre ALLOCATE relative à l'objet ⟨ nom OC ⟩ et dont la clause WITH(GET) contient les noms de tous les objets élémentaires obligatoires pour l'objet ⟨ nom OC ⟩.
- Cet ordre doit se trouver dans les contextes ⟨ étiq2 ⟩, et ⟨ étiq3 ⟩, s'ils sont spécifiés.
- A tout ordre PREPARE doit correspondre un ordre END, explicite ou non, qui clôture son contexte.
- Entre un ordre PREPARE et certains ordres lui faisant référence (MODIFY, REACH, ATTACH, DETACH, TRANSFER , SUPPRESS, CREATE non attaché à ce PREPARE), doit se trouver un ordre CREATE qui lui fait référence.
- Dans le contexte d'une PREPARE doit se trouver un ordre CREATE qui lui fait référence.
- Il n'est pas permis, dans le contexte d'un PREPARE IN de citer des objets élémentaires relatifs à ⟨ étiq1 ⟩ ; on les désignera plutôt par rapport à ⟨ étiq3 ⟩.
- Deux ou plusieurs contextes PREPARE IN emboîtés ne peuvent faire référence à la même étiquette ⟨ étiq3 ⟩.

3. SEMANTIQUE

- L'ordre PREPARE définit un contexte qui d'étendra jusqu'au END qui le clôturera en aval dans le programme ; à ce contexte est associée une étiquette de nom ⟨étiq1⟩ qui pourra dans la suite être affectée à une réalisation de l'objet ⟨nom OC⟩ qui sera créée par un ordre CREATE situé dans le contexte ⟨étiq1⟩. Après l'exécution du PREPARE, et dans un contexte, l'étiquette ⟨étiq1⟩ est connue et non affectée.
  La clause FROM, explicite ou non indique au compilateur la base de données à laquelle l'objet ⟨nom OC⟩ appartient.
- L'ordre PREPARE GET(WITH réserve dans la zone de communication de l'espace destiné à contenir les valeurs d'objets élémentaires cités dans la clause GET (ou WITH) ainsi que les données techniques ; la clause GET(WITH)ALL indique que tous les objets élémentaires sont spécifiés alors que l'absence de clause GET(WITH) indique qu'aucun objet n'est demandé. Cet espace est lié au contexte ⟨étiq1⟩.
- L'ordre PREPARE initialise toutes les zones correspondant aux objets élémentaires et aux données techniques.
  S'il s'agit d'un PREPARE WITH(GET), ces zones sont celles qui sont liées au contexte ⟨étiq1⟩ ; s'il s'agit d'un PREPARE IN, ces zones sont celles qui sont liées au contexte ⟨étiq3⟩.
  Cette initialisation consiste :
  - à affecter une valeur conventionnelle aux zones relatives aux objets élémentaires : 0 pour les numériques, "?" pour les alphabétiques ;
  - à positionner tous les compteurs à 0 ;
  - à désinitialiser l'étiquette si elle était au préalable affectée à une réalisation.

4. EXEMPLES

```
@PREPARE  SGBD  WITH (NOM, SIGLE).
@PREPARE  U-BANDE = UB GET ALL.
@PREPARE  MACHINE = M1.
@PREPARE  CPU = C2 IN C1.
```

o

o o

ACCES A DES REALISATIONS D'OBJETS COMPLEXES (@REACH)

1. SYNTAXE

Format 1 :

@REACH ⟨ nom OC ⟩ = ⟨ étiq1 ⟩ [FROM ⟨ étiq2 ⟩] [VIA ⟨ nom rel ⟩]

[IF (⟨ condition ⟩)] $\left[\begin{array}{l} \text{GET} \left[\begin{array}{c} \text{ALL} \\ (\langle \text{ liste OE} \rangle) \end{array}\right]_1^1 \\ \text{IN } \langle \text{ étiq3} \rangle \end{array}\right]$.

Format 2 :

@REACH ⟨ nom OC ⟩[⟨ var.trav ⟩] = ⟨ étiq1 ⟩ [FROM ⟨ étiq4 ⟩] $\left[\begin{array}{l} \text{GET} \left[\begin{array}{c} \text{ALL} \\ (\langle \text{ liste OE} \rangle) \end{array}\right]_1^1 \\ \text{IN } \langle \text{ étiq3} \rangle \end{array}\right]$.

2. REGLES SYNTAXIQUES

- ⟨ nom OC ⟩ est le nom d'un objet complexe d'une base de données qui fait l'objet d'un contexte OPEN-CLOSE dans lequel cet ordre se trouve.
- ⟨ étiq1 ⟩ est le nom d'une étiquette de boucle qui sera attachée au contexte défini par cet ordre ; ce nom ne peut pas être déjà défini.
- ⟨ étiq2 ⟩ est le nom d'une étiquette attachée à un contexte dans lequel on se trouve ; si la clause FROM est omise l'rodre désigne implicitement son contexte de base. Ce contexte peut être défini par un ordre OPEN, REACH, ALLOCATE ou PREPARE suivi d'un CREATE.
- ⟨ nom rel ⟩ est le nom d'une relation ; l'absence de clause VIA indique qu'une relation sans nom est désignée. Il doit exister dans la base de données une relation ainsi nommée de l'objet correspondant à ⟨ étiq2 ⟩ vers l'objet ⟨ nom OC ⟩.
- ⟨ étiq4 ⟩ est le nom d'une étiquette dans le contexte de laquelle on se trouve, et déclarée par un ordre OPEN ; l'absence de cette clause FROM indique que l'on désigne implicitement le bloc OPEN-CLOSE de base.
- ⟨ condition ⟩ est la description d'une expression booléenne de critères simples. Les expressions booléennes sont limitées aux formes suivantes :
  - critère simple,
  - somme logique de critères simples (critères unis par des OR),

- produit logique de critères simples (critères unis par des AND);
- produit de *sommes de critères simples* et de *critères simples*, chaque somme étant entourée d'une paire de parenthèses.

- Un critère simple est constitué de la manière suivante :

  soit ⟨ nom OE ⟩ ⟨ op.compl ⟩ ⟨ dés.valeur ⟩,

  soit ⟨ nom OE ⟩ ⟨ op.comp2 ⟩(⟨ liste dés.valeurs ⟩).

  où :

  - ⟨ nom OE ⟩ est le nom d'un objet élémentaire directement relié à l'objet complexe ⟨ nom OC ⟩, dont la valeur est inférieure à 256 caractères et dont la clause occurs implicite ou non indique : OCCURS (I,1) ;
  - ⟨ op.compl ⟩ est l'un des opérateurs de comparaison : =, NOT =, >, NOT >, <, NOT <, >=, < = ;
  - ⟨ op.comp2 ⟩ est l'un des opérateurs de comparaison : =, NOT = ;
  - ⟨ dés.valeur ⟩ est la désignation d'une valeur, soit sous forme de constante, ou de nom de zone COBOL ou de désignation de valeur d'objet élémentaire dans la zone de communication ;
  - ⟨ liste dés.valeur ⟩ est une liste de ⟨ dés.valeur ⟩ séparée par des virgules ;
  - ⟨ liste OE ⟩ est une liste de noms d'objets élémentaires directement reliés à l'objet ⟨ nom OC ⟩ ;
  - ⟨ étiq3 ⟩ est le nom d'une étiquette attachée à un contexte dans lequel on se trouve et déclarée par un ordre ALLOCATE relatif à l'objet complexe ⟨ nom OC ⟩;
  - ⟨ var.trav. ⟩ est le nom d'une variable de travail ;
  - Dans le contexte ⟨ étiq1 ⟩ on ne peut désigner la valeur d'objets élémentaires relatives à cette étiquette si la clause IN est spécifiée ; on utilisera à cet effet pour cette désignation, l'étiquette ⟨ étiq3 ⟩.

3. SEMANTIQUE

a. Description générale des accès

Sous le format 1, l'ordre REACH commande l'accès à des réalisations d'un objet complexe par une relation d'accès.
Sous le format 2, l'ordre REACH commande l'accès à la réalisation d'un objet complexe désignée par une variable de travail.

b. Format 1

L'ordre REACH commande une boucle d'accès qui exécute la recherche et la mise à disposition des réalisations nécessaires de l'objet complexe ⟨nom OC⟩ obtenues à partir de la réalisation ⟨étiq2⟩ par la relation ⟨nom rel⟩ (ces clauses étant explicites ou non) ; parmi ces réalisations, seules seront retenues celles auxquelles sont associées des valeurs d'objet élémentaires obéissant à la condition ⟨condition⟩. L'interprétation de ⟨condition⟩ correspond à celle qui est d'usage en logique, et en particulier dans la condition COBOL ; un critère qui fait intervenir une ⟨liste dés.valeurs⟩ s'interprète comme suit :

- ⟨nom OE⟩ = (⟨liste dés.valeurs⟩) : la valeur de l'objet élémentaire attachée à la réalisation ⟨étiq1⟩ appartient à la liste des valeurs désignées.
- ⟨nom OE⟩ NOT = (⟨liste dés.valeurs⟩) : la valeur de l'objet élémentaire attachée à la réalisation ⟨étiq1⟩ n'appartient pas à la liste des valeurs désignées.

La réalisation courante de la boucle sera affectée à l'étiquette ⟨étiq1⟩. Le corps de la boucle sera exécuté pour chaque réalisation réussissant la sélection définie par les diverses clauses ; il sera donc effectué selon le cas 0,1 ou plusieurs fois.
Si dans la condition ⟨condition⟩ se trouvent une ou plusieurs clés d'accès (dans le sens qui a été défini dans le DDL), il se peut qu'une technique d'accès rapide soit adoptée par le système de gestion de la base de données ; ce choix est laissé au système et à l'inplémenteur des algorithmes d'optimisation des accès.

c. Format 2

L'ordre REACH commande l'accès à la réalisation référencée par la variable ⟨var.trav.⟩ pourvu que cette dernière soit initialisée, que son contenu spécifie une réalisation encore vivante et que cette dernière appartienne à l'objet ⟨nom OC⟩.
Le paramètre FROM, explicite ou non, indique à quelle base de données appartient la réalisation.
Si la réalisation est trouvée et vérifie les conditions qui ont été évoquées, elle sera affectée à l'étiquette ⟨étiq1⟩ et le corps de la boucle sera exécuté une fois ; dans le cas contraire, la boucle ne sera pas exécutée.

d. Accès aux valeurs d'objets élémentaires

Simultanément à l'accès à chaque réalisation, l'obtention de valeurs d'objets élémentaires qui lui sont reliées peut être demandée par la clause GET. La clause GET spécifie les objets élémentaires dont les valeurs sont requises ; GET ALL signifie que tous les objets élémentaires directement reliés à l'objet ⟨nom OC⟩ sont spécifiés ; l'absence de clause GET (et de clause IN) signifie que l'on ne désire aucune valeur d'objet élémentaire.
Lorsque la clause IN est présente, les objets élémentaires spécifiés sont ceux qui apparaissent dans la clause WITH (ou GET) de l'ordre ALLOCATE qui définit ⟨étiq3⟩ ;

e. Réservation d'espace dans la zone de communication

L'ordre REACH non muni de clause IN réserve dans la zone de communication de l'espace qui accueillera non seulement des données techniques, mais aussi, si l'utilisateur le désire (clause GET) les valeurs d'objets élémentaires.
Si la clause IN est spécifiée, ces données et ces valeurs seront rangées dans l'espace qui a été réservé par l'ordre ALLOCATE.

f. Contexte

L'ordre REACH définit un contexte qui s'étend jusqu'au END qui lui correspond en aval du programme.

4. EXEMPLES

a. Conditions

```
- CPU ... IF (MOT = 8)
          IF (NUMERO = '370          158')
          IF (MOT = (8, 16, NUMBER1))
          IF (MOT = 8 AND PRIX = COST)
          IF (MOT > = 32 OR PRIX < MAX)
- MEMOIRE IF ((CYCLE < 100 OR TECHNO = "CMOS') AND PRIX < 50000 AND BLOC > 32)
```

b. Accès

```
@REACH CONSTRUCTEUR = C1 FROM CONF IF (SIGLE = NAME) GET ALL.
@REACH CPU = CPU1 VIA FOURNI IN CPUØ.
@REACH MEMOIRE $14 = MEM1 GET (NUMERO, PRIX).
@REACH TERMINAL = T.
```

o

o o

SAUVETAGE D'UNE REFERENCE DE BOUCLE (@SAVE)

1. SYNTAXE

@SAVE [⟨étiq⟩] IN ⟨var.trav.⟩.

2. REGLES SYNTAXIQUES

- Cet ordre doit se trouver dans le contexte ⟨étiq⟩.
- ⟨étiq⟩ est le nom d'une étiquette définie dans un ordre REACH, ALLOCATE ou PREPARE suivie d'un CREATE ; ce nom ne peut être "IN".
- Si la clause ⟨étiq⟩ est absente, l'ordre SAVE se réfère implicitement à son contexte de base.
- ⟨var.trav.⟩ est le nom d'une variable de travail.

3. SEMANTIQUE

- Si à ⟨étiq⟩ est associée une réalisation courante, l'exécution de l'ordre SAVE affecte à la variable ⟨var.trav.⟩ une valeur qui est la référence à cette réalisation, et un type qui est l'objet auquel appartient cette réalisation.
- Si à ⟨étiq⟩ n'est encore associée aucune réalisation, l'exécution du SAVE videra la variable ⟨var.trav.⟩ de sa valeur et de son type.

4. EXEMPLES

```
@SAVE  MEM  IN $09.
@SAVE  IN $72.
```

o

o o

SUPPRESSION D'UNE REALISATION D'UN OBJET COMPLEXE (@SUPPRESS)

1. SYNTAXE

@SUPPRESS [⟨ étiq⟩] $\begin{bmatrix}\text{NEXT}\\ \text{EXIT}\end{bmatrix}_1^1$.

2. REGLES SYNTAXIQUES

- Cet ordre doit se trouver dans le contexte ⟨ étiq⟩.
- ⟨ étiq⟩ est le nom d'une étiquette définie dans un ordre REACH, ALLOCATE ou PREPARE suivi de CREATE ; le nom peut être 'NEXT' ou 'EXIT'.
- Si la clause ⟨ étiq⟩ est absente, l'ordre SUPPRESS se réfère implicitement à son contexte de base.
- Le bloc auquel se réfère l'ordre SUPPRESS ne peut se trouver dans aucun contexte auquel serait associée une réalisation d'un objet de la *descendance forte* de l'objet correspondant à ce bloc.
  La *descendance forte* d'un objet A est définie comme l'ensemble constitué des objets complexes suivants :
  - l'objet A lui-même,
  - la descendance forte de chaque objet cible d'une relation du type K > 0 dont A est l'origine, ou origine d'une relation du type I > 0 dont A est la cible.

3. SEMANTIQUES

- L'exécution de l'ordre SUPPRESS supprime de la base de données la réalisation associée au contexte auquel se réfère l'ordre (⟨ étiq⟩ ou contexte de base) ; si à ce contexte n'est associée aucune réalisation, aucune action n'est effectuée.
- Quelle que soit l'issue de l'opération précédante, la clause NEXT ou EXIT est exécutée, provoquant impérativement soit l'itération forcée, soit l'abandon prématuré du bloc auquel se réfère l'ordre SUPPRESS. Dans le cas où ce contexte est défini par un ALLOCATE ou PREPARE, la clause NEXT est autorisée et se comporte comme un EXIT.
- La suppression d'une réalisation peut entraîner la suppression d'occurrences de relations ainsi que d'autres réalisations dans la base de données. Ces suppressions sont réalisées de manière à rendre à la base de données un état cohérent vis-à-vis des contraintes I-J, K-L des relations. Les règles de suppressions indirectes sont :

- si à un instant donné du processus de suppression une réalisation est cible de $K_R > 0$ occurrences de la relation R, ou origine de $I_S > 0$ occurrences de la relation S et si le processus supprime ultérieurement l'une de ces occurrences, cette réalisation sera également supprimée.
- toutes les occurrences de relations auxquelles participe une réalisation à supprimer seront également supprimées.

- Les données de la zone de communication qui correspondent à des réalisations modifiées sont également mises à jour (cas des compteurs).

4. EXEMPLES

-
```
@REACH  CONSTRUCTEUR = C.
        @REACH  CPU = CP VIA DISTRIBUE.
                @REACH MACHINE = M
                  :
                  @SUPPRESS CP NEXT.
                @END.
        @END
@END.
```

- Le programme suivant viole la règle de la descendance forte :

```
@REACH  MACHINE = M.
        @REACH CPU = CP
               @SUPPRESS EXIT.
        @END.
        ⟨instruction⟩
@END.
```

En effet, la descendance forte de CPU est {CPU,MACHINE} et le contexte relatif au SUPPRESS (CP) est emboîté dans un contexte associé à l'objet MACHINE ; sans la règle évoquée, la suppression d'une réalisation de CPU entraînerait la suppression de la réalisation M de MACHINE et risquerait de provoquer une anomalie dans le fonctionnement de ⟨instruction⟩.

o

o o

TEST ET COMPARAISON D'ETIQUETTES ET VARIABLES (@TEST)

1. SYNTAXES

a. $$@TEST\begin{bmatrix} \langle étiq1 \rangle \\ \langle var.trav1 \rangle \end{bmatrix}_1^1 .$$

b. $$@TEST\begin{bmatrix} \langle étiq1 \rangle \\ \langle var.trav1 \rangle \end{bmatrix}_1^1 = \begin{bmatrix} \langle étiq2 \rangle \\ \langle var.trav2 \rangle \end{bmatrix}_1^1 .$$

2. REGLES SYNTAXIQUES

- Si ⟨étiq1⟩ et/ou ⟨étiq2⟩ sont spécifiés, l'ordre TEST doit se trouver dans les contextes ⟨étiq1⟩ et/ou ⟨étiq2⟩.
- ⟨étiq1⟩ et ⟨étiq2⟩ sont des noms d'étiquettes définies dans un ordre REACH, ALLOCATE ou PREPARE.
- ⟨var.trav1⟩ et ⟨var.trav2⟩ sont des noms de variables de travail.

3. SEMANTIQUE

a. Format 1

L'exécution de l'ordre TEST positionne à 1 la variable COBOL SYS-REG si l'étiquette ⟨étiq1⟩ ou la variable ⟨var.trav1⟩ sont initialisées, et à 0 dans le cas contraire.

b. Format 2

L'exécution de l'ordre TEST positionne à 1 la variable COBOL SYS-REG si les deux termes de la comparaison (⟨étiq1⟩/⟨var.trav1⟩ et ⟨étiq2⟩/⟨var.trav2⟩) font référence à la même réalisation, et à 0 dans le cas contraire.
Si les deux termes sont non initialisés, SYS-REG sera mis à 0.

4. EXEMPLES

```
@TEST  UC.
@TEST  $15.
@TEST  MEM1 = MEM2.
@TEST  $08 = PERIPH.
@TEST  $16 = $99.
```

°

° °

REMPLACEMENT D'OCCURRENCES DE RELATIONS (@TRANSFER)

1. SYNTAXE

@TRANSFER [⟨ étiq1 ⟩] [⟨ transfert partiel ⟩]$_1^{20}$.

⟨ transfert partiel ⟩ ::= [FROM [⟨ étiq2 ⟩]$_1^{20}$] TO [⟨ étiq3 ⟩]$_1^{20}$ [⟨ dés.rel ⟩]

2. REGLES SYNTAXIQUES

- ⟨ étiq1 ⟩, ⟨ étiq2 ⟩ et ⟨ étiq3 ⟩ sont des noms d'étiquettes relatives à des ordres REACH, ALLOCATE ou PREPARE suivis d'un CREATE ;
- Si ⟨ étiq1 ⟩ est absente, l'ordre désigne implicitement l'étiquette du contexte de base ;
- Cet ordre doit se trouver dans tous les contextes ⟨ étiq1 ⟩, ⟨ étiq2 ⟩ et ⟨ étiq3 ⟩ ;
- Dans un même ⟨ transfert partiel ⟩, l'étiquette ⟨ étiq2 ⟩ et ⟨ étiq3 ⟩ doivent être relatives au même objet ;
- ⟨ dés.rel ⟩ est la désignation d'une relation entre objets complexes. Soit O l'objet correspondant à ⟨ étiq1 ⟩ (ou au contexte de base) et C l'objet correspondant aux étiquettes ⟨ étiq2 ⟩ et ⟨ étiq3 ⟩. Les formes de désignation possibles sont les suivantes :
  - VIA ⟨ nom rel ⟩ : il y correspond la relation de nom ⟨ nom rel ⟩ de O vers C,
  - VIA ⟨ nom rel ⟩*: il y correspond la relation de nom ⟨ nom rel ⟩ de C vers O,
  - ⟨ vide ⟩ : il y correspond la relation sans nom de O vers C,
  - VIA * : il y correspond la relation sans nom de C vers O.

  Il doit exister dans la base de données une relation ainsi désignée ;
- Les relations désignées d'origine O doivent être telles que I > 0. Les relations désignées d'origine C doivent être telles que K > 0 ;
- Dans un ⟨ transfert partiel ⟩, la clause FROM peut être omise si la relation désignée soit est d'origine O et telle que J = 1, soit est de cible O et telle que L = 1 ; dans ce cas d'omission, est désignée implicitement, l'unique réalisation respectivement cible ou origine de l'occurrence de ⟨ dés.rel ⟩ à laquelle la réalisation ⟨ étiq1 ⟩ participe;
- Le nombre d'étiquettes ⟨ étiq2 ⟩ et le nombre d'étiquettes ⟨ étiq3 ⟩ d'un même transfert partiel peuvent être différents ;
- Une étiquette ⟨ étiq2 ⟩ ne peut s'appeler "TO" et une étiquette ⟨ étiq3 ⟩ ne peut s'appeler "VIA".

3. SEMANTIQUE

- Pour chaque ⟨transfert partiel⟩, l'exécution de l'ordre TRANSFER remplace des occurrences de la relation ⟨dés.rel⟩ par d'autres occurrences de cette même relation, ces occurrences concernent toutes la réalisation ⟨étiq1⟩ (implicite ou explicite) si elle existe ; ce remplacement constitue du point de vue global de l'utilisation de la base de données une action indécomposable.
  Appelons o la réalisation attachée à ⟨étiq1⟩, C1 les réalisations attachées à ⟨étiq2⟩ et C2 les réalisations attachées aux ⟨étiq3⟩ pour un même ⟨transfert partiel⟩.
  En cas d'absence de certaines clauses on se référera aux options par défaut définies plus haut.
  A la suite de l'exécution de chaque ⟨transfert partiel⟩, les occurrences de ⟨dés.rel⟩ (o,C1) ont été supprimées et les occurrences de cette relation (o,C2) ont été créées du moins si o et C existent effectivement (si la désignation de relation est suffixée par *, le raisonnement sera tenu pour les occurrences (c1,o) et (C2,o).
  Si la réalisation o ne vérifiait plus ses conditions d'existence suite à l'exécution d'un transfert partiel , celui-ci ne serait pas exécuté.
  Si la création de certaines occurrences (o,C2) ou (C2,o) amenaient à violer la caractéristique J ou L de la relation ⟨dés.rel⟩$_k$, elles ne seront pas créées.
- Si les relations spécifiées sont munies d'une inverse, les occurrences de ces dernières sont également remplacées ou créées.

4. EXEMPLES

```
@TRANSFER  CPU1  FROM CONSTR1 TO CONSTR2 VIA FOURNI.
@TRANSFER  TO  CONSTR2 VIA FOURNI.
@TRANSFER  MEM1  TO CONSTR2.
@TRANSFER  SGBD1  FROM C1, C2 TO C3, C4, C5.
@TRANSFER  MACH  TO INSTAL VIA COMPORTE, TO CPU2.
@TRANSFER  MACH  TO CPU3 VIA *.
```

o

o o

ERREURS COMPILATION

LES MESSAGES DU COMPILATEUR DML

Lorsque le compilateur DML détecte une erreur ou, d'une manière générale, une anomalie lors de l'analyse d'un programme DML, il le signale au moyen du n° de la ligne où cette anomalie est apparue, d'un code de quatre chiffres, d'un libellé et d'un paramètre éventuel. Les messages DML sont imprimés sur le listing de compilation après le texte source et avant les erreurs COBOL ; les lignes explicitant ces dernières sont d'ailleurs préfixées chacune par le numéro de la ligne du texte source (donc avant expansion par le compilateur DML), l'utilisateur n'a donc jamais à analyser que le texte COBOL du programme source. On notera que la numérotation des lignes est effectuée par le compilateurs DML et qu'elle remplace toute numérotation préalable éventuelle.

1. LES COMPOSANTS DU MESSAGE

- *Numéro de la ligne* dans le programme source où l'anomalie a été détectée.

- *Code numérique* de l'anomalie ; ce code est constitué de deux parties :
  - numéro du verbe ou de la fonction relatif à l'instruction dans laquelle une anomalie a été détectée (2 chiffres) ;
  - code de l'anomalie (2 chiffres).

  Les messages globaux cependant n'obéissant pas à cette règle.

- *Libellé du message*

- *Paramètre éventuel* : en général, symbole ou identificateur ayant entraîné la détection de l'anomalie

2. LES NUMEROS DE VERBES ET DE FONCTIONS

00 :	PROGRAMME PRINCIPAL
01 :	OPEN
02 :	CLOSE
03 :	REACH
04 :	EXIT
05 :	NEXT
06 :	END

07 :	ALLOCATE
08 :	PREPARE
09 :	CREATE
10 :	MODIFY
11 :	ATTACH
12 :	DETACH
13 :	TRANSFER
14 :	SUPPRESS
15 :	TEST
16 :	SAVE
17 :	MOVE
18 :	CLEAR
19 :	COUNT
20 :	PROTECTION

3. LES MESSAGES GLOBAUX

Les messages sont émis lorsque le compilateur ne peut plus poursuivre son analyse, soit à cause du programme-source, soit à cause du compilateur lui-même (erreur ou saturation des tables). Le code numérique contient le préfixe 'SY'.

SYS1	WORKING-STORAGE OR PROCEDURE DIVISION MISSING
SYS2	STACK NOT EMPTY AT THE END OF THE PROGRAM
SYS3	PROGRAM FILE TOO SHORT
SYS4	MORE THAN 99 ENTRIES IN THE STACK
SYS5	OVERFLOW OF A COMPILER TABLE
SYYY	PRECOMPILER ERROR
SY$$	I/O ERROR IN SCHEMA READING ($$ IS THE ERROR CODE OF THE SCHEMA READING MODULE).

4. LES MESSAGES GENERAUX

Ces messages sont communs à la plupart des verbes DML ; ils peuvent éventuellement être accompagnés d'un paramètre. Dans la liste ci-dessous, le n° du verbe a été remplacé par 'XX'.

XX01	SYNTACTICAL ERROR
XX02	LABEL UNKNOWN IN THIS CONTEXT
XX03	INVALID LABEL TYPE

XX04	INCORRECT LABEL, VARIABLE OR DATA NAME
XX05	DUPLICATED LABEL
XX06	E-O NAME UNKNOWN, DUPLICATED, OR TOO LONG VALUE
XX07	UNKOWN C-O NAME
XX08	UNKNOWN RELATION NAME
XX09	RELATION CHARACTERISTICS VIOLATION
XX10	USE OF LABEL OF INCORRECT MACRO - NO DIAGNOSTIC
XX11	USE/CLOSE OF A PREPARE NOT FOLLOWED BY CREATE
XX12	USE OF OBJECTS IN DIFFERENT DATA BASES

5. LES MESSAGES SPECIALISES

Ces messages sont spécifiques d'une fonction ou d'un verbe.

	MAIN
0051	SYNTAX ERROR
0052	UNKNOWN DML VERB
0053	THE DML VERB IS NOT THE FIRST SYMBOL
	OPEN
0117	DATA BASE ALREADY OPENED
0118	UNKNOWN DATA BASE
	REACH
0313	INVALID BOOLEAN OPERATOR
0314	INVALID COMPARISON OPERATOR
0315	CHARACTER STRING TOO LONG OR NON CONTIGOUS
0316	ERROR IN THE LIST OF VALUE
0317	CLAUSE 'IN' INCOMPATIBLE WITH OBJECT TYPE
	PREPARE
0842	INCORRECT OBJECT TYPE IN CLAUSE 'IN'
0843	MANDATORY E-O MISSING
	CREATE
0944	PROGRAM STRUCTURE ERROR
0951	DUPLICATED RELATION NAME
0953	FORBIDDEN LABEL IN A TARGET LIST

MODIFY

1014 MORE THAN 20 LABELS

1015 DUPLICATED LABEL IN THE LIST

TRANSFER

1314 TOO ~~MUCH~~ TARGETS IN A CLAUSE 'TO' MANY

1315 MORE THAN 20 LABELS IN A PARTIAL TRANSFER

1316 COMMON LABELS IN 'FROM' AND 'TO' LIST

1317 MORE THAN 20 PARTIAL TRANSFERS

SUPPRESS

1415 A FORBIDDEN OBJECT REALIZATION IS IN THE STACK

PROTECTION

2014 MORE THAN 255 OBJECTS SPECIFIED IN THE PROGRAM

2015 MORE THAN 24 DATA BASES OPENED

°

° °

LES CODES D'ERREUR DU SYSTEME SPHINX

Au terme de l'exécution de chaque ordre DML, le système SPHINX indique au programme d'application la manière dont les opérations se sont effectuées. Un code de retour général est placé dans SYS-ERROR, et si nécessaire, des codes de retour détaillés sont placés dans le vecteur SYS-ERLIST dont le nombre d'éléments significatif est le contenu de SYS-ERNBR ; ce vecteur est garni lorsque l'exécution d'un ordre à opérations multiples (MODIFY, CREATE, TRANSFER, ATTACH, DETACH) s'est accompagné d'une anomalie légère.

Le code de retour général (SYS-ERROR) indiquera trois classes de conditions de terminaison :

SYS-ERROR = '00' : l'opération s'est effectuée normalement.

'01' $\leqslant$ SYS-ERROR $\leqslant$ '10' : une anomalie légère est apparue, l'opération n'a pas été effectuée complètement, mais le programme peut se dérouler normalement.

SYS-ERROR > '10' ou SYS-ERROR < '00' : une anomalie grave a été détectée ou un accident est survenu , affectant l'intégrité de la base de données concernée, celle du système SPHINX ou encore celle du programme application ; le déroulement normal de ce dernier est arrêté et l'exécution se poursuit à partir de l'étiquette SYS-DUMP que tout programme DML doit contenir et qui se trouve à l'extérieur de tout contexte ; avant ce branchement, le système clôture tous les contextes de ce programme et coupe la communication.

Le vecteur SYS-ERLIST contiendra des codes compris entre '00' et '09' si au moins une anomalie apparaît dans un ordre à opérations multiples et si ces anomalies sont légères ; SYS-ERROR contiendra alors un code de '01' à '09' correspondant au maximum des codes du vecteur. (sauf cas particulier du CREATE)

1. LES CODES D'ANOMALIE GRAVE

Ces codes indiquent que l'ordre demandé n'a pas été exécuté et que d'une manière générale le travail sur la base de données (et parfois avec SPHINX) doit être stoppé. Nous classerons ces codes selon la source de l'anomalie.

1.1. Le système SPHINX est inaccessible ou refuse le travail

77 : la banque de données est bloquée suite à la détection d'une perte d'intégrité ;

80 : le système SPHINX vient d'être tué suite à une erreur grave (SPHINX ne répond pas au gestionnaire des communications) ;

82 à 85 : erreur dans les communications avec SPHINX ;

86 : le système SPHINX n'a pas encore répondu x minutes après la dernière demande (x = 10 minutes) ;

87 :
- la tâche du programme d'application est encore renseignée dans les tables de SPHINX comme supprimée, ou bien
- l'opérateur de SPHINX vient de supprimer la tâche du programme d'application même si cette dernière fonctionnait correctement.

88 : le système SPHINX est dans l'état 'SHUTDOWN' et refuse tout nouveau programme d'application ;

89 : le système SPHINX n'est pas chargé.

1.2. Une erreur a été détectée dans le programme d'application

58 : mauvaise manipulation d'une variable de travail (O.C. incorrecte) ;

59 : mauvaise manipulation d'une variable de travail (B.D. incorrecte) ;

75 : version de la base de données incompatible avec la version du programme d'application ;

97 et 98 :destruction d'un paramètre d'appel à SPHINX dans le programme d'application ;

98 : un branchement (GO TO, PERFORM, USE) dans le programme ne respecte pas les règles de la structure des blocs.

1.3. Une erreur a été détectée dans la base de données

77 : cf. plus haut ;

79 : erreur dans le schéma interne de SPHINX.

1.4. Une limite du système a été atteinte

71 : troncature des valeurs lues dans la base de données ;

72 à 74 : paramètres d'appel trop longs;

81 : buffer de communication trop petit pour la réponse.

1.5. Autres codes

Erreurs dans la gestion physique de la base de données.

2. LES CODES D'ANOMALIES LEGERES

Un certain nombre d'ordres DML sont l'objet du renvoi de codes de retour dans SYS-ERROR compris entre '01' et '09' ; il s'agit de l'ordre OPEN et des ordres de mise à jour MODIFY, CREATE, ATTACH, DETACH et TRANSFER.

2.1. Ordre OPEN

Lorsque la base de données n'existe pas (c'est-à-dire n'a pas été initialisée) ou n'est pas en ligne, le système indique :

SYS-ERROR = '10'

L'exécution se poursuit après l'ordre CLOSE correspondant au contexte de cet OPEN.

2.2. Ordre MODIFY

'01' : liste des valeurs à modifier vide, pas de modification effectuée ;

'03' : l'étiquette n'est pas affectée à une réalisation, par de modification effectuée.

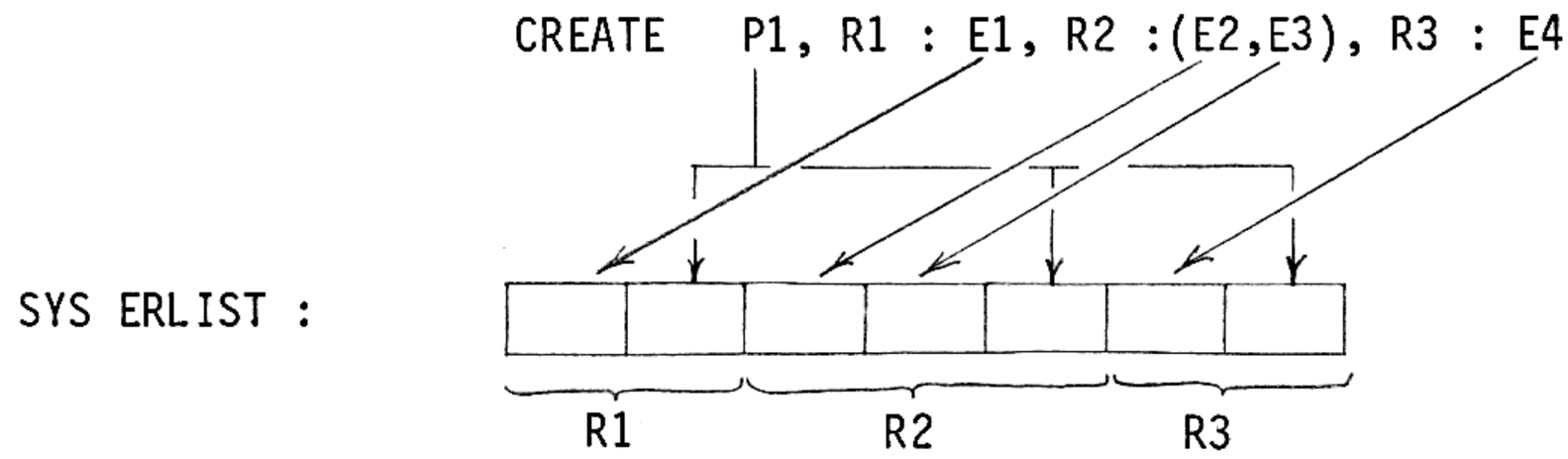
Chaque élément de SYS-ERLIST correspond à une étiquette (explicite ou non) de la liste spécifiée dans l'ordre MODIFY ; cette correspondance se fait dans l'ordre naturel de gauche à droite.

2.3. Ordre CREATE

Un ordre CREATE demande de relier la réalisation à créer à un certain nombre de réalisations par des relations spécifiées ; chaque relation fait l'objet d'une clause qui indique le nom de la relation ainsi qu'un certain nombre d'étiquettes. A chacune de ces clauses correspond dans le vecteur d'erreurs une suite contiguë de codes dont le dernier (code origine) concerne la réalisation à créer, et les autres (code cibles) les réalisations correspondant

aux étiquettes spécifiées, dans l'ordre où elles se présentent dans la clause ; chaque code indique la situation de cette réalisation notamment vis-à-vis du respect des caractéristiques de la relation.

Exemple :



les codes cibles :

'00' : pas d'anomalie.

'01' : cette étiquette désigne une réalisation déjà rencontrée dans cette liste.

'03' : cette étiquette n'est affectée à aucune réalisation.

'04' : la réalisation correspondante est déjà cible de L occurrencesde la relation (violation du paramètre L de la relation).

les codes origines:

'00' : les paramètres I et J de la relation seront respectés pour la réalisation à créer.

'05' : violation du paramètre J de la relation.

'06' : violation du paramètre I de la relation.

'07' : plus de 61 cibles distinctes sont citées pour cette relation.

le code global SYS-ERROR :

'00' à '07' : il s'agit du plus élevé des codes de SYS-ERLIST.

'08' : la réalisation à créer est associée à un nombre insuffisant de valeurs d'un objet élémentaire.

'09' : il existe déjà une réalisation associée à la valeur de l'objet élémentaire identifiant.

Ces deux derniers codes ne signifient pas qu'aucune anomalie n'est renseignée dans SYS-ERLIST , ce vecteur devra être consulté.

La réalisation a été créée si SYS-ERROR = '00', '01' ou '03' ; elle ne l'a pas été dans tous les autres cas. D'autre part, *si la réalisation n'a pas été créée, l'exécution du programme se poursuit après le END qui clôture le contexte de base du CREATE* ; c'est donc à cet endroit que sera rédigée l'analyse des codes de retour si SYS-ERROR > '03'.

2.4. Ordre ATTACH

A chaque occurrence spécifiée dans l'ordre correspond un élément dans SYS-ERLIST ; le rang de ces derniers et celui de la clause qui leur correspond dans l'ordre ATTACH.
SYS-ERROR reprend le code le plus élevé de SYS-ERLIST.

'00' : occurrence créée.
'01' : l'occurrence existait déjà.
'03' : l'étiquette spécifiant la cible n'est affectée à aucune réalisation.
'05' : tentative de violation de la caractéristique J de la relation ; occurrence non créée.
'07' : tentative de violation de la caractéristique L de la relation ; occurrence non créée.
'09' : l'étiquette (explicite ou non) spécifiée n'est affectée à aucune réalisation ; aucune occurrence n'est créée.

2.5. Ordre DETACH

Le principe de l'indication des anomalies est le même que celui de l'ordre ATTACH.

'00' : occurrence détruite.
'01' : l'occurrence n'existait pas.
'03' : cfr. ATTACH.
'04' : tentative de violation du paramètre L de la relation ; occurrence non détruite.
'06' : tentative de violation du paramètre I de la relation ; occurrence non détruite.
'09' : cfr. ATTACH.

2.6. Ordre TRANSFER

Cet ordre peut être vu comme une demande, pour chaque relation

spécifiée (transfert partiel) d'une séquence de DETACH élémentaire suivis d'une séquence de DETACH élémentaire suivis de séquence d'ATTACH-élémentaires ; à chacune de ces opérations élémentaires est associée une élément de SYS-ERLIST dans l'ordre où ces opérations sont spécifiées.
SYS-ERROR contient le code maximum repris dans SYS-ERLIST.

'00' : pas d'anomalie.

'01' : une même étiquette références plsuieurs fois dans un transfert partiel.

'02' :
- étiquette relative à la cible non affectée à une réalisation ;
- occurrence à supprimer inexistante, ou occurrence à créer déjà existante ;
- tentative de violation des caractéristiques K ou L pour cette cible.

'03' : tentative de violation des caractéristiques I ou J.

'07' : plus de 20 occurrences distinctes spécifiées dans une liste d'un transfert partiel.

'08' : l'étiquette relative à l'origine n'est pas affectée à une réalisation.

3. ANALYSE DES ERREURS

Le programmeur a à sa disposition un certain nombre de renseignements lui permettant d'analyser les conditions de terminaison d'un ordre DML. Ces renseignements sont rangés dans les zones suivantes, déclarées automatiquement par le compilateur DML dans la zone de communication :

01 SYS-ERROR PIC X(2) : code de retour global.

01 SYS-STATUS PIC X(2) : code numérique de la fonction DML qui vient d'être l'objet d'une anomalie.

01 SYS-MAC-NUM PIC X(4) : n° séquentiel de l'ordre DML qui a été l'objet d'une anomalie.

01 SYS-ERAREA

02 SYS-ERNBR PIC 9(3) COMP. : nombre d'éléments actifs dans le vecteur d'erreurs.

02 SYS-ERLIST PIC X(2) OCCURS... : liste des codes de retour pour les ordres multiples.

o

o o

N.U.L.

LANGAGE D'INTERROGATION CONVERSATIONNEL

1. INTRODUCTION

Le système NUL (Navigational User's Language) supporte un langage conversationnel qui permet de consulter une ou plusieurs bases de données. La connaissance de certaines notions, propres à ce système est indispensable pour pouvoir travailler avec NUL.

a. Session

Une session est constituée de l'ensemble des actions demandées par l'utilisateur au cours d'une même activation du système.

b. Consultation

Une consultation est formée de l'ensemble des messages concernant une même base de données dans une session. Tous les messages envoyés entre l'ouverture et la fermeture d'une base de données appartiennent donc à une même consultation.

Une session peut contenir plusieurs consultations successives.

c. Contexte

Un contexte est une vue partielle de la base de données.

Un contexte est défini par un ensemble de définitions ; chaque définition est associée à un objet complexe.

d. Commande d'impression

Une commande d'impression déclenche des accès à la base de données consultée. Seuls sont accessibles les OC du contexte courant. Cette commande provoque l'impression de valeurs d'OE sélectionnées, sur le terminal et/ou sur un listing, d'une manière standard.

Il est possible d'envoyer successivement plusieurs commandes d'impression portant sur un même contexte.

e. Requête

Une requête est formée d'une définition de contexte et de l'ensemble des commandes d'impression relatives à ce contexte.

Une consultation peut contenir plusieurs requêtes successives.

f. Mode d'ouverture

C'est l'état initial du système ou l'état atteint après la clôture du mode commande par la commande de contrôle CLOSE DB. Dans cet état, l'utilisateur peut ouvrir une base de données ou arrêter la session.

g. Mode contexte

C'est l'état du système pendant lequel l'utilisateur peut envoyer des messages de définition du contexte.

h. Mode commande

C'est l'état du système pendant lequel l'utilisateur peut envoyer des commandes d'impression.

o

o o

2. LE MODE OUVERTURE

Dès que le système NUL est chargé, il envoie à l'utilisateur le message suivant :

ENTER DATE BASE NAME / HALT -

Réponse de l'utilisateur :

- ⟨ROOT-NAME⟩. Ce doit être le nom d'une base de données ayant un schéma interne connu du système. Par cette réponse, l'utilisateur rend disponible la base de données dont la racine a pour nom ⟨ROOT-NAME⟩. Il sera possible de travailler sur cette base entre ce message et le message de fin de consultation CLOSE DB ou le message de fin de session HALT.

- HALT : Cette réponse indique la fin d'une session de travail. Le système est déchargé.

Règle : Deux consultations d'une session ne peuvent se recouvrir ; on ne peut donc travailler que sur une seule base de données à la fois.

o

o o

3. LE MODE CONTEXTE

3.1. Définition de contexte

3.1.1. Syntaxe

$$[\text{FOR Snn } [\{\begin{matrix}\text{BY}\\ \text{VIA}\end{matrix}\}\langle \text{Rel-name1}\rangle]]\langle \text{CO-name1}\rangle [\begin{bmatrix}\text{SUCH THAT}\\ \text{WHERE}\\ \text{WHEN}\\ \text{ST}\\ \text{WH}\end{bmatrix}_1^1 \langle \text{critère}\rangle [\text{AND}\langle \text{critère}\rangle]_0^n];$$

3.1.2. Fonctions générales

Une définition a pour but de sélectionner des réalisations de l'OC de nom <CO-name1>.
Si la clause FOR... est absente, l'accès aux réalisations de ⟨CO-name1⟩ sera fait à partir de la racine de la base.
Si la clause FOR... est présente, l'accès aux réalisations de⟨CO-name1⟩ sera fait à partir des réalisations de l'OC associé à l'étiquette Snn. Le nom ⟨Rel-name1⟩ de la relation à emprunter est spécifié s'il existe.
⟨Co-name1⟩ est éventuellement suivi d'un ou plusieurs critères, simples ou composés. La fonction des critères est de restreindre l'ensemble des réalisations de ⟨CO-name1⟩ accessible aux réalisations qui vérifient les critères ; un critère est introduit par le mot-clé SUCH THAT, WHERE, ...

3.1.3. Règles générales

- ⟨CO-name1⟩ doit être le nom d'un OC de la base consultée ;
- Si dans la clause FOR..., le nom de la relation ⟨Rel-name1⟩ n'est pas spécifié, il faut qu'il existe une relation sans nom de ⟨CO-name1⟩ vers l'OC associé à l'étiquette Snn ;
- La clause FOR... est toujours absente de la première définition d'un contexte ;
- Avant chaque définition le système imprime une étiquette sous la forme Snn⟨- où nn est un nombre de deux chiffres.
  Toute référence à l'OC associé à une définition (⟨CO-name1⟩) sera faite par l'étiquette qui précède la définition.

3.1.4. Règles d'écriture

- Toute définition se termine par ';'.

- L'utilisateur peut envoyer une définition à l'aide de plusieurs messages. Tant que la définition est incomplète, NUL renvoie le message Snn... où Snn est l'étiquette de la définition en cours.

  Exemple :

```
S01 <- CONSTRUCTEUR
S01 ... SUCH THAT NOM
S01 ... = 'SIEMENS'
S01 ... ;
S02 <-
```

o

o o

3.2. Structure d'un contexte

La structure d'un contexte est une arborescence ou un ensemble d'arborescences disjointes.

La racine d'arborescence est un OC (et par extension son étiquette) auquel l'accès est fait à partir de la racine de la base de données. On reconnaît une racine d'arborescence à l'absence de la clause FOR... dans sa définition.

Un contexte définit une hiérarchie d'accès. L'accès à un OC quelconque d'une arborescence n'est possible qu'à partir d'accès aux OC qui se trouvent dans l'arborescence sur la branche qui relie la racine d'arborescence à cet OC.

Remarque : Aucun accès n'est fait au moment de la définition du contexte.

o

o o

3.3. Critères simples

3.3.1. Syntaxe

$$\langle \text{EO-name} \rangle \; [\,\text{NOT}\,] \begin{bmatrix} > \\ < \\ = \end{bmatrix}_1^1 \langle \text{valeur} \rangle$$

3.3.2. Fonction

Un critère simple exprime une condition sur la valeur de l'OE de nom ⟨ EO-name ⟩.
Cet OE doit être relié directement à l'OC sur lequel porte la critère. Un critère porte toujours sur l'OC qui précède l'opérateur ST (ou un synonyme) qui l'a introduit.

Remarque : Un critère simple peut porter directement sur l'OC ⟨ OC-name1 ⟩ associé à la définition en cours. Il peut porter sur un autre OC dans un qualificatif de critère complexe.

3.3.3. Description de ⟨ valeur ⟩

$$\langle \text{valeur} \rangle = \begin{bmatrix} \text{valeur simple} \\ \text{liste de valeurs} \\ \text{valeur tabulée} \\ \text{valeur incomplète} \end{bmatrix}_1^1$$

Valeur simple :

Si l'OE est défini en alphanumérique, la valeur doit être mise entre quotes.
Si l'OE est défini en numérique, la valeur doit être un nombre écrit sans quotes.

Liste de valeur :

C'est un ensemble de valeurs simples séparées par des virgules. Il peut y avoir au maximum 5 valeurs.

Valeur tabulée :

L'OE doit être composé et défini en alphanumérique. L'ensemble /mm indique pour chaque valeur d'OE simple, sa position mm par rapport au début de la valeur de l'OE dont il est composant.
Exemple :

		position
1 ADRESSE		
2 NUMERO	PIC X(4)	1
2 RUE	PIC X(18)	5
2 LOCALITE	PIC X(18)	23

'21'/4'RUE GRANDGAGNAGE'/22'NAMUR'

Valeur incomplète :

Pour un OE défini en alphanumérique, une valeur incomplète est une valeur dont la longueur est inférieure à celle de l'OE.

Exemple :

1 NOM PIC X(8)

Critère simple : ST NOM = 'I'

Signification : Tous les noms commençant par I

o

o o

3.4. Critère complexe

3.4.1. Syntaxe

$$[\ \{\begin{matrix}BY\\VIA\end{matrix}\}\langle Rel\text{-}name2\rangle]\ \left[\begin{matrix}HAVE\\HAS\\IS\\ARE\\WITH\end{matrix}\right]_1^1 [\langle quantificateur\rangle]\ \langle CO\text{-}name2\rangle\ [=Xpp]\ ;$$

3.4.2. Fonction

Un critère complexe exprime une condition sur l'existence et les propriétés de réalisations d'OC ⟨CO-name2⟩ reliées à une réalisation d'OC.
L'objet ⟨CO-name2⟩ doit être relié à l'OC qui précède l'opérateur ST (ou un synonyme) qui a introduit le critère complexe.
Un critère complexe peut porter directement sur l'OC ⟨CO-name1⟩ associé à la définition en cours. Il peut porter sur un autre OC dans un qualificatif de critère complexe.
La clause BY (ou VIA) ⟨Rel-name2⟩ spécifie la relation qui unit les OC. Cette clause est omise si la relation est dans nom.
La clause = Xpp indique que l'on désire qualifier les réalisations de ⟨CO-name2⟩.

3.4.3. Description de ⟨quantificateur⟩

$$\langle quantificateur\rangle = \left[\begin{matrix}SOME\\NO\\ALL\\AT\ LEAST\ N\\AT\ MOST\ N\end{matrix}\right]_1^1$$

- Un quantificateur modifie la signification d'un critère complexe.
- Une quantificateur exprime une condition sur le nombre de réalisations de l'objet ⟨CO-name2⟩ vérifiant l'éventuelle qualification qui doivent être reliées par la relation ⟨Rel-name2⟩ à chaque réalisation de l'OC auquel est attaché ce critère pour que cette dernière réalisation soit retenue.
- N est un nombre représentant un nombre entier positif.
- *AT LEAST N* signifie que la réalisation est retenue si *au moins N* réalisations de l'objet ⟨CO-nombre2⟩ (vérifiant l'éventuel qualificatif [= Xpp]) lui sont reliées par la relation ⟨rel-name2⟩.
- *AT MOST N* correspond à *au plus N*.

- *ALL* indique que *toutes* les réalisations reliées doivent vérifier le qualificatif.
- *NO* indique qu'*aucune* réalisation reliée ne peut vérifier le qualificatif.
- *SOME* est équivalent à AT LEAST 1.
- En l'absence du qualificateur, *SOME* et *ALL* sont équivalents.

<u>Exemples</u> :

a. S01 <- CONSTRUCTEUR ST VIA DISTRIBUE HAS SOME CPU ; constructeurs distribuant au moins 1 CPU.

b. S01 <- MACHINE SUCH THAT HAVE AT LEAST 2 TERMINAL = X01 ; machines comportant au moins 2 terminaux vérifiant le qualificatif X01 que l'on exprimera dans la suite.

o

o o

3.5. Qualificatif de critère complexe

Un qualificatif est un critère simple ou complexe relatif à des réalisations d'un OC spécifié dans un critère complexe. Par extension, on appelle aussi qualificatif,l'étiquette Xpp qui introduit le qualificatif dans le critère complexe.

La clause =Xpp terminant un critère complexe signifie que l'utilisateur désire imposer un ou plusieurs critères portant sur ⟨CO-name2⟩.

Dans ce cas, le système imprime l'étiquette pour que l'utilisateur puisse expliciter le qualificatif

$$\underbrace{\text{Xpp} \langle\text{-}}_{\text{imprimé par NUL}} \left[\begin{array}{l}\text{SUCH THAT}\\ \text{WHERE}\\ \text{WHEN}\\ \text{ST}\\ \text{WH}\end{array}\right]_1^1 \langle\text{critère}\rangle\ [\text{AND}\ \langle\text{critère}\rangle]_0^n\ ;$$

les règles de définition de ⟨critère⟩ sont identiques à celles spécifiées dans les parties correspondantes des paragraphes concernant les critères simples et les critères complexes.

Remarque : Les qualificatifs introduisent une notion de récursivité. Un critère complexe peut contenir un autre critère complexe.

Exemple :

1. S01 ⟨- INSTALLATION ST VIA COMPORTE HAVE MACHINE = X01 ;
   X01 ⟨- ST HAVE AT LEAST 3 U-DISQUE = X02 ;
   X02 ⟨- ST CAPACITE > = 200.000 ;

   Cette expression désigne les installations comportant au moins une machine dotée d'au moins 3 unités à disques d'une capacité de 200 Mb ou plus.

2. S01 ⟨- SYSTEME-EXPLOIT ST VIA SUPPORTE HAVE SGBD = X01 AND VIA SUPPORTE ;
   S01... HAVE LANGAGE = X02 ;
   X01 ⟨- ST NORME = 'CODASYL' ;
   X02 ⟨- ST NOM = 'COBOL-74' ;

   Cette expression désigne les systèmes d'exploitation disposant d'un SGBD de norme CODASYL et d'un compilateur COBOL 74.

4. LE MODE COMMANDE (COMMANDE D'IMPRESSION)

4.1. Syntaxe

PRINT ⟨liste d'étiquettes⟩

4.2. Fonction

Une commande PRINT va provoquer l'impression, sur le terminal et/ou sur un listing, de manière standard, de certaines valeurs d'OE reliées directement aux réalisations d'OC définies dans le contexte.

Les référence aux OC du contexte se font par l'intermédiaire de ⟨liste d'étiquettes⟩. Chaque étiquette de la liste fait référence à l'OC qui lui a été associé au moment de la définition du contexte.
⟨liste d'étiquettes⟩ permet de :
- sélectionner des valeurs d'OE,
- choisir une séquence d'impression.

4.3. Règle

Une commande ne peut faire référence qu'à des OC d'une seule arborescence du contexte.

4.4. Sélection des valeurs d'OE

- Si l'utilisateur veut obtenir les valeurs de tous les OE d'un OC, il doit simplement indiquer l'étiquette associée à l'OC dans le contexte ;
- Si l'utilisateur ne veut obtenir les valeurs que d'un seul OE, il spécifie l'étiquette concaténée au nom de l'OE. S'il donne les valeurs de plusieurs OE, il spécifie une liste dont chaque élément est l'étiquette concaténée avec le nom d'un des OE ; deux éléments sont séparés par un virgule.

Exemple :

```
SO1 <- CONSTRUCTEUR
PRINT   SO1
PRINT   SO1.NOM
PRINT   SO1.NOM, SO1.SIGLE
```

o

o o

4.5. Choix de la séquence d'impression

L'ordre (que nous appellerons arborescence d'impression) suivant lequel les valeurs d'OE des OC vont être imprimées, n'est pas forcément conforme à l'ordre arborescent décrit dans la définition du contexte (ordre que nous appellerons arborescence du contexte) et qui correspondra précisément aux accès qui seront réalisés.

La racine d'impression est le premier OC (et par extension la première étiquette) citée dans ‹liste d'étiquettes›. La racine peut être un OC quelconque d'une arborescence de contexte.

A partir de cet OC, deux directions sont possibles :

a. Descente dans l'arborescence du contexte :

Cela revient à respecter la hiérarchie du contexte. Les valeurs d'OE d'un OC sont imprimées APRES celles des OC de niveau supérieur dans l'arborescence du contexte et qui ont permis l'accès.

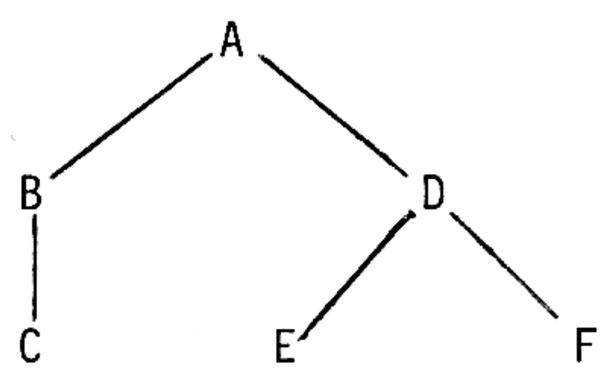
b. Remontée dans l'arborescence du contexte :

Cela revient à inverser la hiérarchie du contexte. Les valeurs d'OE d'un OC sont imprimées AVANT celles des OC de niveau supérieur dans l'arborescence du contexte et qui ont permis l'accès.

Remarques : Autant à la descente qu'à la remontée, il est possible de *sauter* des niveaux. Le niveau supérieur n'est donc pas toujours le niveau immédiatement supérieur.
Il est possible d'écrire une commande PRINT ne contenant qu'une seule étiquette (la racine) et donc sans effectuer ni descente ni remontée dans l'arborescence.

Les exemples se réfèrent au contexte suivant :

```
S01 <- A
S02 <- FOR S01 B
S03 <- FOR S02 C
S04 <- FOR S01 D
S05 <- FOR S04 E
S06 <- FOR S04 F
S07 <- END
```



4.5.1. Syntaxe

- La notion de descente s'exprime par des parenthèses :
  Exemple : PRINT SO2 (SO3)
  Signification : chaque réalisation de B (SO2) est suivie de toutes les réalisations de C (SO3) qui lui sont reliées.

- La notion de remontée s'exprime par des virgules :
  Exemple : PRINT SO5, SO4
  Signification : après chaque réalisation de E (SO5), est imprimée la réalisation de D (SO4) qui a permis d'y accéder.

4.5.2. Règles

- Il peut y avoir autant de parenthèses imbriquées qu'il y a de descentes sucessives dans l'arborescence.
  Exemple : PRINT SO1 (SO2(SO3)).
  Signification : chaque réalisation de A (SO1) est suivie par toutes les réalisations de B (SO2) qui lui sont reliées, chaque réalisation de B étant elle-même suivie de toutes les réalisations de C (SO3) qui lui sont reliées.

- A l'intérieur des parenthèses, les virgules n'expriment pas la notion de remontée. Elles permettent de citer un ensemble d'étiquettes qui doivent être celles d'OC appartenant à la descendance de l'OC dont l'étiquette précède la parenthèse.
  Exemple : PRINT SO1 (SO6, SO4).
  Signification : après chaque réalisation de A (SO1), sont imprimées les réalisations de F (SO6) puis celles de D (SO4) qui lui sont reliées.

4.5.3. Noeud de bifurcation

C'est un noeud de l'arborescence cité au cours d'une remontée et qui est le point de départ d'une descente.

Exemples : PRINT SO6, SO4 (SO5),
PRINT SO5, SO1 (SO2, SO3),

SO4 et SO1 sont des noeuds de bifurcation.

Règles :

Il est possible de descendre après être remonté ;

Il n'est pas possible de remonter après être redescendu, sauf à partir de la racine d'impression. Dans ce cas, il est possible de descendre

dans l'arborescence en citant à l'intérieur d'une parenthèse une ou plusieurs étiquettes, puis de remonter dans l'arborescence à partir de la racine d'impression.
Exemple : PRINT S04 (S05, S06), S01.

4.6. Formats d'impression

Après chaque commande PRINT, le système envoie à l'utilisateur le message suivant pour lui demander de choisir un format :

FORMAT OF NUL REPORT : 01/02/03 -

L'utilisateur indique alors le numéro du format retenu. Les trois formats listent une valeur par ligne.
Le format 01 liste toutes les valeurs associées à chaque réalisation de la racine d'impression, sans rupture ni décalage. Le niveau d'imbrication dans la hiérarchie d'impression, le numéro de l'étiquette concaténés au nom de l'objet élémentaire précédent la valeur sur chaque ligne.
Le format 02 liste toutes les valeurs sans rupture mais avec des décalages qui réflètent la hiérarchie d'impression. L'étiquette concaténée au nom de l'objet élémentaire précède la valeur sur chaque ligne.
Le format 03 liste toutes les valeurs avec ruptures et décalages qui réflètent la hiérachie d'impression. Seule la valeur de l'objet élémentaire apparaît sur une ligne.

L'introduction du numéro de format déclenche l'impression des valeurs.

o

o o

5. TRAITEMENT DES ERREURS

Le traitement des erreurs diffère suivant le mode.

5.1. Mode ouverture

Si le nom de la base de données est erroné, NUL renvoie un message d'erreurs puis le message

ENTER DATA BASE NAME / HALT -

5.2. Mode contexte

NUL réimprime la définition envoyée, positionne un marqueur '?' en face du symbole erroné et renvoie un message pour expliciter la nature de l'erreur. NUL imprime la partie de la définition considérée correcte pour indiquer à l'utilisateur l'endroit à partir duquel il doit compléter sa définition.

5.3. Mode commande

NUL renvoie un message pour expliciter la nature de l'erreur. La commande est ignorée et l'utilisateur doit introduire une nouvelle commande.

Remarque : Tant en mode contexte qu'en mode commande, NUL n'effectue aucune vérification tant que la définition ou la commande n'est pas clôturée par un point-virgule.

5.4. Messages d'erreur

WARNING - INVALID LABEL
WARNING - COMPLEX OBJECT NOT FOUND
WARNING - RELATION NOT FOUND
WARNING - ELEMENTARY OBJECT NOT FOUND
WARNING - COMPARISON OPERATOR MISSING
WARNING - ATTRIBUTE VALUE NOT ENCLOSED BETWEEN QUOTES
WARNING - ATTRIBUTE VALUE NUMERIC
WARNING - INVALID ELEMENTARY OBJECT TYPE
WARNING - ELEMENTARY OBJECT VALUE TOO LONG
WARNING - SEPARATOR MISSING
WARNING - INVALID QUANTIFIER

WARNING - INVALID QUALIFICATION FLAG
WARNING - TOO MANY QUALIFICATIONS
WARNING - VALUE OVERFLOW
WARNING - CONTEXT TABLE OVERFLOW
WARNING - X-ENTRY MAY NOT CONTAIN AN END COMMAND
WARNING - TABULATION ERROR
WARNING - INVALID COMPARISON OPERATOR WITH LIST OF VALUE
WARNING - INVALID COMMAND
WARNING _ INVALID LABEL
WARNING - LABEL IS NOT DECLARED
WARNING - PRINT COMMAND DOES NOT CONCERN A UNIQUE HIERARCHY
WARNING - INVALID CONCATENATION OF LABEL
WARNING - INVALID ELEMENTARY OBJECT
WARNING - SEPARATOR MISSING
WARNING - BRACKET IS MISSING OR INVALID
WARNING - TOO MANY LEVELS IN PRINT HIERARCHY
WARNING - TOO MANY ARGUMENTS IN PRINT COMMAND
WARNING - PRINT ROOT SPECIFIED TWO TIMES.

o

o o

6. LES COMMANDES DE CONTROLE

Les commandes de contrôle modifient le mode courant ou permettent d'annuler le dernier message envoyé.

HALT : Cette commande peut être utilisée dans les trois modes. Elle clôture la session et assure la fermeture de la base éventuellement ouverte.

END : Cette commande permet de passer du mode contexte en mode commande et inversément.

CLOSE BD :Cette commande ne peut être utilisée qu'en mode commande. Elle assure la fermeture de la base de données courante et repositionne le système en mode ouverture.

: Cette commande peut être utilisée en mode contexte ou en mode commande. Elle annule le message courant et ne modifie pas le mode.

o

o o

7. LES FONCTIONS DE CONTROLE DU SYSTEME

Les fonctions de contrôle du système sont des fonctions annexes qui donnent à l'utilisateur la possibilité de modifier certains paramètres opératoires ou qui le renseignent sur les caractéristiques d'OC ou de relations de la base sur laquelle il travaille.

Pour accéder à ces fonctions, il faut interrompre le système (retour en mode BS2000 : "/") puis envoyer la commande 'INTR'. L'interruption du système n'est possible qu'en mode contexte ou en mode commande.

Les fonctions de contrôle sont les suivantes :

HELP : liste toutes les fonctions de contrôle disponible.

SYNTAX : donne les règles de syntaxe d'une définition de contexte et d'une commande d'impression.

LINE ? : donne la longueur de la ligne d'impression.

LINE = num : modifie la longueur de la ligne d'impression, le nombre doit être compris entre 52 et 130.

OUTPUT ? : donne le milieu de sortie de l'impression.

OUTPUT = BATCH ou STANDARD.
BATCH supprime l'édition sur terminal,
STANDARD restitue cette édition.

USER ? : donne le TSN de l'utilisateur NUL.

DBNAME ? : donne le nom de la base de données ouverte.

ENTITY ? : donne le nom de tous les OC de la base.

ENTITY =<OC name> : liste pour l'OC les OE qui lui sont associés et les relations dont il est origine.

RELATION ? : donne le nom de toutes les relations de la base.

RELATION =<relation name> : liste toutes les relations qui portent le nom cité, de même que leur origine et leur cible.

RELATION/ORIGINE =<OC name>: liste toutes les relations dont l'OC<OC name> est origine.

CONTEXT ? : liste toutes les définitions du contexte.

CONTEXT = SNN : liste la définition dont l'étiquette est SNN.

Remarque

Aucune trace de l'appel à ces fonctions de contrôle n'est conservée.

8. DESCRIPTION DE LA SYNTAXE DE NUL A L'AIDE DU FORMALISME BNF

Les symboles suivants ne sont pas définis par des règles de grammaire et leur signification est donnée ci-dessous :

⟨empty⟩ désigne une chaîne vide de caractères.

⟨name⟩ désigne un nom.

⟨string⟩ désigne n'importe quelle suite de caractères imprimables ou de blancs.

⟨digit⟩ désigne un chiffre.

⟨integer⟩ désigne une suite finie de chiffres.

o

o o

```
⟨session⟩        ::= ⟨consultation⟩ | ⟨consultation⟩ ⟨session⟩
⟨consultation⟩   ::= ⟨requête⟩ | ⟨requête⟩ ⟨consultation⟩
⟨requête⟩        ::= ⟨contexte⟩ ⟨impression⟩
⟨contexte⟩       ::= ⟨définition⟩ | ⟨définition⟩ ⟨contexte⟩
⟨définition⟩     ::= ⟨Référence⟩ ⟨Accès⟩ ⟨CO-name⟩ ⟨Critère⟩ ;
⟨Référence⟩      ::= FOR ⟨label⟩ | ⟨empty⟩
⟨Label⟩          ::= S⟨NN⟩
⟨NN⟩             ::= ⟨digit⟩ ⟨digit⟩
⟨Accès⟩          ::= BY ⟨Rel-name⟩ | VIA ⟨Rel-name⟩ | ⟨empty⟩
⟨Rel-name⟩       ::= ⟨name⟩
⟨CO-name⟩        ::= ⟨name⟩
⟨Critère⟩        ::= ⟨opérateur-1⟩ ⟨Crit-1⟩ | ⟨empty⟩
⟨opérateur-1⟩    ::= SUCH THAT | WHERE | WHEN | ST | WH
⟨Crit-1⟩         ::= ⟨Crit-2⟩ | ⟨Crit-2⟩ AND ⟨Crit-1⟩
⟨Crit-2⟩         ::= ⟨Crit-simple⟩ | ⟨Crit-complexe⟩
⟨Crit-simple⟩    ::= ⟨OE-name⟩ ⟨operateur-2⟩ ⟨valeur⟩
⟨OE-name⟩        ::= ⟨name⟩
⟨opérateur-2⟩    ::= ⟨opérateur-3⟩ | NOT ⟨opérateur-3⟩
⟨opérateur-3⟩    ::= ⟨ | ⟩ | =
⟨valeur⟩         ::= ⟨integer⟩ | '⟨string⟩'
⟨Crit-complexe⟩  ::= ⟨Accès⟩ ⟨Verbe⟩ ⟨Quantif⟩ ⟨Co-name⟩ ⟨Qualif⟩
⟨Verbe⟩          ::= HAVE | HAS | IS | ARE | WITH
⟨Quantif⟩        ::= NO | SOME | ALL | AT LEAST ⟨integer⟩ | AT MOST ⟨integer⟩|⟨empty⟩
⟨Qualif⟩         ::= = X⟨NN⟩ | ⟨empty⟩
⟨impression⟩     ::= PRINT ⟨liste-1⟩ ; | PRINT ⟨liste-1⟩ ; ⟨impression⟩
⟨liste-1⟩        ::= ⟨label⟩ ⟨liste-2⟩ | ⟨label⟩
⟨liste-2⟩        ::= (⟨descente-1⟩) | , ⟨Remontée-1⟩ | (⟨descente-1⟩), ⟨Remontée-1⟩|,
                     ⟨Remontée-1⟩ | (⟨Descente-1⟩)
⟨Descente-1⟩     ::= ⟨label⟩ | ⟨label⟩ (⟨Descente-1⟩) |⟨label⟩, ⟨Descente-1⟩
⟨Remontée-1⟩     ::= ⟨label⟩ | ⟨label⟩, ⟨Remontée-1⟩
```

o

o o