THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Comparaison d'un SGBD CODASYL et d'un SGBD RELATIONNEL

Wu, Suchun

Award date:
1988

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Année académique 1987-1988

Comparaison d'un SGBD CODASYL et d'un SGBD RELATIONNEL

WU SUCHUN

Promoteur: Monsieur le Professeur J-L HAINAUT

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

Résumé:

Les SGBD CODASYL et les SGBD RELATIONNEL sont les plus largement utilisés parmi les SGBD commercialisés. Ce mémoire essaye de faire une comparaison entre un SGBD CODASYL (DBMS) et un SGBD RELATIONNEL (RDB), qui sont implémentés sur le VAX/8600 de Digital.

Afin d'éclairer le choix entre deux SGBD, la comparaison se fait sur les deux aspects. Premièrement, l'aspect logique qui concerne la puissance du modèle de données, l'indépendance de données et le degré algorithmique des programmes. Deuxièmement, l'aspect technique qui est directement lié à l'évaluation de la performance.

Le mémoire établit les critères sur lesquels la comparaison sera effectuée. A cet effet, il définit une typologie des applications et les règles de production de bases de données et de programmes équivalents. Le mémoire conclut sur une comparaison des résultats obtenus pour les deux SGBD.

Abstract:

The CODASYL and the relational database management systems are widely used among those commercialized. This dissertation attempts to make a comparison between one CODASYL system (DBMS) and one relational system (RDB), which are implemented on the VAX/8600 of Digital.

In order to bring a choice between two systems to light, we pursue our comparison in two aspects. First, the logical aspect that is concerned about the power of the data model, the data independence, and the algorithmic degree of the programs. Second, the technical aspect which is directly linked with the evaluation of the performance.

This dissertation establishes the criteria with which the comparison is carried out. To that effect, it defines a typology of the applications and the rules of production of the databases and of equivalent programs. This work concludes with a comparison of the results obtained for the two database management systems.

REMERCIEMENTS

Nous tenons tout d'abord à exprimer notre reconnaissance à monsieur le professeur J-L HAINAUT qui a accepté de diriger ce mémoire et nous a permis par ses conseils constructifs et critiques pertinentes, de mener ce travail à terme.

Nous remercions également mademoiselle Françoise DUBISY et monsieur Laurent GOCHE pour l'aide apportée à la relecture de notre travail.

Nous exprimons notre gratitude à tout le personnel du Centre de Calcul de l'Institut d'Informatique pour leur patience et leur disponibilité.

Nous tenons également à remercier la compagnie DIGITAL pour les informations techniques qu'elle a bien voulu nous communiquer. Les mesures réalisées et les conclusions sont cependant de notre seule responsabilité, DIGITAL n'étant en rien responsable des inexactitudes qui auraient pu se produire.

TABLE DE MATIERES

Deuxième partie: L'ASPECT TECHNIQUE

Première partie: L'ASPECT LOGIQUE

C) Détermination des paramètres physiques des BD
1) Evaluation du volume de la BD/DBMS
2) Paramètres de la création de la BD/DBMS
3) Evaluation du volume de BD/RDB

ANNEXE-3 Algorithmes conformes aux SGBD;

ANNEXE-4 Programmes des applications.

A) Programmes/DBMS/COB

B) Programmes/RDB/COB

CHAPITRE 1
INTRODUCTION

1.1 OBJECTIF DU MEMOIRE

Les organisations conçoivent et utilisent un système d'information pour satisfaire leurs besoins organisationnels. Il est clair que le Système de Gestion de Base de Données (SGBD) joue un rôle central au sein du système d'information. Il y a maintenant beaucoup de SGBD commercialisés et chacun d'eux offre des caractéristiques spécifiques. Les SGBD CODASYL et SGBD RELATIONNEL sont parmi les plus largement utilisés dans le monde. Un utilisateur qui envisage le choix d'un SGBD voudrait souvent estimer, non seulement qualitativement, mais aussi quantitativement le système par rapport à son propre environnement. Nous croyons qu'une "bonne" méthode pour mesurer objectivement le SGBD en fonction des applications de l'utilisateur de la base de données est indispensable pour eclairer ce choix.

Une base de données (BD), qu'elle soit relationnelle ou non, est conçue pour permettre la mise en oeuvre d' une grande variété d'applications, tandis que le SGBD facilite l'interaction entre l'utilisateur et la BD.

L'objectif de ce mémoire est de comparer un SGBD RELATIONNEL et un SGBD CODASYL en fonction de critères qualitatifs et quantitatifs. Cette comparaison sera réalisée dans des conditions semblables: même schéma de la BD, même données de la BD, mêmes applications.

Pour conduire cette expérience nous serons amenés à définir une méthode de test. Celle-ci peut être généralisée pour d'autres types de comparaison.

1.2. ETAT DE L'ART

D'une part, il est difficile de comparer directement les deux types de SGBD en vertu de leur structure et de leur fonction, la philosophie de leur conception étant différente. Le SGBD CODASYL est conçu pour supporter la programmation d'application. Quant au SGBD RELATIONNEL, il est conçu au départ dans le but de supporter surtout une utilisation interactive [HAIN 85]. D'autre part, les deux SGBD ont des modèles de données qui se rapportent à différents niveaux de représentation des informations de BD.

Les défenseurs du modèle relationnel voudraient placer la comparaison au niveau de l'utilisateur, tandis que les défenseurs du modèle réseau préfèrent mener la comparaison au niveau du schéma logique et du travail d'un DBA (Data Base Administrator) [BRAC 75].

L'état de l'art, en ce qui concerne la comparaison des SGBD, peut être divisé en deux catégories: la comparaison qualitative et la comparaison quantitative.

1.2.1 Comparaison qualitative

La comparaison qualitative se situe aux niveaux logique, externe et interne. En général, celle-ci se limite à une ou deux caractéristiques du SGBD. Ce peut être, par exemple, l'indépendance logique et physique. Ce genre de comparaison peut se trouver dans plusieurs références, notamment dans [DATE 81], [STON 75], [MICH 76], [NIJS 74].

D'après les comparaisons présentées dans ces articles, la conclusion générale est que l'utilisation du SGBD RELATIONNEL est plus facile. Il offre plus d'indépendance logique et physique. De plus, la programmation d'application en SGBD RELATIONNEL est plus aisée parce que celui-ci offre un langage de haut niveau. Quant au SGBD CODASYL, il offre d'avantage de l'efficacité.

a) Dans le [DATE 81], l'auteur essaye de montrer que le modèle de données relationnel l'emporte sur le modèle de données CODASYL au niveau conceptuel:

ERRATA

Nous nous excusons auprès des lecteurs de ce mémoire pour les différentes erreurs de dactylographie qu'ils pourraient rencontrer. En vue de faciliter la compréhension du lecteur, nous dressons une liste des erreurs. Dans celle-ci, nous reprenons chaque fois une partie de la phrase où l'erreur se situe et soulignons le ou les mots erronés tels qu'ils doivent être lus. (P: page et L: ligne de la page.)

P.24 L.4: .... il est défini comme un intervalle ....

P.38 (l'item DATE-DEBUT d'EMPRUNT n'est pas une clé d'accès).

P.48 L.3 .... Idem que l'application 10.

P.55 L.19 .... en mode interactif dans un environnement....

P.56 Dans le TAB-4.1: ITEMS NUMAU TYPE IS SIGNED LONGWORD.

P.59 Dans le TAB-4.5: ITEMS NH TYPE IS SIGNED LONGWORD.

P.62 L.12 Note: ... un identifiant doit être defini avec la clause....

P.66 ... et de 5 pour le PRG-4.2.

P.79 ... TAB-4.17 Volume des fichiers ....

P.79 ... TAB-4.18 Volume et taux de remplissage ....

P.86 L.6 ... après la création de la nouvelle BD.

P.87 L.6 Au vu des programmes PRG-4.1 et PRG-4.2, ...

P.90 L.3 ..., le DML a d'ailleurs le même but....

P.91 L.20 ... pas adéquat de définir (nous avons défini) une clé....

P.95 L.8 ... que l'accès par chemin qui a un pointeur direct.

P.105 L.23 ..., la plupart des types d'application que nous avons définis....

P.103 L.3 ..., nous devons évaluer chaque SGBD ....

A2-1 L.22 ... (voir plus loin).

A2-10 L.18 1) EVALUATION DU VOLUME DE LA BD/DBMS.

ADDENDUM (1) - SECTION 4.5

Si l'on observe attentivement les tableaux de la section 4.5, on pourrait se poser la question suivante:

Pourquoi la BD/RDB occupe beaucoup plus de place disque que celle de DBMS, bien que l'on ait chargé les mêmes données ? Plus curieux, c'est que le taux de remplissage pour la BD/RDB est plus élevé que celui de DBMS.

Pour ce fait, il nous manque les informations à propos de la représentation interne des données et de la façon de calculer statistique identique ou non à celle de DBMS. Nous avons discuté avec la compagnie DIGITAL, et elle nous a conseillé de faire les mesures après les opérations "BACKUP" et "RESTORE".

Nous avons refait les mêmes mesures décrites dans la section 4.5. Après ces opérations, nous obtenons les résultats repris dans les tableaux suivants.

type d'article	occurrences d'article	DBMS (bytes)	RDB (bytes)
AUTEUR	501	32946	14529
OUVRAGE	1001	105439	33033
OUVAUT	1950	55848	25350
MCOUV	1950	109391	3900
EXEMPLAIRE	2100	75897	39900
EMPRUNTEUR	100	9054	2900
EMPRUNT	81	2590	1377
ENPRUNT-ARCH	280	10314	5880

TAB-4.19 Volume des fichiers des BD de taille réduite

	DBMS		RDB	
AREA	volume(pages)	taux de remp.	volume(pages)	taux de remp.
FAUTEUR	200	57%	1134	81%
FOUV	300	79%		
FEXEM	200	53%		
FEMPR	200	13%		
TOTAL	900	50,5%		

TAB-4.20 Volume et taux de remplissage des BD de taille réduite

type d'article	occurrences d'article	DBMS (bytes)	RDB (bytes)
AUTEUR	3000	251235	92944
OUVRAGE	10000	761208	350034
OUVAUT	14650	441715	249029
MCOUV	14650	856483	322300
EXEMPLAIRE	25860	1004497	595256
EMPRUNTEUR	300	26479	9198
EMPRUNT	200	7063	4221
ENPRUNT-ARCH	2000	78516	49922

TAB-4.21 Volume des fichiers des BD de grande taille

	DBMS		RDB	
AREA	volume(pages)	taux de remp.	volume(pages)	taux de remp.
FAUTEUR	900	88 %	7323	83 %
FOUV	2100	91%		
FEXEM	1600	89%		
FEMPR	400	32%		
TOTAL	5000	**85,1%**		

TAB-4.22 Volume et taux de remplissage des BD de grande taille

Au vu de ces tableaux, nous trouvons que les résultats pour la BD/DBMS restent les mêmes. Quant à la BD/RDB, après "BACKUP" et "RESTORE", le SGBD a condensé certaines données de gestion, mais pas les données effectives (voir les tableaux concernat le volume des fichiers de la BD). Nous remarquons que le résultat pour la BD de taille réduite est plus intéressant que celui de grande taille.

Le résultat comparatif reste finalement comme suit:

la BD/DBMS occupe moins de place disque que celle de RDB. Par conséquent, la conclusion dans la section 5.5 est valable.

Page 78 : remplacer dans le deuxième tableau, 50,2 % par 53,4 %

page 79 : remplacer dans le deuxième tableau, 77,5 % par 85,1 %

ADDENDUM (2) - SECTION 6.2

critère de domaine	situation	SGBD convenable
STRUCTURE DE DONNEES	schéma simple et évolutif	RDB
	schéma complexe et stable	DBMS
	volume de données faible	RDB
	volume de données élevé	DBMS
OPERATION	peu de modifications	RDB
	modifications nombreuses	DBMS
	accès ponctuel	DBMS
	accès de masse	RDB
OBJET	élémentaire	RDB
	complexe	DBMS
PERFORMANCE	peu importante	RDB
	importante	DBMS
APPLICATION	prévisible (stable)	DBMS
	non prévisible (évolutive)	RDB

TAB-6.1 Domaines d'application de chaque SGBD

1) Comparaison quant à la simplicité: l'approche relationnelle n'a qu'une structure de données essentielle: la relation (ou tableau) elle-même. Toutes les informations de la BD sont représentées sous cette forme unique. Le concept de tableau est aussi familier pour l'homme que le concept décimal en mathématique qui est plus simple que le concept binaire (bien qu'il soit minimal).

Il est facile de manipuler la relation. On a essentiellement besoin d'un seul opérateur pour chacune des fonctions fondamentales de manipulation des données: consultation, ajoute, suppression et mise à jour.

Dans l'approche CODASYL on retrouve tous les opérateurs de l'approche relationnelle et de plus, on a le CONNECT pour créer un set et DISCONNECT ou ERASE pour détruire un set.

2) Comparaison quant à la théorie de base: une théorie de base solide est nécessaire pour que les activités du système puissent être accordées aux prévisions intuitives des utilisateurs en ayant une plus grande extension possible. L'approche relationnelle se base sur certains aspects de la théorie mathématique des ensembles. La théorie de la normalisation offre un guide rigoureux pour concevoir le schéma relationnel.

Par contre, il n'y a pas de théorie pour aider les concepteurs à établir le schéma dans l'approche réseau.

b) Dans [EARN 75], l'auteur est arrivé à des conclusions qui sont contraires à celle de [DATE 81]:

1) Les deux modèles sont pratiquement équivalents;

2) La structure relationnelle semble plus simple que la structure réseau. Mais:

3) Le prix à payer pour 2) est une structure réseau plus puissante et qui offre plus d'indépendance que la structure relationnelle; elle va probablement devenir un standard industriel.

c) L'auteur de [MICH 75] a établi des comparaisons en traitant quelques exemples algorithmiques, sur les aspects suivants:

1) la représentativité des modèles de données: deux modèles sont sensiblement les mêmes;

2) l'efficacité de codage: un langage non-procédural aboutit à une économie du code ressource des programmes;

3) l'efficacité de la machine: dans certaines situations, le système relationnel peut probablement subir une dégradation de performance par comparaison avec le système CODASYL. Selon l'hypothèse de l'auteur, le système CODASYL a pour lui l'avantage de la performance.

1.2.2 Comparaison quantitative

La comparaison quantitative se situe au niveau technique, surtout la comparaison de performance par exemple, entre les SGBD de types différents. Ce type de comparaison manque dans la littérature.

Une raison probable à cela est, à notre avis, liée au problème de la portabilité des modèles de mesures sur les SGBD différents. On a pu effectivement élaborer non seulement des modèles analytiques, mais aussi des méthodes expérimentales pour mesurer quantitativement un SGBD RELATIONNEL ou un SGBD CODASYL déterminé [WONG 81], [HAWT 79] [BORA 84].

Ces modèles ou méthodes sont-t-ils transposables pour d'autres SGBD et en particulier pour des SGBD de types différents en considérant leur structure de données physique et leur mécanisme d'implémentation?

a) Le modèle IPSS [WONG 81], par exemple, permet aux concepteurs du système de réaliser leurs objectifs d'évaluation d'un système d'information en utilisant une série de transformations:

- transformation au niveau du système d'information;
- transformation au niveau des applications;
- transformation au niveau du SGBD;
- transformation au niveau de la gestion des fichiers;
- transformation au niveau du stockage physique.

Ce modèle est applicable, d'après l'auteur, aux SGBD relationnels et aux Systèmes de Gestion de Fichiers (SGF).

b) Les chercheurs du laboratoire d'éléctronique de l'université de California ont évalué la performance du SGBD relationnel INGRE [HAWT 79]. Cette évaluation se basait sur trois benchmarks:

1) requête en "data-intensive";

2) requête en "overhead-intensive";

3) requête en "multi-relation".

Les deux premiers benchmarks se distinguent en termes de temps consommé par une requête. Une requête en "overhead-intensive" consomme plus de temps CPU par rapport au temps d'accès aux données de la BD.

Nous remarquons que la définition ci-dessus est très intuitive. Il est très difficile de garantir que les benchmarks seront indépendants des systèmes que l'on va évaluer. Par ailleur, le troisième benchmark ne convient pas pour les SGBD CODASYL parce qu'il n'y a aucun concept de "multi-relation".

L'autre raison probable de la rareté des références est un problème de contrôle des conditions de comparaison.

La comparaison n'est pas une tâche simple. Tout le monde s'accorde pour dire qu'il faut bien contrôler les conditions et l'environnement de la comparaison, afin que celle-ci puisse avoir un sens.

Lorsqu'un SGBD diffère d'un autre quant à ses modèles de données, il faut qu'étant donné un système d'information, on puisse en premier lieu le modéliser en deux modèles différents de manière identique. Cela exige de l'utilisateur qu'il trouve un moyen ou un modèle qui constitue un référentiel commun qui jouerait le rôle d'un interface entre le système d'information et les modèles de données pour réaliser la transformation identique.

Nous verrons que le Modèle d'Accès Général (MAG) [HAIN 86] va jouer ce rôle intéressant.

1.3. ENVIRONNEMENT EXPERIMENTAL

1.3.1 Le materiel

Nous avons travaillé sur un VAX/8600. Il contient 16 mégabytes de mémoire centrale avec un FPU (Floating Point Unit) et un contrôleur de disques HSC50.

Les bases de données expérimentales sont établies sur le disque RA81 qui a une capacité de 456 mégabytes, un taux de transfert 17,4 megabytes/sec, et un temps moyen de positionnement du bras de 28 msec, et un temps moyen de delai de rotation de 8.3 msec.

1.3.2 Le logiciel

1) VAX/VMS

Le système d'exploitation est un système à mémoire virtuelle. Il supporte le développement et l' exécution de programmes en time-sharing , en batch et en temps-réel. La version est actuellement 4.6.

2) VAX-11 DBMS version 2.4

Le système de gestion de base de données est sous le contôle de VAX/VMS. Il est compatible avec CODASYL et implanté sur le VAX de Digital. Il répond à la norme ANSI DATA DEFINITION LANGUAGE COMMITEE (1981).

Le DBMS est composé de plusieurs parties:

- VAX-11 DBMS DDL (DATA DEFINITION LANGAGE) compilateur;

- le VAX CDD (COMMON DATA DIRECTORY) qui est un dépôt central des définitions logiques de structure de données compilées, y compris SCHEMA (la base de données elle-même); SUBSCHEMA, STORAGE-SCHEMA et SECURITY-SCHEMA;

- le DBCS (Data Base Control System) qui joue le rôle d'interface entre VAX DBMS et VAX/VMS, et les programmes d'application;

- l'U.W.A qui est une zone de communication entre DBMS et les programmes d'application;

- le DBO (Data Base Operator) qui fournit les utilitaires pour créer la BD, charger la BD, et consulter les informations logiques dans CDD.

Le DBMS permet :

- la définition logique des données et leurs relations;

- une méthode pour stocker les valeurs de données;

- diverses vues utilisateurs des données;

- une utilisation simultanée de la base de données par plusieurs utilisateurs;

- des bases de données multiples.

3) VAX-11 RDB/VMS version 2.3-0

Le RDBB/VMS utilise le modèle de données relationnel. Il est implanté aussi sur le VAX de Digital.

Ce SGBD est composé de

- le VAX CDD;

- la base de données qui est un ensemble de tableaux à deux dimensions;

- le RDO qui est un utilitaire interactif pour définir et manipuler les données dans la BD.

Il permet

- l'intégration des définitions de schéma logique, de schéma physique, et des contraintes d'intégrité dans un seul schéma.

-la manipulation des données avec mécanisme d'optimisation d'accès à la BD;

-l'utilisation simultanée des données par plusieurs utilisateurs;

- la protection de données;

4) VAX-11 COBOL VERSION 2

Le langage COBOL lui-même est un langage industriel de haut niveau et structuré. Le VAX-11 COBOL offre une extension COBOL DML pour que les programmes d'application fonctionnant avec le DBMS puissent accéder à la base de données.

VAX RDB est compatible avec le VAX-11 COBOL.

Le développement des programmes/COBOL s'effectue sous VAX/VMS.

1.4 CHOIX D'UNE DEMARCHE

Cette partie de l'introduction présente les grandes lignes de notre démarche de comparaison entre un SGBD CODASYL et un SGBD RELATIONNEL. L'objectif de cette démarche est précisément de déterminer les critères de comparaison, de fournir un plan d'expérience de mesures et de contrôler les conditions de comparaison.

1.4.1 Questions initiales

Pour accomplir l'objectif de comparaison dans la section 1.1, nous nous posons les questions suivantes:

- Quels sont les critères de comparaison ?
- Quelles mesures allons-nous effectuer ?
- Comment réaliser des conditions de comparaison ?

1.4.2 Critères de comparaison

Du point de vue des applications qui utilisent et manipulent les données dans la BD, la comparaison se rapporte aux deux aspects suivants:

L'aspect logique qui concerne la puissance du modèle des données et les primitives offertes par les SGBD.

L'aspect technique qui est directement lié à l'évaluation de la performance avec laquelle se fait la programmation d'application.

1.4.3 Plan d'expérience

Il s'agit de ce que nous voulons mesurer selon les critères de comparaison. Il existe en général deux types de mesures:

1) Les mesures qualitatives
Ce sont des mesures sur l'aspect logique des critères de comparaison. Nous décidons

- d'évaluer la puissance d'un SGBD pour représenter la structure de données;

- d'analyser l'impact de modifications des schémas sur les programmes d'application;

- d'estimer le degré algorithmique et la complexité d'un programme relatif à l'application donnée;

2) Les mesures quantitatives

Les mesures sont prises sur l'aspect technique des critères de comparaison. Nous voudrions

- évaluer la performance des SGBD, par exemple, tester des temps d'exécution, le nombre d'opérations entrées/sorties de petits programmes de types différents;

- estimer l'efficacité des programmes d'application sur la BD pour évaluer les efforts des SGBD ;

- mesurer le volume et le taux de remplissage d'une BD qui est sous le contrôle d'un SGBD donné.

1.4.4 Conditions de comparaison

Pour que la comparaison puisse avoir un sens il faut solliciter les systèmes observés de façon identique. C'est-à-dire qu'il faut bien contrôler les conditions expérimentales de la comparaison.

a) Le but

Le but de cette démarche dans notre cas est

- d'obtenir des schémas équivalents, étant donné un même schéma conceptuel qui est composé des données abstraites associant au profil d'activités reflétant les usages souhaités,

- de produire les mêmes programmes, étant donné la spécification d'une application et

- de charger les mêmes données dans les bases de données différentes.

Pour atteindre ce but, cette démarche s'est déroulée en plusieurs étapes:

- transformation du schéma conceptuel en schémas logiques contenant les mêmes informations que ce schéma conceptuel;

- élaboration des schémas physiques qui ont les mêmes possibilités d'accès à la BD;

- production des programmes qui ont la même spécification et la même structure algorithmique;

- chargement des mêmes données dans les BD.

b) Les méthodes choisies:

La méthode choisie pour élaborer les schémas est basée sur le Modèle d'Accès Général (MAG) [HAIN 86]. Ce modèle est capable de représenter la sémantique du schéma conceptuel et de décrire avec précision l'organisation des données du point de vue des accès techniques. Nous transformerons d'abord le schéma conceptuel en schéma d'accès nécessaire du MAG, ensuite le schéma d'accès nécessaire en schéma conforme au SGBD.

La méthode utilisée pour la construction des programmes d'application est basée sur une série de transformations automatiques décrite dans [HAIN 86]. A partir de la spécification d'une application donnée, nous utiliserons le Langage de Désignation de Données (LDA) pour dériver directement l'algorithme prédicatif. Puis, nous établissons, en considérant les primitives des SGBD, les algorithmes conformes. Enfin nous arriverons à programmer de façon systématique, en respectant certaines régles de transformation.

A l'égard de la génération de données de test, la démarche employée est la même que pour la construction des programmes.

1.5 DEMARCHE ET PLAN DU MEMOIRE

Le choix d'un SGBD dépend des différents besoins des utilisateurs et de leurs applications. Un des besoins vitaux des utilisateurs est, par exemple, celui de la performance du SGBD. Le besoin de la séparation des données et des programmes d'application qui traitent ces données est aussi fondamental. Il y en a encore d'autres. Il faudra donc en premier lieu déterminer sur quel aspect du SGBD se fera la comparaison et comment la réaliser. Le chapitre 2 déterminera les critères de la comparaison et à chaque critère, nous fournira un plan d'experience pour expliquer ce que nous voulons mesurer.

Le chapitre 3 illustre la démarche à suivre pour construire les schémas/MAG et les programmes d'application, et fait un rappel de l'algorithme prédicatif, de l'algorithme conforme et de l'ADL .

Après avoir étudié et défini les objectifs de la comparaison, il est primordial de connaître ce que l'on veut exactement mesurer et comment matérialiser les mesures voulues. Cette définition doit cerner ce qui figure dans les critères de la comparaison. Le chapitre 4 est consacré à ce problème.

Dans le chapitre 5 , nous essayerons d'analyser et de comparer les resultats des mesures pour notre cas. L'utilisateur sera capable de juger lui-même, si notre démarche est bien raisonnable, à partir des mesures prises.

Enfin, nous passerons à la fin de notre démarche, à l'élaboration de la conclusion de notre expérience.

CHAPITRE 2
CRITERES DE LA COMPARAISON ET PLAN D'EXPERIENCE

Ce chapitre détermine plus précisément les critères en fonction de ce que nous voulons mesurer. Pour chaque critère cité, nous assignerons les paramètres à mesurer. Dans le cadre de notre expérience, nous nous limiterons à l'étude de certaines caractéristiques du SGBD. Les autres seront ignorées.

première partie: L'ASPECT LOGIQUE

2.1 CRITERE 1 : LE MODELE DES DONNEES

2.1.1 L'analyse des concepts

En informatique, le modèle de structure des informations ENTITE-ASSOCIATION (E/A) est le plus largement accepté pour représenter le réel perçu.

Une question se pose quand on définit à partir du modèle E/A la structure de données dans la BD en fonction des applications organisationnelles: quel est le modèle acceptable ?

Il y a beaucoup d'aspects à considérer à ce propos. Nous ne nous concentrons que sur les aspects de représentation d'E/A et de réalisation de certaines contraintes d'intégrité par les différents SGBD.

2.1.2 Le plan d'expérience

Afin de comparer ultérieurement les qualités des SGBD quant à l'expression des concepts, la démarche consiste à relever les concepts fondamentaux à propos du modèle E/A et d'observer de quelle façon le SGBD traduit ces concepts.

a) Représentation d'entité et type d'entité

Un SGBD doit être capable de stocker et consulter les donnees qui sont sous forme d'entité. Cette entité ou type d'entité est une abstraction d'un objet perçu dans la réalité.

- Nous distinguerons pour chaque SGBD la forme de représentation logique et la description physique d'entité.

b) Représentation d'association et type d'association binaire

Une association existe entre deux types d'entité non forcément distincts. Un type d'association est caractérisé par la connectivité avec ses entités.

- Pour une association et deux types d'entité nous observerons comment le SGBD réalisera les différents types de connectivité (c'est-à-dire leur format logique et leur description physique):

1-1, 1-N, N-1 et N-N ;

- Pour une association entre le même type d'entité: récursive, nous ferons la même chose que ci-dessus.

c) Représentation d'association et type d'association ternaire

Nous nous contenterons d'observer comment les SGBD représentent une association ternaire:

- avec un attribut ou
- sans attribut

d) Représentation des attributs

Les attributs sont des caractéristiques ou qualités d'une entité ou d'une association.

A l'égard des propriétés d'un attribut, nous observerons le format logique et la description physique de

- l'attribut simple et élémentaire;
- l'attribut simple et décomposable;
- l'attribut répétitif et élémentaire;
- l'attribut répétitif et décomposable.

e) En ce qui concerne l'identifiant d'un type d'entité, nous allons observer:

- un identifiant qui est un attribut;

- un identifiant qui est un groupe d'attributs;

- plusieurs identifiants dans un type d'entité.

Dans la mesure où l'approche relationnelle est en grande partie basée sur la logiqué, lors de la transformation du réel perçu en modele relationnel, les domaines et les relations sont soumises aux diverses contraintes qui représentent des propriétés stables de ce réel perçu. [HAIN 87]. Ceci nous amène à tenter d'examiner certains types de contraintes prises en charge par le SGBD en observant le modèle de données:

- identité et unicité;

2.2 CRITERE 2 : INDEPENDANCE DES DONNEES

2.2.1 L'analyse des concepts

L'indépendance des données concerne le problème de la séparation des programmes d'application sur la BD et de la structure et organisation des données dans la base de données. Elle est un des besoins fondamendaux des utilisateurs. La définition formelle se trouve dans [DATE 81].

Dans notre cas, nous voudrions exprimer par ce terme, que l'indépendance des données est une capacité du programme de fonctionner correctement après un changement des types de données dans la BD. Il est clair qu'elle est aussi une fonction vitale du SGBD.

2.2.2 Plan d'expérience

Normalement, l'indépendance se situe à deux niveaux differents:

a) indépendance logique

Cette indépendance se situe entre le schéma conceptuel et le schéma externe (SUBSCHEMA) par le raisonnement qui suit : Un programme d'application travaille avec un schéma externe. Si la modification du schéma conceptuel n'entraîne aucune modification du shéma externe, alors il y a une indépendance entre eux et par conséquent le programme d'application ne change rien.

Afin d'envisager les possibilités et les restrictions, nous envisageons:

- d'ajouter un item à une entité;

- d'ajouter un type d'entité au schéma conceptuel;

- d'implanter une association nouvelle entre deux entités existantes dans le schéma conceptuel;

- d'examiner les règles de modification des informations du schéma de la BD;

- d'examiner les règles de suppression des informations du schéma de la BD.

Notons qu'en cas de changement ou suppression d'un nom d'item ou d'un type d'entité qui est déjà utilisé par les programmes d'application, la modification de ces derniers est inévitable.

b) indépendance physique

Le cas opposé est où il n'est pas possible de changer l'organisation physique sans entraîner une modification des programmes d'application. Cette définition nous amène à essayer d'examiner les situations suivantes:

- Pour un type d'article, par exemple, EMPRUNT (voir schéma-1 dans chapitre 3) de chaque BD, changer DATE-DEBUT en une clé d'accès indexé afin d'accelérer le processus de traitement pour fournir un relevé des "dates-en-retard" pour tous les emprunteurs.

2.3 CRITERE 3 : DEGRE ALGORITHMIQUE DU PROGRAMME

2.3.1 L'analyse des concepts

Il est intéressant de savoir avec quel SGBD on travaille de façon la plus "stable" et la plus "aisée". Ce problème est lié au DML (DATA MANIPULATION LANGUAGE) qui est procédural ou non. Normalement, "non procedural" implique l'écriture aisée de programmes d'application, tandis que "procédural" implique qu'il faut toujours spécifier le processus quand un utilisateur prétend écrire le programme d'application. Ce dernier cause donc l'augmentation des codes sources du programme. En ce sens nous pouvons dire que ce critère correspond aussi à mesurer la complexité de la structure de programmes semblables.

En ce qui concerne la procéduralité, elle est liée au mode de description des données.

Un DML est procédural si le programme doit spécifier les accès nécessaires pour obtenir les données;

Un DML est non procédural si le programme peut décrire les données par ses propriétés logiques.

En termes de comparaison, si l'on écrit à l'aide du DML d'un SGBD, un programme d'application qui est non procédural ou moins procédural, alors le SGBD est plus puissant.

2.3.2 Le plan d'expérience

Nous nous engagerons à compter pour un programme d'application:

- le nombre total d'instructions exécutables;

- le nombre d'instructions DML;

- la proportion des deux paramètres. Il convient de s'assurer s'il est vrai qu'il y a moins d'instructions sources du programme lorsque ce programme est non procédural.

- le nombre de structure IF ou FOR dans l'algorithme/ADL conforme. Ce paramètre implique la possibilité de contrôle explicite et de navigation de la manipulation des données dans la BD;

- le nombre de déclarations spécifiques de zones de données à la BD dans le programme/COBOL;

Deuxième partie L'ASPECT TECHNIQUE

2.4 CRITERE 4 : PERFORMANCE DES SGBD

2.4.1 l'analyse des concepts

Le besoin d'une bonne performance du système vis-à-vis de certains goulots d'étranglements potentiels conduit à un besoin de mesure et de prédiction de cette performance. Ici, nous ne faisons que des mesures sur les programmes d'application de la BD.

En effet, pour évaluer la performance de SGBD, il convient d'exploiter les demandes (applications) d'utilisateurs et leurs effets sur les activités gérées par le SGBD. Dans cet ordre d'idées, les programmes d'application sont vus comme des charges de SGBD et les mesures sur ces charges signifient donc la collecte des données concernant les activités du SGBD quand il sert les charges.

Les mesures de performance exigent un choix des charges qui vont être réalisées lors de la collecte des données. Ces charges artificielles sont appellées "benchmark" de mesures.

Au niveau d'un programme d'application qui utilise et manipule les données dans la BD, l'efficacité, un indice important de la performance du programme, dépend de l'organisation de la BD et du nombre d'opérations d'E/S.

D'une part, les E/S dans un SGBD sont souvent une cause de goulot d'étranglement. Il est donc souhaitable d'organiser les records dans la BD de sorte que le nombre d'opérations E/S soit minimal.

D'autre part, un programme d'application, avant qu'il retrouve les records désirés dans la BD, doit faire référence à d'autres records lors de la recherche de l'index et/ou de la vérification des contraintes d'intégrité sur les records. Il est donc souhaitable de mesurer les effets des programmes d'application en termes des records logiques de référence.

2.4.2 La classification des applications

Il est évident que les applications peuvent être très variées selon les besoins et l'environnement de l'utilisateur. Nous insistons donc encore une fois sur le fait que nous ne nous intéressons qu'aux problèmes liés aux données dans la BD. Nous aborderons la classification des applications dans cet esprit.

En effet, les différents types d'analyse pour classifier les applications sont des critères selon lesquels nous choisirons celles-ci. La comparaison des performances des SGBD va dépendre des applications que nous allons définir dans cette section. Dans notre expérience, nous en avons développé 25. Les spécifications des applications et leur typologie sont décrites dans le chapitre 3.

a) COMPOSANTES METHODOLOGIQUES

Pour avoir une terminologie homogène, nous allons définir quelques termes:

BASE DE DONNEES

La base de données est supposée représenter toutes les données du système d'information ou, autrement dit, une réalisation du schéma conceptuel que l'on a fourni.

OBJET

Un objet est une description de certaines entités logiques. Par exemple, il peut être utilisé comme un record dans le système de fichiers traditionnels, ou un tuplet d'une relation du SGBD RELATIONNEL.

Un objet est aussi une unité élémentaire d'information qui est demandée par les utilisateurs ou utilisée par les programmes d'application. Son contenu est donc différent d'une application à l'autre Par exemple, pour l'application "Lister tous les numéros d'OUVRAGE.", l'objet est un record d'OUVRAGE, tandis que pour l'application "Fournir une liste d'ouvrages qui ont au moins un exemplaire.", l'objet est constitué d'un record d'OUVRAGE et d'un record d'EXEMPLAIRE.

ATTRIBUT ET VALEUR D'ATTRIBUT

Les attributs sont des propriétés associées aux objets. La caractéristique distinguée d'un attribut est indiquée par la valeur de celui-ci. Par exemple, pour le type d'article (ou un fichier) OUVRAGE dans le schéma conceptuel de BIBLIOTHEQUE (voir schéma-1 au chapitre 3), chaque article est un objet et les attributs sont NUMOU, TITRE, ANNEE, EDITEUR. Un objet spécifique a par exemple, les valeurs: NUMOU-12345, TITRE-BASE DE DONNEES, ANNEE-1986 et EDITEUR-PARIS.

CLE D'ACCES

Une clé d'accès est un attribut ou groupe d'attributs d'un même objet tel qu'il existe un mécanisme permettant d'accéder aux objets.

Une des fonctions d'un SGBD est de retrouver les objets désirés dans la base de données en répondant à la demande d'un programme d'application. Pour ce faire, le programmeur peut indiquer la clé d'accés dans le programme.

b) LES TYPES D'ANALYSE

§1: TYPE D'OPERATION SUR LA BASE DE DONNEES

Un programme d'application sur la base de données a pour but de manipuler des données dans cette base de données. En fonction du DML (Data Manipulation Langage), les applications peuvent se classer en deux types:

- consultation;
  Les opérations typiques:
    FIND...; FETCH...; en VAX DBMS et
    FOR...GET; en RDB/VMS;

- mise à jour.
  Les opérations typiques:
    MODIFY; ERASE; STORE.
    En VAX DBMS il y en a encore CONNECT et DISCONNECT.

§2: TAILLE DE L'OBJET A TRAITER

Chaque programme d'application traite interactivement ou non des objets qui contiennent une ou plusieurs entités. La taille de l'objet est déterminée par le nombre d'entités qui le constituent:

- élémentaire:
  correspondant à une unité élémentaire d'information. Par exemple, prenons un programme d'application qui imprime le titre d'un ouvrage étant donné un numéro d'ouvrage;

- moyennement complexe :
  un objet regroupe des entités de deux ou trois types. Par exemple, un ouvrage avec ses exemplaires.

- complexe:
  un objet qui groupe des entités d'un plus grand nombre de types. Par exemple une application qui fournit toutes les informations concernant un ouvrage donné.

§3: NOMBRE D'OBJETS A TRAITER

Un programme d'application traite les données dans la BD de manière différente selon la conception d'objet:

- transactionnelle ou ponctuelle;
  Le programme s'exécute une fois en utilisant les primitives d'accès ponctuel qui est défini par la séquence et par la position des objets dans la BD.

- de masse ou répétitive;

Le programme d'application s'exécute en faisant reference à une grande quantité d'objets pour accomplir son objectif.

Par exemple, une application "Etant donné un numéro d'ouvrage, fournir ses exemplaires", est dite transactionnelle, tandis que l'application "Donner une liste des exemplaires de tous les ouvrages" est appelée application de masse.

§4 : MECANISME D'ACCES UTILISE

Pour accéder à la BD, le programme d'application doit employer un mécanisme d'accès. Ou bien le SGBD s'en occupe, ou bien le programmeur le prend en considération. Il y a trois types de mécanismes d'accès:

- séquentiel;

Le programme d'application accède aux objets dans l'ordre (premier,suivant; ou dernier, précédent) des objets stockés dans la BD sans clé d'accès, ainsi qu'aux valeurs des attributs.

- par clé (index, calcul) ou par chemin;

Ceci correspond à un mécanisme d'accès dont l'utilisation conduit à la sélection des objets. En termes de MAG, il s'agit d'une condition évaluable par accès.

- séquentiel filtré;

Il s'agit de la condition de sélection des objets dans une séquence par le programme d'application. Cette condition ne peut être évaluée par simple activation des mécanismes d'accès. Dans ce cas, il n'existe aucun mécanisme d'accès qui permette d'obtenir les éléments sélectionnés. Le programme doit donc accéder séquentiellement à tous les objets et vérifier explicitement pour chacun d'eux la condition de sélection. Les SGBD RELATIONNELS offrent en générale la possibilité d'accès par filtre, dans ce cas, c'est le SGBD lui-même qui effectue l'accès séquentiel et la vérification de la condition de sélection.

c) LA TYPOLOGIE

En combinant ce que nous avons écrit dans le point b), nous obtenons 24 types d'applications sur la BD (voire la figure-2.1).

type d'opération	taille de l'objet	nombre d'objets	mecanisme d'accès	numéro
CONSULTATION	ELEMENTAIRE	transactionnel	par clé	1
		de masse	séquentiel	2
			seq.filtre	3
			par clé	4
	MOY.COMPLEXE	transactionnel	par clé	5
		de masse	séquentiel	6
			seq.filtre	7
			par clé	8
	COMPLEXE	transactionnel	par clé	9
		de masse	séquentiel	10
			séq.filtré	11
			par clé	12
MISE A JOUR	ELEMENTAIRE	transactionnel	par clé	13
		de masse	séquentiel	14
			séq.filtré	15
			par clé	16
	MOY.COMPLEXE	transactionnel	par clé	17
		de masse	séquentiel	18
			séq.filtré	19
			par clé	20
	COMPLEXE	transactionnel	par clé	21
		de masse	séquentiel	22
			séq.filtré	23
			par clé	24

FIGURE-2.1 La typologie des applications

2.4.3 Le plan d'expérience

a) En ce qui concerne le temps, pour chaque programme des applications nous observerons:

- le temps de réponse ou temps total d'exécution d'un programme : il est défini comme un intervale de temps entre le moment où le programme commence son exécution et le moment où le programme se termine;

- overhead CPU entraîné: il s'agit du temps CPU consomme par le système qui va prendre en compte et satisfaire la requête (un programme d'application), avec recherche des index, ou calcul des adresses de données desirées et gestion de fichiers etc.

b) En termes d'accès à la BD:

- le nombre d'accès au disque par un programme d'application;

- le nombre de défauts des pages disque produits par un programme d'application.

c) Nous voudrions l'examiner en termes de records logiques et de records physiques.

Nous définissons tout d'abord l'efficacité d'une requête en termes de records logiques comme suit:

$$LRE = \frac{\text{NOMBRE DE RECORDS QUI SATISFONT UNE REQUETE}}{\text{NOMBRE TOTAL DE RECORDS LUS DANS LA BD}}$$

$$= \frac{NRS}{NRL}$$

Il est clair que l'efficacité est ici dépendante de l'organisation physique des données. Un programme d'application fonctionnant avec un SGBD est dit efficace si la proportion LRE approche l'unite. Par exemple, lorsque 10 records satisfont une requête mais que le système doit accéder à 5 index records avant d'accéder aux 10 records, alors la proportion est de 10/15 = 0.67.

Maintenant nous définissons l'efficacité d'une requête en termes de records physiques :

$$PRE = \frac{\text{NOMBRE TOTAL D'ACCES AU DISQUE DE LA REQUETE}}{\text{NOMBRE DE RECORDS QUI SATISFONT LA REQUETE}}$$

$$= \frac{NAP}{NRS}$$

Nous notons dans cette définition qu'un accès au disque correspond à une page du disque et qu'une page peut contenir plusieurs records logiques. Plus la page contient de records logiques, moins les opérations E/S seront produites pour répondre à la requête.

Par cette définition, un programme P1 est dit plus efficace que l'autre p2, si la valeur du PRE de P1 est plus petite que celle de p2. A titre d'exemple de calcul, si l'on a besoin de 3 pages pour répondre à une requête qui désire 15 records, alors le PRE est égal 3/15 = 0.2.

La motivation de ces deux dernières définitions est de tester les efforts du SGBD pour un programme d'application donné.

2.5 CRITERE 5 : VOLUME ET TAUX DE REMPLISSAGE DE LA BD

Le volume et le taux de remplissage de la BD sont des caractéristiques importantes de performance, bien qu'ils soient moins cruciaux grâce aux développements technologiques du hardware.

Nous mesurerons après la création de la BD:

- le volume statique de celle-ci;
- le taux de remplissage de celle-ci.

CHAPITRE 3
ELABORATION DES SCHEMAS ET DES PROGRAMMES

En ce qui concerne la comparaison entre deux systèmes différents, il est crucial de contrôler les conditions expérimentales. Les précisions ont déjà été apportées dans l'introduction. Dans ce chapitre, nous présenterons la manière dont nous avons réalisé le contrôle des conditions et rappellerons les méthodes dont nous disposons pour le réaliser.

La section 3.1 va être consacrée à la définition de l'équivalence de bases de données et de programmes d'application.

La section 3.2 présente la transformation du schéma conceptuel BIBLIOTHEQUE en schémas/MAG conformes aux SGBD.

La section 3.3 illustre la démarche de la construction des programmes d'application et fait un rappel d'algorithme prédicatif, d'algorithme conforme au SGBD et de langage ADL avec lesquels nous réaliserons les programmes d'application.

Enfin, la section 3.4 donne les grandes lignes du programme de chargement des données de test.

3.1 PROBLEME D'EQUIVALENCE

En vue de comparer les deux SGBD selon des applications sur la BD, il nous faudrait d'abord créer deux bases de données, CODASYL et RELATIONNELLE, qui respectent le même schéma conceptuel et représentent la même réalité. En plus, étant donné la spécification d'une application, nous développerons deux programmes de l'application qui fonctionnent avec les SGBD différents et sont aussi équivalents. Ceci nous conduit à définir la conception d'équivalence:

3.1.1 Equivalence d'états des bases de données

Nous voudrions dire que deux états des bases de données sont équivalents à un moment donné si ces deux bases de données représentent la même réalité au moment considéré. Dit de manière plus concrète: les deux bases de données contiennent les mêmes données.

Nous ne parlons pas de l'équivalence du modèle sémantique des données. Pour cela, nous renvoyons le lecteur à la référence [SHEL 81].

Remarquons que dans notre expérience, les états initiaux des bases de données sont assurés par la transformation systématique de schéma conceptuel en schéma d'accès necessaires de MAG, et en schémas conformes aux SGBD (voir 3.2.3). Nous avons par ailleurs mis au point deux programmes de chargement de la BD en utilisant aussi la méthode systèmatique (voir la section 3.4).

3.1.2 Equivalence des programmes d'une application

Soient ;

- deux bases de données CODASYL et RELATIONNEL,

- deux programmes P1 et P2 qui sont dérivés de la même description D d' une application et qui s'exécutent respectivement sur la BD CODASYL et BD RELATIONNEL

tels que

avant des exécutions des P1 et P2, les BD soient dans les états BD1i et BD2i dits "équivalents" et

après les exécutions des P1 et P2, nous obtenions les résultats R1 et R2 et que les BD soient dans les états BD1i+1 et BD2i+1.

Nous définissons que

$$P1 = P2$$

si et seulement si

$$R1 = R2 \text{ et}$$

$$BD1i+1 = BD2i+1.$$

On verra dans la section 3.3 que l'équivalence des programmes d'application peut être garantie par la transformation systèmatique de la spécification d'une application en programmes effectifs de l'application, si l'on respecte des règles de transformation.

3.2 ELABORATION DES SCHEMAS/MAG

Nous développerons très brièvement cette partie de la démarche. Notons que nous ne présenterons pas dans ce paragraphe les textes des schémas des SGBD. En effet, pour produire les schémas/MAG conformes (voir 3.2.3), nous devons tenir compte des primitives (voir 3.2.3) des SGBD. Or, la traduction de schéma conforme/MAG en schéma d'un SGBD ne pose pas de problème. Nous les présentons dans l'annexe-1.

3.2.1 Brève description du système BIBLIOTHEQUE

Une bibliothèque gère le stockage, l'indexation et l'emprunt d'un ensemble de livres. Chaque livre est un exemplaire d'un ouvrage auquel peuvent correspondre un nombre quelconque d'exemplaires. Un ouvrage est caractérisé par un numero identifiant, un titre, ses auteurs éventuels, l'année de parution, l'éditeur et les mots-clés qui le décrivent. On désire connaître, si possible, l'origine de chaque auteur. Chaque exemplaire est caractérisé par un code identifiant et sa localisation (étage, travée, rayon). D'un emprunteur on connaît le numero identifiant , le nom, l'adresse, les exemplaires en cours d'emprunt accompagnés de la date d'emprunt, ainsi que tous les emprunts clôturés, avec leurs dates de début et de fin.

Le schéma-1 est le schéma conceptuel correspondant à cette description.

Les caractéristiques des attributs sont les suivantes:

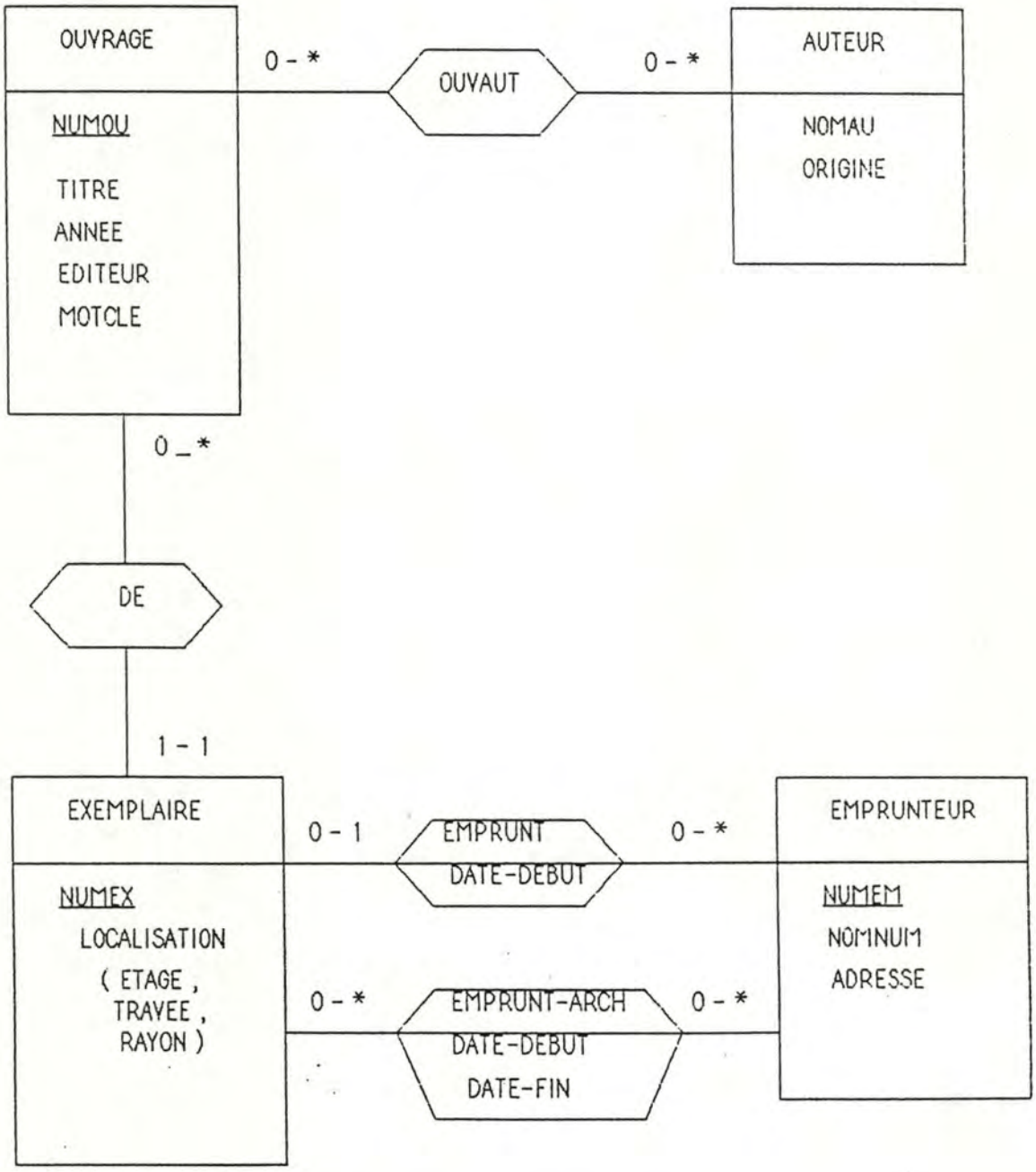
NUMAU D'AUTEUR :	DECIMAL 6;
NOM D'AUTEUR :	CHARACTER 30;
ORIGINE D'AUTEUR :	CHARACTER 30;
NUMOU D'OUVRAGE :	DECIMAL 6;
TITRE D'OUVRAGE :	CHARACTER 30;
ANNEE D'OUVRAGE :	DECIMAL 6;
EDITEUR D'OUVRAGE:	CHARACTER 30;
MOTCLE D'OUVRAGE :	CHARACTER 30;
NUMEX D'EXEMPLAIRE :	DECIMAL 6;
ETAGE D'EXEMPLAIRE :	DECIMAL 2;
TRAVEE D'EXEMPLAIRE:	DECIMAL 2;
RAYON D'EXEMPLAIRE :	DECIMAL 2;

NUMEM D'EMPRUNTEUR : DECIMAL 6;
NOM D'EMPRUNTEUR : CHARACTER 30;
ADRESSE D'EMPRUNTEUR: CHARACTER 30;

DATE-DEBUT D'EMPRUNT: DECIMAL 6 (an-mois-jour);

DATE-DEBUT D'EMPRUNT-ARCH: DECIMAL 6;
DATE-FIN D'EMPRUNT-ARCH : DECIMAL 6;

;



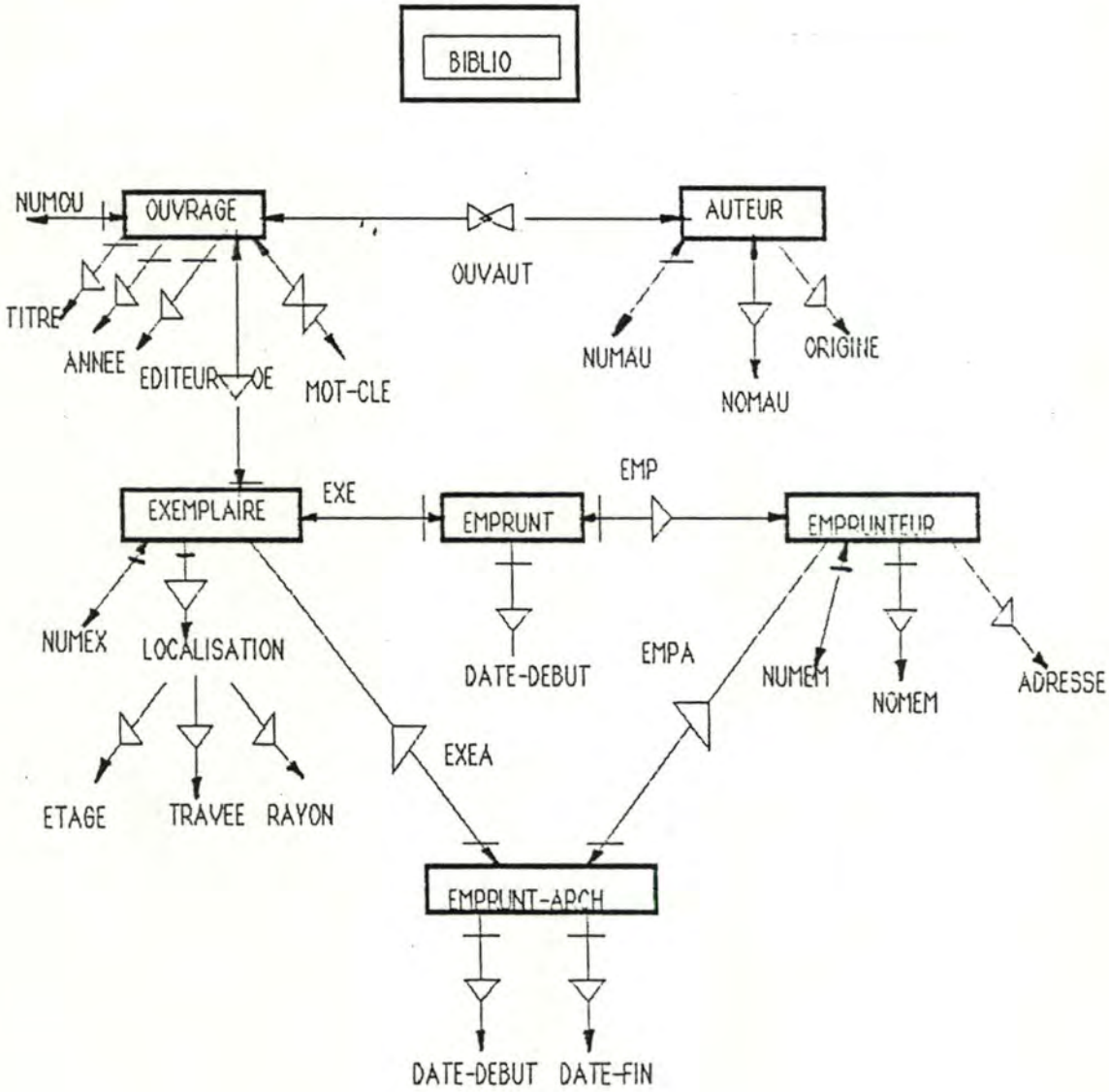
SCHEMA 1 Schéma conceptuel des données

3.2.2 La construction du schéma/MAG des accès nécessaires

Le résultat est représenté dans le schéma-2.

Nous avons choisi de représenter le type d'association avec attributs par le type d'article et les types de chemins associés. Donc EMPRUNT et EMPRUNT-ARCH dans le schéma-1 doivent être transformés (voir schéma-2). L'équivalence est assurée par la règle de transformation rigoureuse de schéma conceptuel en schéma MAG (cfr. chapitre 7 de [HAIN 86]

Notons que la détermination des accès nécessaires est normalement faite à partir d'un relevé des structures de données strictement nécessaires à l'exécution des algorithmes effectifs des programmes d'application, selon la nature du système d'information et les besoins des applications. Nous ne faisons pas ce relevé et nous considérons que les accès nécessaires sont aussi donnés. La garantie des mêmes possiblités d'accès aux BD doit être sollicitée à partir d'ici.



SCHEMA 2 Schéma des accès nécessaires de la BD BIBLIO

3.2.3 Les principales primitives et restrictions des SGBD par rapport au MAG

Afin d'élaborer des schémas conformes au SGBD et des algorithmes conformes (3.3.2), nous donnons dans ce paragraphe les principales primitives et des restrictions offertes par les SGBD par rapport au MAG.

A) VAX-11 DBMS

1) Une seule clé d'accès dans un set par type d'article;

2) Un identifiant d'un article est une clé d'accès de cet article;

3) Tout type de chemin est 1-N et est doté d'un inverse;

4) L'origine et les cibles d'un type de chemin sont des types d'article distincts;

5) Contrainte d'existence sur les cibles des types de chemins 1-N uniquement;

6) Il n'y a pas d'identifiant dans la BD par type d'article.

B) VAX-11 RDB/VMS

1) Pas de type de chemin;

2) Pas d'item décomposable;

3) Pas d'item répétitif;

4) Possibilité de valeur NULL pour les items facultatifs;

5) Au moins un item par type d'article;

6) Au moins un identifiant par type d'article.

3.2.4 La construction des schémas conformes

Cette construction est une transformation du schéma d'acces nécessaires en schémas conformes aux SGBD.

Lors de ce processus, on doit tenir compte des restrictions et des primitives (crf. 3.2.3) offertes par les SGBD.

a) Le schéma conforme à DBMS (voir schéma-3)

- Nous chercherons à éliminer les types de chemins N-N dans le schéma-2. Le type de chemin OUVAUT doit être éliminé selon la règle de transformation suivante (figure-3.1):

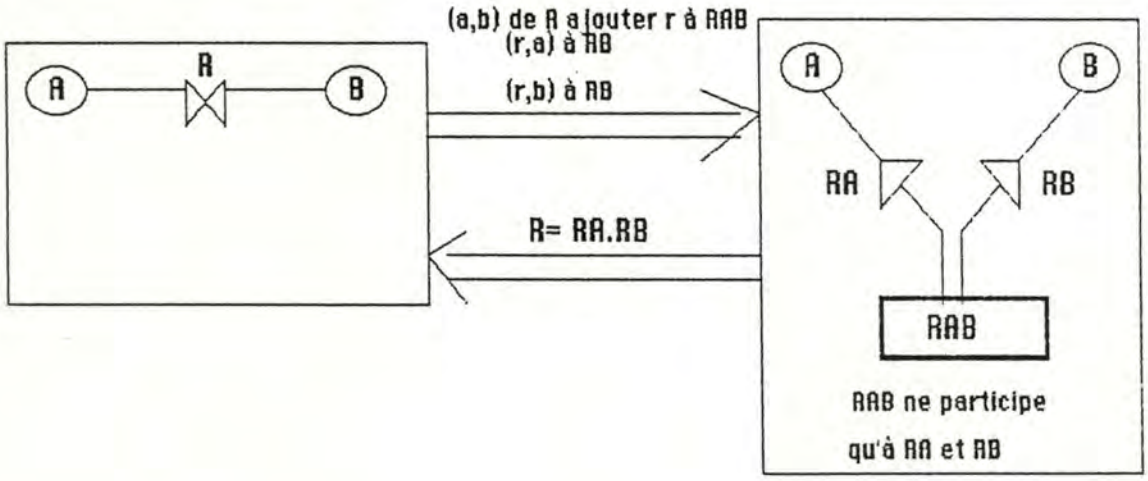
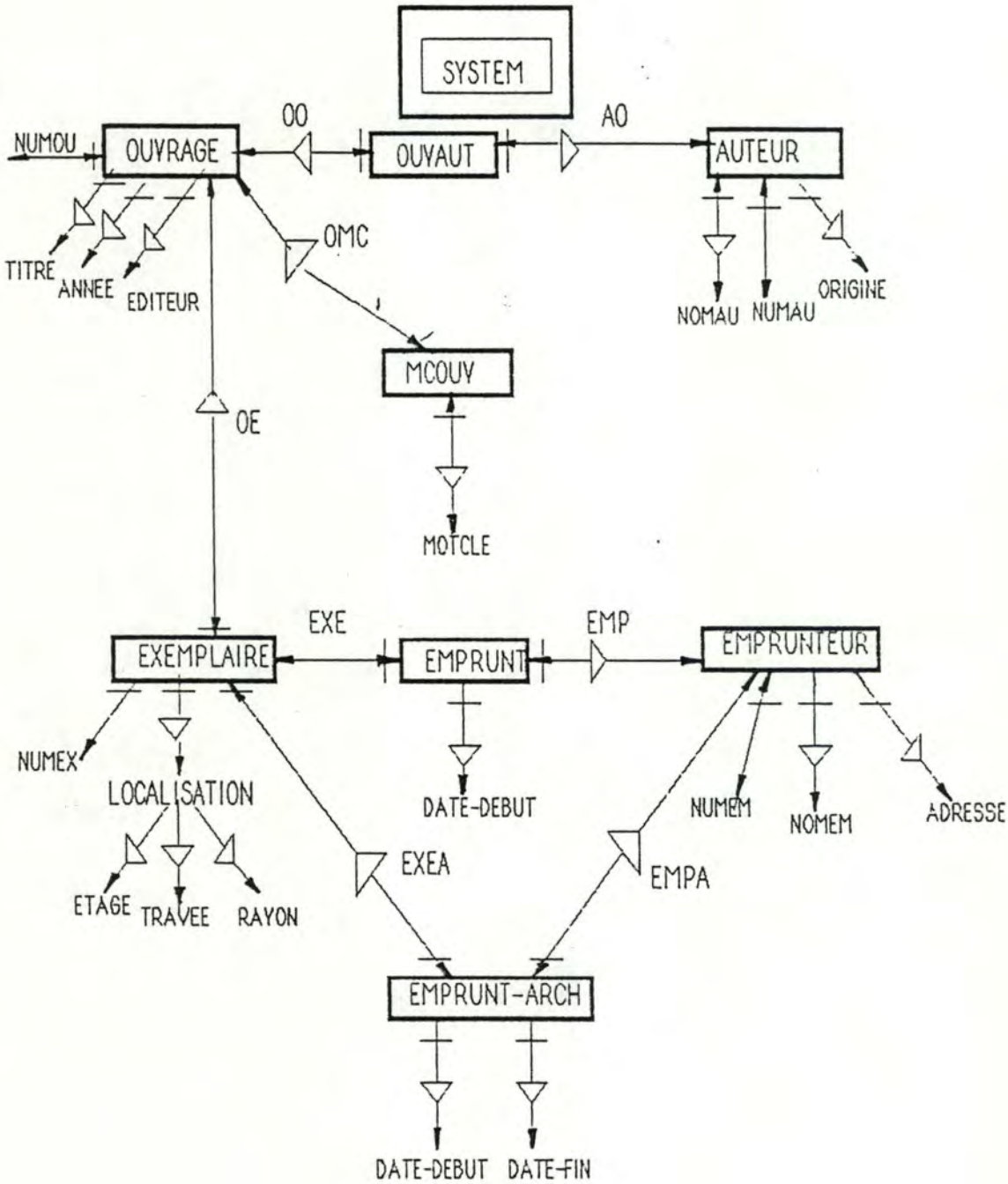


FIGURE-3.1 Création d'un type d'article

- L'item MOTCLE d'OUVRAGE est répétitif. Donc nous l'éliminerons avec la même règle que ci-dessus (cfr. figure-3.1).

Notons que le DBMS admet un type d'article qui peut avoir deux clés d'accès. Mais il faut les définir dans des SET différents. Il est donc possible d'implanter les deux clés d'accès NUMAU et NOMAU pour AUTEUR.



CONTRAINTE SUPPLEMENTAIRE

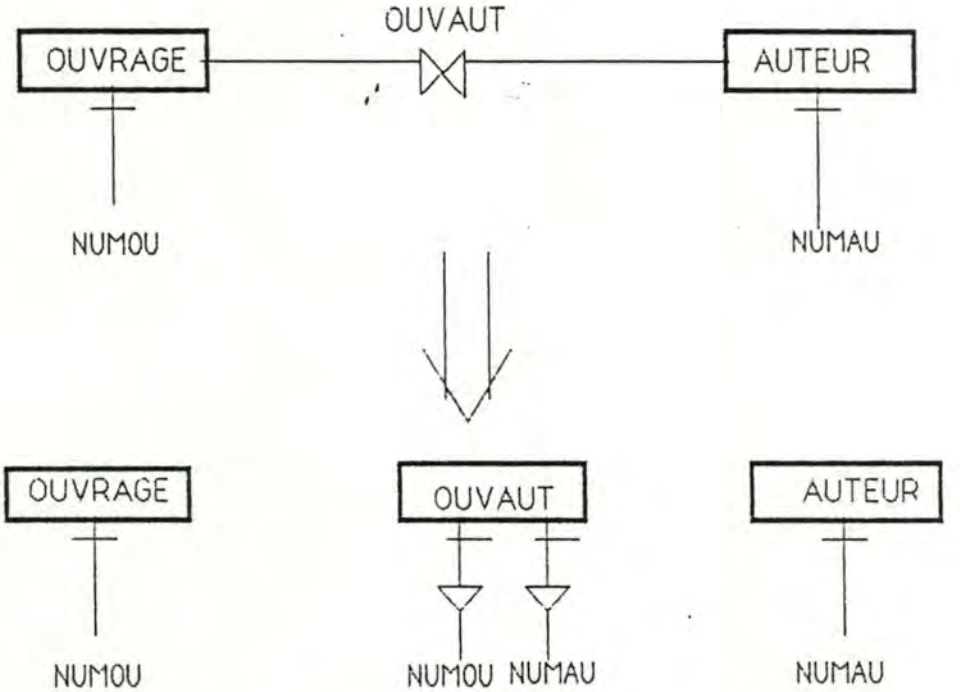
MCOUV est identifié par MOTCLE et OUVRAGE via OMC;

REMARQUE: OUVRAGE, AUTEUR, EXEMPLAIRE, EMPRUNTEUR sont des MEMBER records de SYSTEM SET.

SCHEMA-3 Schéma de la DB BIBLIO1 conforme à DBMS

b) Le schéma conforme à RDB (voir schéma-4)

- Du schéma-2, nous éliminerons tous les types de chemin par rotation vers les identifiants des types d'article liés à ces chemins. Par exemple:



NUMOU(: OUVAUT) in NUMOU(: OUVRAGE)
NUMAU(: OUVAUT) in NUMAU(: AUTEUR)

FIGURE-3.2 Exemple d'élimination d'un type de chemin

Remarquons que les deux items d'OUVAUT doivent être les clés d'accès dans la figure-3.2 afin d'assurer l'équivalence d'accès. Mais tous les deux ne sont pas identifiants. Nous combinons les deux items qui jouent ce rôle.

Remarquons encore que pour définir un identifiant AB en combinant deux clés indexées (en RDB une clé d'accès est obligatoirement une clé indexée) A et B, il suffit de définir A comme clé indexée et AB comme une clé indexée et non dupliquée. A ce moment-là, B est automatiquement une clé indexée, parce que l'on sait qu'une clé indexée est une clé d'accès et aussi une clé triée.

Nous présentons cette situation à partir de l'exemple OUVAUT dans la figure-3.3.

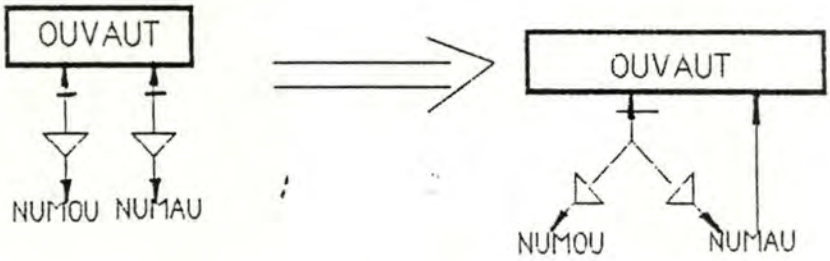
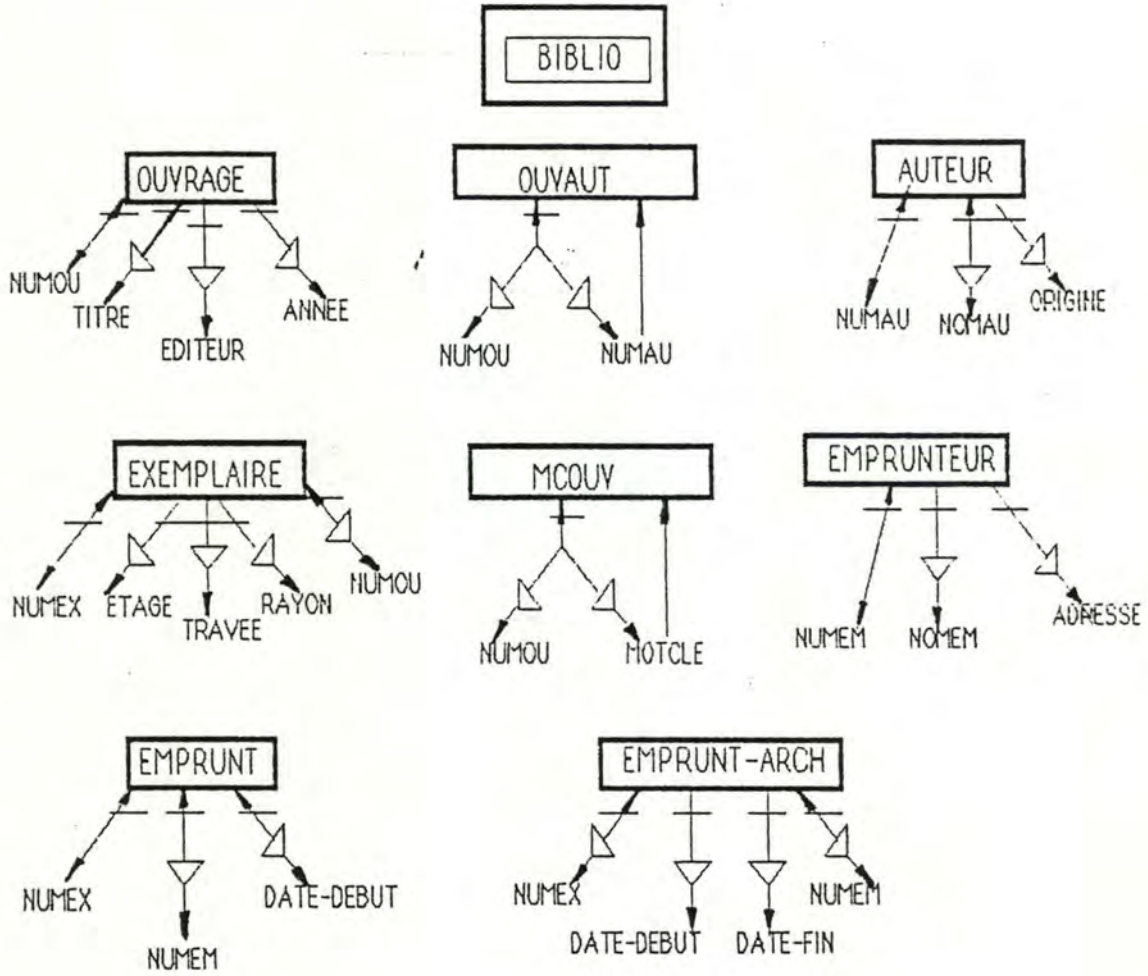


FIGURE-3.3 Représentation d'un identifiant avec deux attributs

- En ce qui concerne l'item répétitif MOTCLE, nous le transformons de la même manière que le type de chemin N-N (voir le point a)). Nous obtenons un type d'article MCOUV (voir schéma-3). Nous éliminons ensuite le type de chemin entre OUVRAGE et MCOUV de la même façon que ci-dessus.

- Nous remplaçons l'item décomposable LOCALISATION par ses composants.

Le résultat de cette transformation est présente dans le schéma-4. Nous donnons en même temps les contraintes d'intégrité.



CONTRAINTES SUPPLEMENTAIRES:

1. NUMAU(: OUVAUT) in NUMAU(: AUTEUR);
2. NUMOU(: OUVAUT) in NUMOU(: OUVRAGE);
3. NUMOU(: MCOUV) in NUMOU(:OUVRAGE);
4. NUMOU(: EXEMPLAIRE) in NUMOU(:OUVRAGE);
5. NUMEX(: EMPRUNT) in NUMEX(: EXEMPLAIRE);
6. NUMEM(: EMPRUNT) in NUMEM(: EMPRUNTEUR);
7. NUMEX(: EMPRUNT-ARCH) in NUMEX(: EXEMPLAIRE);
8. NUMEM(: EMPRUNT-ARCH) in NUMEM(: EMPRUNTEUR);
9. OUVAUT est identifié par NUMOU et NUMAU;
10. MCOUV est identifié par NUMOU et MOTCLE.

SCHEMA-4 Schéma de la BD BIBLIO conforme à RDB

3.3 L'ELABORATION DES PROGRAMMES

Dans cette section, nous allons décrire une partie de la transformation de l'application en programmes pouvant fonctionner effectivement sur base des différents SGBD avec le même langage COBOL. La figure-3.4 illustre cette transformation.

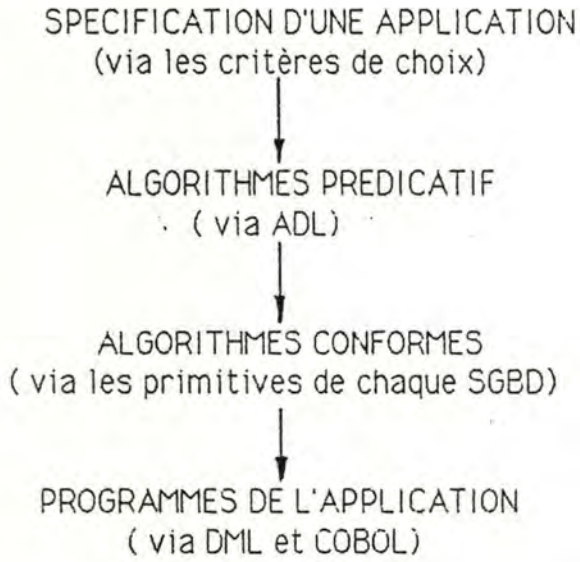


FIGURE-3.4 Transformation d'application

L'objectif de cette section est d'illustrer la démarche utilisée. Nous pouvons considérer que la construction des programmes d'application est un processus systématique grâce à l'algorithme prédicatif et à l'algorithme conforme [HAIN 86].

A titre d'exemple nous ne nous donnons qu'une application pour chaque stade de la transformation. Nous attachons cette application P4 à un module applé module-1 qui va jouer le rôle de conversion des appels de P4 en appels au SGBD réel.

La description de l'application est la suivante:

Etant donné le titre, l'année d'édition et le nom d'un auteur d'un ouvrage, fournir le numéro d'un exemplaire disponible.

3.3.1 L'algorithme prédicatif et ADL/MAG

Cet algorithme est dérivé directement de la spécification de l'application et utilise le Langage de Désignation de Données (ADL). En ce qui concerne la désignation des données auxquelles l'application accède dans la base de données, l'algorithme utilise une description qui précise les propriétés de ces données (par une condition de sélection ou prédicat) et non la manière d'y accéder. Pour le module-1 nous avons l'algorithme prédicatif suivant:

```
input ( XTITRE, XAN, XNOMAU ) ;
for EX := # 1 EXEMPLAIRE ( ( EXE : O EMPRUNT ) and
                           ( OE : OUVRAGE ( ( : TITRE = XTITRE ) and
                           ( : ANNEE = XAN ) and
                            ( OUVAUT: AUTEUR ( : NOMAU = XNOMAU )))) do
   print NUMEX ( : EX ) ;
endfor;
```

ALG-3.1 Algorithme prédicatif du module-1

Nous présenterons toutes les spécifications d'applications de notre expérience et les algorithmes prédicatifs correspondants au point 3.3.4.

3.3.2: L'algorithme conforme au SGBD

Ce processus suit le précedent. Il produit des versions équivalentes qui sont conformes au SGBD. C'est-à-dire que l'algorithme opère sur le schéma conforme au SGBD (voir schéma-3, et schéma-4).

A ce stade de transformation de l'algorithme prédicatif en algorithme conforme, il faut tenir compte des principales primitives et des restrictions offertes par le SGBD CODASYL et le SGBD RELATIONNEL (cfr. 3.2.3) par rapport au MAG. Pour le module-1 on a

a) Conforme à DBMS :

```
input ( XTITRE , XAN , XNOMAU ) ;
trouve := false ;
for A := AUTEUR ( : NOMAU = XNOMAU ) do
    for O := OUVRAGE ( OUVAUT : A ) do
        if (TITRE( : O ) = XTITRE) and (ANNEE( : O ) = XAN )) then
          for EX := EXEMPLAIRE ( OE : O ) do
              EX-OK := TRUE ;
              for EM := EMPRUNT ( EXE : EX ) do
                  EX-OK := FALSE ;
              endfor ;
               if EX-OK then
                  trouve := true ;
                  print NUMEX ( :EX ) ;
                  exit A ;
                endif ;
            if trouve then exit EX ; endif ;
          endfor ;
        endif ;
      if trouve then exit O ; endif ;
    endfor ;
  if trouve then exit A ; endif ;
endfor ;
```

ALg-3.2 Algorithme conforme à DBMS du module-1

b) Conforme à RDB :

```
input ( XTITRE, XAN , XNOMAU ) ;
trouve:= false ;
for O := OUVRAGE (( : TITRE = XTITRE ) and ( : ANNEE = XAN ) and
                  ( : NUMOU in NUMOU ( : OUVAUT ( :NUMAU
                      in NUMAU ( : AUTEUR ( : NOMAU
                                          = XNOMAU))))) do
    for EX := EXEMPLAIRE ( ( : NUMEX not in NUMEX ( : EMPRUNT ))
                  and ( : NUMOU = NUMOU ( : O ))) do
        trouve := true ;
        PRINT NUMEX ( : EX ) ;
        exit EX ;
    endfor ;
    if TROUVE then exit O endif ;
endfor;
```

ALG-3.3 Algorithme conforme à RDB du module-1

Si le problème était assez simple et que le SGBD offrait un DML qui est un langage de très haut niveau, l'algorithme prédicatif serait déjà probablement exécutable. On ne fait donc pas cette transformation. Prenons par exemple l'application 2 (voir l'algorithme prédicatif à 3.3.4):

Il est évident que l'algorithme prédicatif de l'application 2 n'est pas nécessairement transformé car c'est une requête simple avec condition évaluable.

Nous signalons que le SGBD RELATIONNEL assure l'optimisation des accès. Donc, la dérivation des algorithmes conformes à partir des algorithmes prédicatifs est souvent immédiate.

Tous les algorithmés conformes se trouvent dans l'annexe-3.

3.3.3 La traduction des programmes d'application

Cette phase a pour objet de produire les programmes/COBOL exécutables. Chaque algorithme conforme attaché à un module doit maintenant être rédigé dans le COBOL et le DML de son propre SGBD.

Nous signalons qu'il y a une différence fondamendale entre DML/CODASYL et DML/RELATIONNEL en ce qui concerne la façon de manipuler les données de la BD. Le DML/RELATIONNEL est doté d'une structure de boucle sur des séquences de données. Ce n'est pas le cas pour DML/CODASYL. Il accède à la BD de façon "one-record-at-a-time", c'est-à-dire, accès ponctuel. Sur ce point, nous pouvons déjà sentir que la traduction de boucle "for" de l'algorithme conforme en programme/COB est plus facile en RDB qu'en DMBS.

- Programme/DBMS/COB

Des règles systématiques de traduction de LDA en COBOL-DML sont proposées dans [HAIN 81]. A titre d'exemple, nous développons une boule d'accès selon la primitive d'accès ponctuel.

Nous simplifions l'alg-3.2 comme suit:

```
for B := BIBLIO do
    input ( XT, XAN, XAU);
    trouve := false;
    for AU := AUTEUR (: NOM = XAN) do
        trouve-ouvrage;
        if trouve then exit AU endif;
    endfor;
endfor.
```

La traduction est:

```
:
:
PROCEDURE DIVISION.
  BIBLIO-BEGIN.
     START-TRANSACTION  (* READY en DBMS *).
      input ( XT, XAN, XAU ).
      MOVE 0 TO TROUVE;
      PERFORME TROUVE-AU THRU AU-EXIT.
      COMMIT.
      STOP RUN.

  TROUVE-AU.
      FETCH FIRST AUTEUR WITHIN SET-AUTEUR AT END
             GO TO AU-EXIT.
      IF NUMERO OF AUTEUR = XAU
         PERFORM TROUVE-OUVRAGE
         IF TROUVE = 1
            GO TO AU-EXIT
         ELSE
             MOVE 0 TO FIN-SET
             PERFORM TROUV-NEXT-AU THRU NEXT-AU-EXIT.

  AU-EXIT.
      EXIT.

  TROUV-NEXT-AU .
       FETCH NEXT AUTEUR WITHIN SET-AUTEUR AT END
              MOVE 1 TO FIN-SET
          GO TO NEXT-AU-EXIT.
    IF NUMERO OF AUTEUR = XAU
       PERFORM TROUVE-OUVRAGE
       IF TROUVE = 1
          MOVE 1 TO FIN-SET
          GO TO NEXT-AU-EXIT.
 NEXT-AU-EXIT.
      EXIT.
```

- Programme/RDB/COB

RDB admet la structure de boucle, la jointure de plusieurs relations et la condition filtré de sélection de données. La traduction est par conséquent souvent immédiate. par exemple, dans l'alg-3.3 la boucle

```
:
:
input ( XTITRE, XAN, XAU );
for O := OUVRAGE ((:TITRE = XTITRE) and
                  (:ANNEE = XAN) and (OUVAUT: AUTEUR
                  (: NOMAU = XAU ))) do
        séquence d'instructions
    endif;
endfor;
```

peut se traduire:

```
&RDB& FOR O IN OUVRAGE CROSS
-             OU IN OUVAUT CROSS
-             A IN AUTEUR  WITH
-             O.TITRE = XT AND
-             O.NUMAU = A.NUMAU AND
-             O.ANNEE = XAN AND
-             A.NOMAU = XAU
          séquence d'instructions
&RDB&    END-FOR;
```

Les programmes/DBMS/COB et /RDB/COB se trouvent en annexe-4.

Remarque:

Dans l'annexe-4, nous ne donnons qu'une partie des nos programmes: à savoir, les numéros d'applications 1, 6, 8, 9, 18, 21, 25. Pour le reste, le lecteur peut s'adresser au bureau de monsieur J-L HAINAUT, à l'Institut d'Informatique de NAMUR.

3.3.4 DESCRIPTION ET ALGORITHME PREDICATIF DES APPLICATIONS

Dans ce paragraphe, nous donnons toutes les descriptions et tous les algorithmes prédicatifs des applications. En tête de chaque application, nous trouvons le nom de l'application et sa typologie correspondante (cfr. Figure-2.1).

A) ACCES AUX ARTICLES

1) APPLICATION1 : Consultation , élémentaire , de masse , séquentiel .

Description:
Lister tous les numeros d' OUVRAGE .

Algorithme-prédicatif :

```
for O : = OUVRAGE do
    print NUMAU ( : O ) ;
endfor ;
```

2) APPLICATION2 : Consultation , élémentaire , transactionnel , par clé

Description :
Etant donné un numéro d' OUVRAGE , fournir son titre .

Algo-préd :

```
input ( XNUMOU ) ;
for O : = OUVRAGE ( : NUMOU = XNUMOU ) do
    print TITRE ( : O ) ;
endfor ;
```

3) APPLICATION3 : Consultation , élémentaire , de masse , séq. filtré
Description :
Donner une liste d' ouvrages qui sont parus dans l' année 1970 .

Algo-préd :

```
input ( 1970 );
for O : = OUVRAGE ( : ANNEE = 1970) do
    print O
endfor;
```

4) APPLICATION4 : Consultation, moy. complexe, de masse, séquetiel.

Description:
Fournir une liste d'ouvrages qui ont au moins un exemplaire.

Algo-préd:

```
for O : = #1 OUVRAGE ( OE : EXEMPLAIRE ) do
    print O ;
endfor;
```

5) APPLICATION5 : Consultation, moy. complexe, de masse, séquentiel.

Description :
Fournir une liste d'ouvrages qui ont au moins un auteur.

ALgo-préd :

```
for O : = #1 OUVRAGE ( OUVAUT : AUTEUR ) do
    print O;
endfor ;
```

6) APPLICATION6 : Consultation, moy.complexe, transactionnel, par chemin.

Description :
Fournir le(s) ouvrage(s) qui est (sont) écrit(s) par un auteur donné (étant donné le numéro d' AUTEUR).

Algo-préd :

```
input ( XNUMAU ) ;
for O : = OUVRAGE ( OUVAUT : AUTEUR ( : NUMAU = XNUMAU ) ) do
    print O ;
endfor ;
```

7) APPLICATION7 : Idem que l'application6

Description :
Idem que l'application6. Mais on donne un nom d'AUTEUR.

Algo-préd :

```
input (XNOMAU) ;
for  O : = OUVRAGE (OUVAUT : AUTEUR (:NOMAU = XNOMAU)) do
    print O ;
endfor ;
```

8) ALLICATION8: Consultation, moy.complexe, transactionnel, séq. filtre et par chemin

Description :

Idem que l'application6. Mais on donne une ORIGINE d'AUTEUR (ORIGINE n'est pas une clé d'accès).

Algo-préd :

```
input ( XORIGINE ) ;
for O : = OUVRAGE (OUVAUT : AUTEUR (: ORIGINE = XORIGINE)) do
    print O ;
endfor ;
```

9) APPLICATION9 : Consultation, complexe, transactionnel, par clé.

Description :

Etant donné le titre, l'année d'édition et le nom d'un auteur d'un ouvrage, fournir le numéro d'un exemplaire disponible.

Algo-préd :

```
input ( XTITRE, XAN, XNOMAU ) ;
for EX := #1 EXEMPLAIRE ( ( EXE : O EMPRUNT ) and
                          ( OE : OUVRAGE ( ( : TITRE = XTITRE )
                        and ( : ANNEE = XAN ) and
                          ( OUVAUT: AUTEUR ( : NOMAU = XNOMAU
                                            ) ) ) ) do
    print NUMEX ( : EX ) ;
  endfor;
endFor .
```

10) APPLICATION10 : Consultation, complexe, de masse, séquentiel.

Description :

Etant donné un numéro d'un ouvrage, fournir toutes les informations qui y sont liées.

Algo-préd :

```
input ( XNUMOU ) ;
for O := OUVRAGE ( : NUMOU = XNUMOU ) do
    print O ;
    for A := AUTEUR ( OUVAUT : O ) do
        print A ;
    endfor ;
    for EX := EXEMPLAIRE ( OE : O ) do
        print E ;
        for EM := EMPRUNT ( EXE : EX ) do
            print EM ;
            for EP := EMPRUNTEUR( EMP : EM ) do
                print EP ;
            endfor ;
        endfor ;
        for EA := EMPRUNTARCH ( EXEA : EX ) do
            print EA ;
        endfor ;
    endfor ;
endfor ;
```

11) APPLICATION11: Idem que l'application10

Description :
item que l'application10 . Mais on donne un numéro d'auteur . C-à-d que l'on accède à la BD à partir d' AUTEUR .

Algo-préd :
La même manière que l' application10 .

12) APPLICATION 12
Description :
Idem que application10 . Mais on donne un numéro d'exemplaire (à partir d'EXEMPLAIRE) .

13) APPLICATION13
Description :
Idem que l'application10 . Mais on donne un numéro d'emprunteur (à partir d'EMPRUNTEUR) .

B) MISE A JOUR DE LA BASE DE DONNEES

14) APPLICATION14: Mise à jour, élémentaire, transactionnel, par clé.
Description :
Etant donné un numéro d' emprunteur, changer la valeur de son adresse.

Algo-préd :

```
input ( XNUMEM , XADR ) ;
for E := EMPRUNTEUR ( : NUMEM = XNUMEM ) do
    modify E ( : ADRESSE = XADR ) ;
endfor ;
```

15) APPLICATION15 : Idem que l'application14
Description :
Etant donné un numéro d' un auteur, modifier le nom de l'auteur.

Algo-préd :

```
input ( XNUMAU , XNOMAU ) ;
for A := AUTEUR ( : NUMAU = XNUMAU ) do
    modify A ( : NOMAU = XNOMAU ) ;
endfor ;
```

16) APPLICATION16 : Mise à jour, moy.complexe, transactionnel, par clé

Description :
Etant donné un numéro d'un ouvrage,on change la valeur de ses motclés.

Algo-préd :

```
input ( XNUMOU, XMOTCLE ) ;
for O := OUVRAGE (: NUMOU = XNUMOU) do
    modify O (: MOTCLE = XMOTCLE) ;
endor;
```

17) APPLICATION17 : Mise à jour, moy. complexe, transactionnel, par clé.

Description :
Ajouter un exemplaire à un ouvrage.

```
Algo-préd :
        input ( XNUMOU ) ;
        for O := OUVRAGE ( : NUMOU = XNUMOU ) do
            input ( XEXEMP ) ;
            if NUMEX ( : XEXEMP ) not in NUMEX ( : EXEMPLAIRE ) then
              create EX := EXEMPLAIRE ( (: NUMEX = NUMEX(: XEXEMP )) and
                          (:LOCALISATION = LOCALISATION(:XEXEMP)) and
                          ( OE : O ) ) ;
            endif;
        endfor ;
```

18) APPLICATION18: Idem que l'application 17

Description :
Supprimer un exemplaire d' un ouvrage .

```
Algo-préd :
        input ( XNUMOU ) ;
        for EX := 1 EXEMPLAIRE (( OE : OUVRAGE
                                  ( : NUMOU = XNUMOU)) and
                                  (EXE: O EMPRUNT)) do;
            delete EX;
        endfor;
```

19) APPLICATION19: Idem que application 17

Description :
Enregistrer un ouvrage avec 3 auteurs et 3 exemplaires (les auteurs et les exemplaires sont contenus dans les listes LIAUT[i] et LIEXE[i]).

```
Algo-préd :
        input ( XOUVRAGE ) ;
        if NUMOU(: XOUVRAGE) not in NUMOU (: OUVRAGE) then
            create O : = OUVRAGE((:NUMOU = NUMOU(:XOUVRAGE)) and
                                 (:TITRE = TITRE(:XOUVRAGE)) and
                                 (:ANNEE = ANNEE(:XOUVRAGE)) and
                                 (:EDITEUR = EDITEUR(:XOUVRAGE)));
            for i := 1 to 3 do
                input ( LIAUT[i], LIEXE[i]) ;
                if NUMAU(:LIAUT[i]) not in NUMAU(:AUTEUR) then
                  create A := AUTEUR((:NUMAU = NUMAU(:LIAUT[i])) and
                                     (:NOMAU = NOMAU (:LIAUT[i])) and
                                     (:ORIGINE = ORIGINE(:LIAUT[i])) and
                                     ( OUVAUT : O ));
                endif ;
                 if NUMEX (LIEXE[i] not in NUMEX(:EXEMPLAIRE) then
                  create EX := EXEMPLAIRE((:NUMEX = NUMEX(:LIEXE[i])) and
                          ((:LOCALISATION = LOCALISATION(:LIEXE[i])) and
                          ( OE : O ));
                 endif ;
            endfor ;
        endif ;
```

20) APPLICATION20: Mise à jour, complexe, transactionnel, par clé.

Description :
Supprimer un ouvrage avec ses auteurs et ses exemplaires.

```
Algo-préd :
    input ( XOUVRAGE ) ;
    for O := OUVRAGE( (:NUMOU = NUMOU(:XOUVRAGE) and (DE: EXEMPLAIRE)) do
        for A := AUTEUR ( OUVAUT : O ) do
         delete A ;
         endfor ;
         for EX := EXEMPLAIRE (( OE : O ) and (EXE: O EMPRUNT)) do
            delete EX ;
         endfor ;
         delete O;
     endfor ;
```

21) APPLICATION21: Mise à jour, complexe , transactionnel, par clé.

Description :
Enregistrer la restitution d' un exemplaire emprunté de numéro donné.

```
Algo-préd :
        input ( XEXEMP ) ;
        for E := EMPRUNT ( EXE : EXEMPLAIRE ( : NUMEX = XEXEMP )) do
            create EA := EMPRUNTARCH ( ( EXEA :EXEMPLAIRE (EXE : E ))
                                    and ( EMPA : EMPRUNTEUR ( EMP: E ))
                                and ( : DATE-DEBUT =DATE-DEBUT ( :E ))
                                and ( : DATE-FIN = DATE-DU-JOUR ) ) ;
            delete E ;
        endfor ;
```

22) APPLICATION22: Mise à jour, moy. complexe, de masse, par clé (ou séquentiel).

Description :
Etant donné une liste LISAN (i) d' années, modifier la valeur des éditeurs d' OUVRAGE et ajouter un auteur.

```
Algo-préd :
      input ( LISAN( 1...N) ) ,
      for i = 1 to N do
          for O := OUVRAGE ( : ANNEE = LISAN(i) ) do
              input ( XEDITEUR  , XAUTEUR ) ;
              modify O ( : EDTEUR = XEDITEUR ) ;
              if  NUMAU(:XAUTEUR) not in NUMAU(:AUTEUR) then
                  create A := AUTEUR( (: NUMAU = NUMAU(:XAUTEUR)) and
                                      (:NOMAU = NOMAU(:XAUTEUR)) and
                                      (:ORIGINE = ORIGINE(:XAUTEUR)) and
                                      ( OUVAUT : O ) );
              endif ;
          endfor ;
      endfor ;
```

23) APPLICATION23: Mise à jour, complexe, de masse, par clé.

Description :
Supprimer toutes les informations concernant les emprunteurs dont les numéros sont donnés dans la liste LISNUM[i].

```
Algo-préd :
        input ( LISNUMEM( 1...N) ) ;
        for i = 1 to N do
            for EM := EMPRUNTEUR ( : NUMEM = LISNUM(i) ) do
                for E := EMPRUNT ( EMP : EM ) do
                    delete E;
                endfor ;
                for EA := EMRUNTARCH ( EMPA : EM ) do
                    delete EA ;
                endfor ;
                delete EM ;
            endfor ;
        endfor ;
```

C) BATCH

24) APPLICATION24: Consultation, complexe, de masse, séquentiel

Description :
Lecture de toute la base de données à partir d' OUVRAGE .

```
Algo-préd :
        for O := OUVRAGE do
            print O ;
            for A := AUTEUR ( OUVAUT : O ) do
                print A ;
            endfor ;
            for EX := EXEMPLAIRE ( OE : O ) do
                print E ;
                for EM := EMPRUNT ( EXE : EX ) do
                    print EM ;
                    for EP := EMPRUNTEUR( EMP : EM ) do
                        print EP ;
                    endfor ;
                endfor ;
                for EA := EMPRUNTARCH ( EXEA : EX ) do
                    print EA ;
                endfor ;
            endfor ;
        endfor ;
```

D) PROGRAMME DE CHARGEMENT

25) APPLICATION25: Mise à jour, complex, de masse, séquentiel.

Description:

Charger les données dans la BD BIBLIO selons les quantifications prédéfinies des données (voir le paragraphe 3.4.2) et les répartitions des données (voir l'annexe-2).

Alogorithme:
Cfr. annexe-2.

3.4 GENERATION DES DONNEES DE TEST

Une fois les définitions des bases de données rédigées, il nous faut les créer physiquement et y charger les données.

Nous ne nous préoccuperons pas ici de la détermination des paramètres pour la création des BDs. Nous signalons seulement que le critère, dans ce cas, est de rendre la base de données la plus performante possible selon les caractéristiques de chaque SGBD. Nous trouverons les paramètres de création de bases de données en annexe-2.

Notre but est d'avoir les mêmes données pour la BD/DBMS et la BD/RDB. Ceci est réalisé grâce aux programmes de chargement automatique. Ceux qui s'intéressent au fonctionnement du programme en trouveront une description détaillée en annexe-2.

3.4.1 Les caractèristiques du programme de chargement

Dans le programme de chargement, il n'y a aucun fichier COBOL qui contient les données à charger. Toutes les données sont générées par les programmes eux-mêmes. Par exemple, pour un nom d'AUTEUR d'une longueur de 30 caractères, nous avons dans le programme une procédure de permutation qui génère tous les noms d' AUTEUR. Trois mérites doivent être reconnus au chargement automatique. Premièrement, il peut refléter adéquatement la réalité: comme la génération de la série de caractères, les chiffres numériques sont encore plus faciles à générer. Deuxièmement, nous pouvons contrôler la taille de la base de données. En effet, nous avons prédéfini les éléments de quantification et les répartitions des données avant de charger ces données dans la base de données (les répartitions détaillées des données se trouvent en annexe-2). Ceci facilite les calculs. Troisièmement, nous gagnons du temps parce qu'il n'est pas possible de créer manuellement une base de données assez volumineuse pour l'expérience en temps limité.

3.4.2 La définition des éléments de quantification

Afin que les programmes de chargement chargent les mêmes données, nous prédéfinissons la distribution des données. De plus, afin que les mesures et la comparaison ultérieures soient significatives et "réalistes", nous avons créé deux bases de données pour un même SGBD d'un système d'information BIBLIOTHEQUE. L'une à la taille réduite, l'autre à la taille relativement grande. Nous donnerons les éléments quantitatifs des bases de données suivants:

§1: Eléments de quantification pour la BD de taille réduite

- Il ya 1000 ouvrages, 500 auteurs, 2.100 exemplaires, et 100 emprunteurs;

- Il y a de 0 à 5 auteurs par ouvrage, avec une moyenne de 2. Nous fixons 5% d'ouvrages sans auteurs;

- Il y a de 0 à 7 exemplaires par ouvrage;

- Il y a de 0 à 5 valeurs de mot-clé par ouvrage;

- Il y a 380 exemplaires qui ont emprunté de 1 à 5 fois et 80 emprunts sont en cours;

- Il y a 1.950 occurrences de OUVAUT et MCOUV.

§2: Eléments de quantification pour la BD de grande taille

- Il y a 10.000 ouvrages, 3.000 auteurs, 25.860 exemplaires, 300 emprunteurs.

- Il y a de 0 à 8 auteurs par ouvrage, on fixe 15% d'ouvrages sans auteurs.

- Il y a de 0 à 20 exemplaires par ouvrage.

- Il y a de 0 à 8 valeurs de mot-clé par ouvrage.

- Il y a 2.000 exemplaires qui ont été empruntés, 200 emprunts sont en cours.

- Il y a 14.650 occurrences d'OUVAUT et MCOUV.

3.4.3 La construction des programmes de chargement

Nous suivons la même démarche que celle décrite dans le paragraphe pécédent. C'est-à-dire que nous élaborons d'abord un algorithme de niveau 4 qui est attaché à un seul module d'accès. Nous le transformons ensuite directement en deux programmes exécutables : l'un fonctionne sur DBMS et l'autre sur RDB.

Lorsque les deux programmes/COBOL sont traduits de façon systématique à partir d'un même algorithme et de la même répartition de données, nous pouvons nous assurer que les deux bases de données DBMS et RDB contiendront les mêmes données. C'est-à-dire que les états des deux BDs seront équivalents, après que les programmes de chargement aient fonctionné.

Nous ne donnons pas ici l'algorithme de chargement et les programmes correspondants. Ils se trouvent respectivement à l'annexe-2 et l'annexe-4.

CHAPITRE 4
MESURES ET RESULTATS

Une fois fixé le plan d'expérience qui a pour objectif de définir les paramètres de mesures, un ensemble de mesures divisé en plusieurs sessions peut être commencé. Il a pour but de former les tableaux de résultats. Les données sont, soit obtenues à l'aide des facilités du système, soit rassemblées directement à partir des manuels, soit calculées en fonction des caracteristiques des paramètres de mesures.

Le critère principal est de recouvrir tous les paramètres à mesurer.

Avant de commencer cette série de mesures, il faudrait en premier lieu en construire le plan de réalisation suivant la nature du type d'application observée et le critère de la mesure.

Ceci exige de prendre en considération les trois points suivants:

- le choix d'une méthode de mise en oeuvre de mesures;

- la précision des conditions expérimentales des mesures. Surtout par exemple, pour évaluer la performance du SGBD (voir 4.4). Lorsqu'un programme de l'application donnée fonctionne en mode intéractif dans un environnement en temps partagé, le moment, la durée de mesures et le nombre d'exécutions du programme sont des paramètres qui conditionnent les résultats de l'expérience;

- la représentation des résultats de mesures.

Ce chapitre est composé de cinq paragraphes qui présentent les cinq sessions de mesures correspondant respectivement aux cinq critères de comparaison.

4.1 REPRESENTATION DE LA STRUCTURE DE DONNEES

Ce paragraphe a pour objet d'élaborer, selon le point 2.2.2, les résultats en choisissant des exemples représentatifs.

Première partie: L'ASPECT LOGIQUE

4.1.1 La représentation d'un type d'entité

Soit un type d'entité AUTEUR qui a les attributs:
NUMAU, NOMAU, ORIGINE.
où NUMAU est un identifiant d'AUTEUR.

- La représentation graphique (les deux modèles DBMS et RDB sont équivalents):

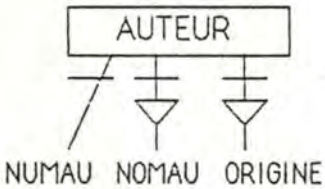


FIGURE-4.1 Représentation graphique d'un type d'entité

- Représentation logique:

DBMS	RDB
RECORD NAME IS AUTEUR WITHIN FAUTEUR ITEMS NUMAU TYPE IS DECIMAL 6 ITEMS NOMAU TYPE IS CARACTER 30 ITEMS ORIGINE TYPE IS CARACTER 30	DEFINE FIELD NUMAU DATATYPE IS SIGNED LONGWORD DEFINE FIELD NOMAU DATATYPE IS TEXT SIZE IS 30
SET NAME IS SYSTAUTEUR OWNER IS SYSTEM MEMBER IS AUTEUR INSERTION IS AUTOMATIC RETENTION IS FIXED ORDER IS SORTED BY ASCENDING NUMAU DUPLICATES ARE NOT ALLOWED.	DEFINE FIELD ORIGINE DATATYPE IS TEXT SIZE IS 30 DEFINE RELATION AUTEUR NUMAU NOMAU ORIGINE END AUTEUR RELATION.

TAB-4.1 Représentation logique d'un type d'entité

- Représentation physique

DBMS	RDB
RECORD NAME IS AUTEUR PLACEMENT IS CLUSTERED VIA SYSTAUTEUR SET NAME IS SYSTAUTEUR MODE IS INDEX NODE SIZE IS 400 BYTES	DEFINE INDEX AU-NUMAU FOR AUTEUR DUPLICATES ARE NOT ALLOWED NUMAU. END AU-NUMAU INDEX.

TAB-4.2 Représentation physique d'un type d'entité

Remarque

RDB ne permet pas la séparation entre description logique et description physique de la BD. Les définitions de TAB-4.1 et TAB-4.2 pour RDB doivent donc se trouver dans un même texte.

4.1.2 La représentation d'un type d'association binaire:

1) Une association entre deux types d'entité différents

soit X , Y deux types d'entité dotés respectivement des identifiants idx, idy.

- Représentation graphique (voir la figure-4.2):

- Représentation logique:

En DBMS : cfr. TAB-4.1;

En RDB : cfr. TAB-4.1 pour la définition de RELATION.

La définition de contrainte est exprimée comme suit:

```
DEFINE CONSTRAINT nom-contrainte
  FOR RX IN X
  REQUIRE ANY RY IN Y WITH
    RX.idx = RY.idy.
```

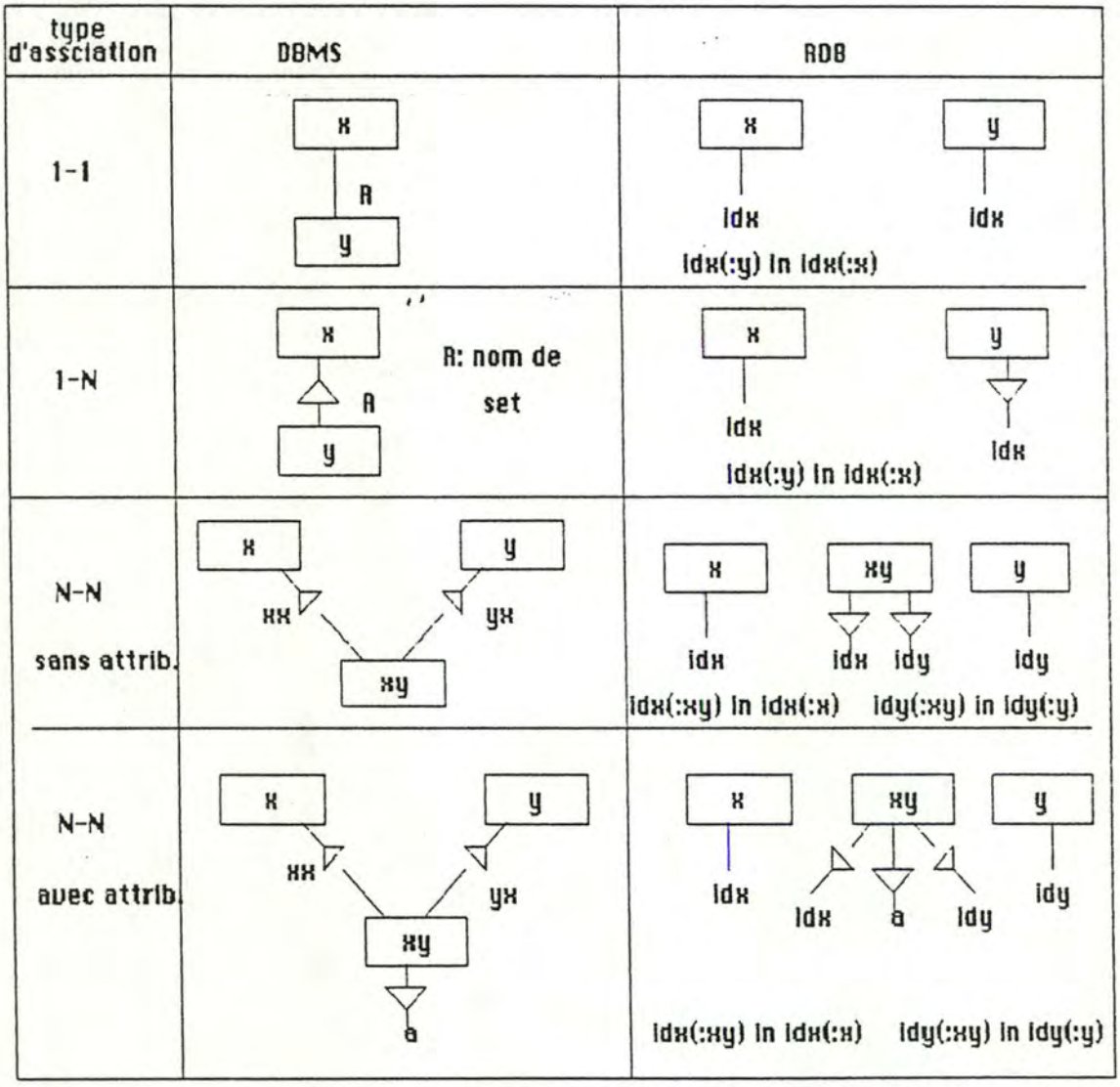


FIGURE-4.2 Représentation graphique d'association

2) Une association entre un type d'entité et lui-même

- Représentation graphique (voir figure_4.3):

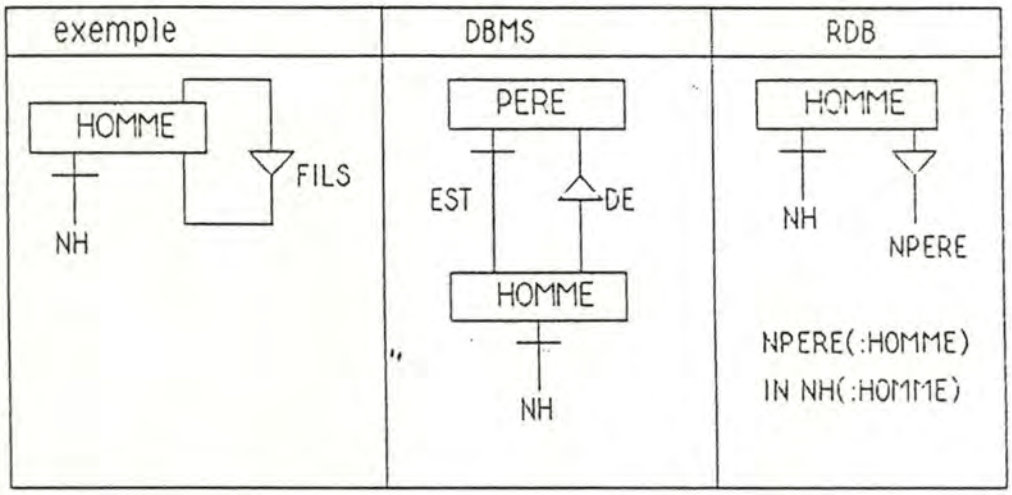


FIGURE-4.3 Représentation graphique d'une association récursive

- Représentation logique:

DBMS	RDB
RECORD NAME IS PERE WITHIN AREA-1	DEFINE FIELD NH DATATYPE IS SIGNED LONGWORD
RECORD NAME IS HOMME WITHIN AREA-1 ITEM NH TYPE IS DECIMAL 6	DEFINE FIELD NPERE DATATYPE IS SIGNED LONGWORD
SET NAME IS EST OWNER IS PERE MEMBER IS HOMME INSERTION IS AUTOMATIC RETENTION IS MANDATRY	DEFINE RELATION HOMME NH NPERE END HOMME RELATION
SET NAME IS DE OWNER IS PERE MEMBER IS HOMME INSERTION IS AUTOMATIC RETENTION IS MANDATRY	DEFINE CONSTRAINT nh-npere FOR RH IN HOMME REQUIRE ANY RH IN HOMME WITH RH.NPERE = RH.NH.

TAB-4.5 Représentation logique d'une association récursive

- Représentation logique: Cfr TAB-4.2.

4.1.3 La représentation d'un type d'association ternaire:

- Représentation graphique:

Soient

- x y z trois types d'entité dotés respectivement des identifiants idx, idy, idz ,

- A un attribut.

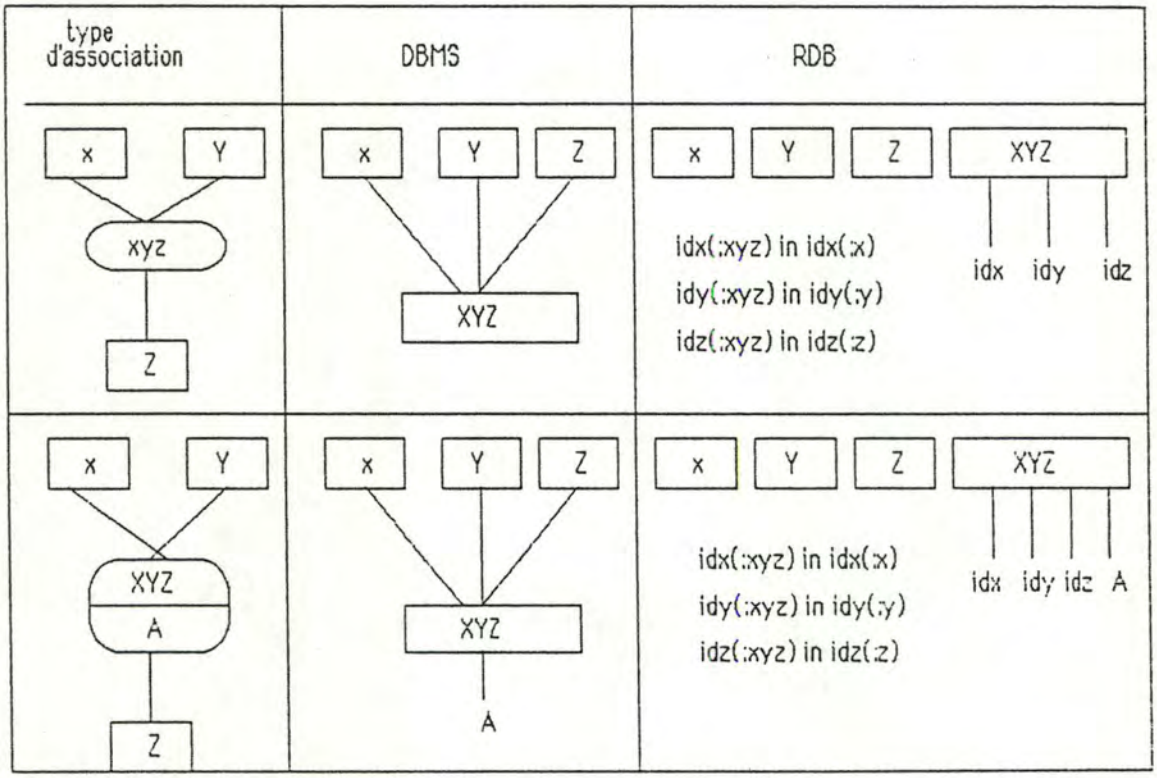


FIGURE-4.4 Représentation d'association ternaire

4.1.4 La représentation des attributs:

- Représentation graphique (voir figure-4.5):

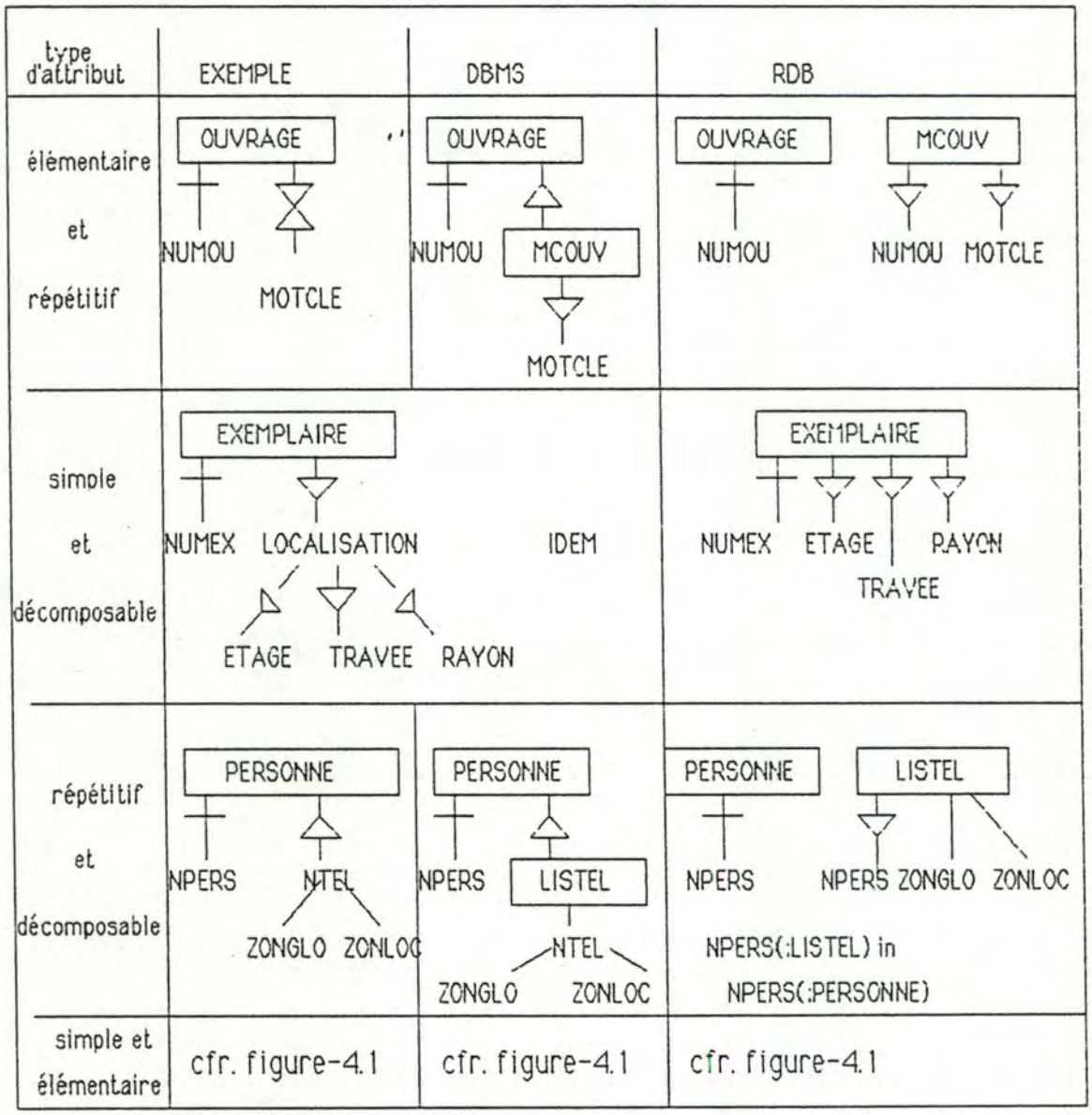


FIGURE-4.5 Représentation des attributs

4.1.5 La représentation d'un identifiant:

- Representation logique (voir TAB-4.6)

identifiant	DBMS	RDB
un attribut	cfr. TAB-4.1	cfr. TAB-4.2
N attributs Soit un type d'entité ENT avec a et b qui jouent le rôle d'identifiant	SET NAME IS SYSTENT OWNER IS SYSTEM MEMBER IS ENT : ORDER IS SORTED BY ASCENDING a b DUPLICATED ARE NOT ALLOWED Note: En DBMS un identifiant doit être défini avce clause SORTED. En effet, c'est une clé indexée.	DEFINE INDEX a-b FOR ENT DUPLICATES ARE NOT ALLOWED a. b. END a-b INDEX
Plusieurs identifiants pour un type d'entité ENT. Soit a et b les identifiants de l'ENT	SET NAME IS SYSENT OWNER IS SYSTEM MEMBER IS ENT : ORDER IS SORTED BY ASCENDING a DUPLICATED ARE NOT ALLOWED SET NAME IS SYSENT1 OWNER IS SYSTEM MEMBER IS ENT : ORDER IS SORTED BY ASCENDING b DUPLICATED ARE NOT ALLOWED	DEFINE INDEX INEX-a FOR ENT DUPLICATES ARE NOT ALLOWED a END INDEX-a INDEX DEFINE INDEX INDEX-b FOR ENT DUPLICATES ARE NOT ALLOWED b END INDEX-b INDEX.

TAB-4.6 Représentation logique d'un identifiant

4.2 POSSIBILITE ET RESTRICTION DE MODIFICATION DES STRUCTURES DES DONNEES

Dans cette section nous nous reporterons aux manuels pour observer les possibilités et les restrictions des changements des structures des données par rapport aux programmes d'application.

Le résultat est représenté dans le TAB-4.7:

type de changement	possibilité et restriction	
	DBMS	RDB
ajouter un item	Oui. L'item ajouté doit suivre les items existants dans l'ordre	Oui
ajouter un type d'entité	Oui. Le nouveau record doit suivre les records existants dans l'ordre.	Oui
ajouter une association entre deux types entité	Oui. La définition du set ajouté doit être définie après la définition de sets existants	Oui
régles de supp. des inform. du schéma de la BD	On ne peut pas supprimer les définitions de records, items, sets, AREA, ou clé d'accès du schéma et du STORAGE_SCHEMA.	On peut supprimer les définitions de relations et des FIELDS. Même les vues et les index.
régles de modi. des inform. du schéma de la BD	- L'ordre d'un set en mode SORTED ne peut être changé. Les autres sont permis. - On peut changer le PLACEMENT d'un set dans le STORAGE_SHEMA.	On peut changer un field qui est défini dans un index ou une vue; on ne peut ni changer ni ajouter un VALID IF dans un field
changer un item en une clé d'accès	Possible. Mais affecte le progr. d'application.	Oui

TAB-4.7 Les possibilités de changement des structures des données

4.3 COMPTAGE SIMPLE DU DEGRE ALGORITHMIQUE DES PROGRAMMES

Selon les paramètres à mesurer (voir 2.2.3), cette session de mesures consiste en un comptage simple des valeurs à partir des programmes d'application qui sont déjà codés et de leurs algorithmes conformes.

A titre d'exemple, nous développons l'algorithme prédicatif de l'application 8 (voir le point 3.3.4) en deux algorithmes conformes et en deux programmes exécutables.

4.3.1 Au niveau de l'algorithme conforme

Nous développons d'abord l'algorithme prédicatif en algorithmes conformes.

```
input ( XORIGINE ) ;
for A := AUTEUR do
    if ORIGINE( : A ) = XORIGINE then
      for OU := OUVAUT ( AO : A ) do
          for O := OUVRAGE ( OO : OU ) do
              print O ;
          endfor ;
      endfor ;
    endif ;
endfor ;
```

ALG-4.1 Algorithme de l'application 8 conforme à DBMS

```
input ( XORIGINE ) ;
for O := OUVRAGE ( : NUMOU in NUMOU ( : OUVAUT ( : NUMAU
                    in NUMAU ( : AUTEUR ( : ORIGINE =
                                XORIGINE ) ) ) ) ) do
    print O ;
endfor ;
```

ALG-4.2 Algorithme de l'application 8 conforme à RDB

Il est facile de se rendre compte qu'il y a 4 structures "if" et "for" (NSIFOR) dans l'ALG-4.1 et qu'il n'y en a qu'une dans l'ALG-4.2.

4.3.2 Au niveau du programme exécutable

Nous allons maintenant développer cette application en deux programmes exécutables:

a) En DBMS

```
        :
        :
    WORKING-STORAGE,SECTION.
     01 FIN-SET   PIC 9.
     01 TROUVE    PIC 9.
     01 I         PIC 9.

    PROCEDURE DIVISION.
     100-BEGIN.
1          READY FOUV FAUTEUR PROTECTED RETRIEVAL. ( 1 )
2          PERFORM TROUVE-AU.
3          COMMIT.                                 ( 2 )
4          STOP RUN

   TROUVE-AU.
5          FETCH FIRST AUTEUR WITHIN SYSTAUTEUR.     (3 )
6          IF ORIGINE OF AUTEUR = "NAMUR"
            PERFORM FET-OUVAUT.
7          MOVE 0 TO TROUVE.
8          PERFORM NEXT-AU THRU NEXT-AU-EXIT UNTIL TROUVE = 1.

  NEXT-AU-EXIT.
9           EXIT.

  FETCH-OUVAUT.
10          FETCH FIRST OUVAUT WITHIN AO.             (4 )
11          PERFORM FET-OU.
12          MOVE 0 TO FIN-SET.
13          PERFORM NEXT-OUVAUT THRU NEXT-EXIT UNTIL FIN-SET = 1.

  NEXT-OUVAUT.
14           FETCH NEXT OUVAUT WITHIN AO AT END        ( 5 )
15                MOVE 1 TO FIN-SET
16                GO TO NEXT-EXIT.
17           PERFORM FET-OU.

  NEXT-EXIT.
18           EXIT.

  FET-OU.
19         FETCH OWNER WITHIN OO.                      ( 6 )
20         DISPLAY NUMOU, TITRE, ANNEE, EDITEUR.
```

<u>PRG-4.1 Programme/DBMS/COB de l'application 8</u>

B) En RDB:

```
      :
      :
    WORKING-STORAGE SECTION.
     01 I          PIC 9.
     01 NUMOU1     PIC 9.
     01 TITRE1     PIC X(30).
     01 ANNEE1     PIC X(4).
     01 EDITEUR1   PIC X(30).

    PROCEDURE DIVISION.
    100-BEGIN.
1     &RDB& START-TRANSACTION                    ( 1 )
2           PERFORM BIBLIO.
3     &RDB& COMMIT                               ( 2 )
4     &RDB& FINISH                               ( 3 )
5           STOP RUN.

    BIBLIO.
6   &RDB& FOR A IN AUTEUR CROSS                  ( 4 )
    &RDB&     O IN OUVRAGE CROSS OU IN OUVAUT WITH
    &RDB&       A.ORGINE = "NAMUR"
    &RDB&       OU.NUMAU = A.NUMAU AND
    &RDB&       O.NUMOU = OU.NUMOU
7   &RDB&       GET NUMOU1 = O.NUMOU;            ( 5 )
    &RDB&           TITRE1  = O.TITRE;
    &RDB&           ANNEE1 = O.ANNEE;
    &RDB&           EDITEUR1 = O.EDITEUR
    &RDB&       END-GET
8           DISLAY NUMOU1, TITRE1, ANNEE1, EDITEUR1
    &RDB& END-FOR.
```

PRG-4.2 Programme/RDB/COB de l'application 8

Dans les programmes ci-dessus, nous marquons un chiffre à gauche de chaque instruction exécutable. Donc, le nombre total d'instructions exécutables est de 20 pour le PRO-4.1 et de 8 pour le PRG-4.2.

Nous marquons un chiffre dans la paranthèse à droite de chaque instruction de DML. Il est clair que le nombre d'instructions de DML est de 6 pour le PRG-4.1 et de 5 pour le PRO-4.2.

Quand au nombre de déclarations spécifiques de zone de données (NDZDS) dans les programmes, nous le comptons dans le "WORKING-STORAGE SECTION" de chaque programme. On en compte 3 pour le PRG-4.1 et 5 pour le PRG-4.2.

Les résultats de cette session de mesures se trouvent dans le TAB-4.8.

numéro d'applic	NSIFOR		NDZDS		NTIE		NINSD		PROP	
	DBMS	RDB	DBMS	RDB	DBMS	RDB	DBMS	RDB	DBMS	RDB
1	1	1	2	2	14	10	3	4	0.21	0.4
2	1	1	4	5	12	13	3	4	0.25	0.31
3	2	1	2	5	15	11	4	5	0.27	0.45
4	2	1	2	5	14	11	6	5	0.43	0.45
5	2	1	2	5	16	11	6	5	0.38	0.45
6	3	1	2	5	19	11	6	5	0.32	0.45
7	3	1	2	5	19	12	6	6	0.32	0.5
8	4	1	3	5	26	12	7	6	0.27	0.5
9	9	3	5	7	51	33	11	7	0.23	0.21
10	8	8	8	19	56	19	15	15	0.27	0.79
11	8	8	8	19	58	19	14	15	0.24	0.79
12	8	8	8	19	58	19	14	15	0.24	0.79
13	8	8	8	19	58	19	14	15	0.24	0.79
14	1	1	5	4	13	12	4	4	0.31	0.33
15	1	1	5	4	14	12	4	5	0.29	0.42
16	2	1	5	5	23	14	7	6	0.30	0.43
17	2	2	5	6	24	22	6	4	0.25	0.18
18	4	3	6	8	28	31	10	19	0.36	0.61
19	10	7	18	22	42	38	8	9	0.19	0.23
20	7	7	10	11	64	42	18	16	0.28	0.38
21	3	1	3	5	25	19	9	7	0.36	0.37
22	3	2	9	10	43	34	10	9	0.23	0.26
23	3	4	8	8	29	16	9	8	0.31	0.5
24	8	8	8	18	67	19	15	15	0.22	0.77

TAB-4.8 Mesure du degré algorithmique

où:

NSIFOR: Nombre de structures de "IF" et "FOR" dans l'algorithme conforme;

NDZDS: Nombre de déclarations spécifiques de zone de données dans le programme/COB;

NTIE: Nombre total d'instructions exécutables d'un programme;

NINSD: Nombre d'instructions de DML;

PROP: = NINSD / NTIE

Deuxième partie: L'ASPECT TECHNIQUE

4.4 MESURE DES PERFORMANCES

4.4.1 L'observation des performances

a) Méthode

Dans notre cas," la méthode utilisée pour mesurer les performances est essentiellement basée sur les deux procédures offertes par la librairie (RUN-TIME-LIBRARY) de VAX/VMS:

LIB$INIT_TIMER : qui initialise le comptage;
LIB$SHOW_TIMER : qui stoppe le comptage et affiche les résultats.

Ces deux procédures (voir les programmes à l'annexe-4), appelées de manière à encadrer la suite d'instructions (ou plus précisément la transaction) pour laquelle nous voulons mesurer les événements, permettent de disposer des statistiques suivantes:

1) Temps réel écoulé (ELAPSED TIME): correspond au temps total défini au point 2.4.3.
Format: hhhh:mm:ss.cc;

2) Temps CPU (CPU TIME): correspond à overhead CPU entraîné défini au point 2.4.3;
Format: idem;

3) Comptage des opérations d'E/S directes: correspond au nombre d'accès au disque;
Format: nnnn;

4) Comptage des défauts de pages disque;
Format: nnnn.

Remarquons que, dans notre expérience, la plupart des applications à mesurer sont en mode de traitement interactif et qu'elles fonctionnent en temps partagé.

La précision des mesures dépend du critère suivant:

La performance d'un programme interactif s'exprime le plus souvent en fonction de la moyenne du temps de son exécution. Il est clair que quand le processeur est partagé (en effet, le VAX/8600 ayant un processeur unique est fort partagé), la mesure du temps d'exécution du programme est insuffisante. Le mécanisme de tranche de temps et le temps utilisé pour l'OS lui-même expliquent les différences observées. Nous nous attendons cependant à des temps relativement stables.

D'une manière générale, nous décidons que:

- pour chaque programme, nous réaliserons les exécutions des programmes dans une situation mono-utilisateur. C'est-à-dire que les mesures se font le soir ou pendant les weekends dans notre environnement universitaire;

- pour pratiquement toutes les mesures d'un même programme, nous réaliserons d'une façon générale 5 exécutions d'un même programme. Nous allons alors calculer la moyenne du temps CPU et du temps total écoulé lors de ces 5 exécutions.

- en ce qui concerne les programmes (vus comme des transactions) d'applications de type CONSULTATION ou MISE A JOUR, ELEMENTAIRE ou MOYENNEMENT COMPLEX, TRANSACTIONNEL et PAR CLE, nous les programmons de la façon suivante:

```
for B:= BIBLIO do
      INIT_TIMER;
      for i := 1 to n do
          begin
              START-TRANSACTION;
                      :
                      :
              COMMIT;
          end;
       endfor;
      SHOW_TIMER;
   endfor.
```

Où i, n : integer;

n: nombre de fois prédéfini pour répéter la transaction.

Ceci est fait dans le but d'obtenir les temps significatifs de l'ordre des secondes.

- pour les autres types d'applications, les programmes fonctionnant avec les SGBD différents ne s'exécutent qu'une seule fois de la façon suivante:

```
for B := BIBLIO do
    begin
        INIT_TIMER;
        START_TRANSACTION;
                :
                :
        COMMIT;
        SHOW_TIMER;
    end;
endfor.
```

b) Résultats

Les resultats de cette session de mesures se trouvent dans le TAB-4.9 et le TAB-4.10 pour les BD de taille réduite et le TAB-4.11 et le TAB-4.12 pour les BD de grande taille.

NOM DE PROGRAM	NERE	TEMPS TOTAL	TEMPS CPU	NOMBRE E/S	DEFAUTS DE PAGES
APPLI 1	10	27.75	22.96	20	355.2
APPLI 2	100	2.39	1.21	17	505.2
APPLI 3	1	4.07	2.40	20	407.6
APPLI 4	1	5.01	3.46	21	425
APPLI 5	1	5.24	3.46	21	441.8
APPLI 6	50	3;88	2.40	17	565.4
APPLI 7	50	4.13	2.43	17	539
APPLI 8	10	9.03	7.17	29	411.2
APPLI 9	1	2.52	1.43	13	312.8
APPLI 10	50	3.90	2.71	13	372.2
APPLI 11	50	8.33	6.55	20	561
APPLI 12	50	2.80	8.59	13	380
APPLI 13	50	3.15	2.03	12	363
APPLI 14	50	17.64	1.65	341	447
APPLI 15	50	17.90	1.62	328	452
APPLI 16	50	17.45	2.11	320	461
APPLI 17	50	30.57	4.66	532	606.2
APPLI 18	50	4.47	0.75	60	534.2
APPLI 19	50	54.87	15.10	944	653.8
APPLI 20	10	5.46	0.91	48	505
APPLI 21	10	6.99	1.05	111	439
APPLI 22	5	51.64	20.53	717	792.4
APPLI 23	15	5.77	1.64	78	457.4
APPLI 24	1	54.36	32.63	379	1079
APPLI 25	1	3:56.52	2:16.20	2179	943

NERE: Nombre de répétitions de la transaction pour une exécution du programme

TAB-4.9 Performance des programmes/DBMS pour la BD de taille réduite

NOM DE PROGRAM	NERE	TEMPS TOTAL	TEMPS CPU	NOMBRE E/S	DEFAUTS DE PAGES
APPLI 1	10	8.79	6.25	66	704.2
APPLI 2	100	9.26	2.50	249	820.6
APPLI 3	1	3.99	1.70	50	723.6
APPLI 4	1	7.81	2.28	57	1177
APPLI 5	1	7.86	2.41	57	1095.6
APPLI 6	50	19.94	11.85	156	852.4
APPLI 7	50	29.54	11.85	160	841.8
APPLI 8	10	10.85	4.53	78	894.6
APPLI 9	1	7.12	1.84	70	1026
APPLI 10	50	13.23	4.82	173	926.2
APPLI 11	50	2:96.72	32.70	1581	970
APPLI 12	50	12.23	4.02	178	1037.6
APPLI 13	50	1:60.17	14.11	1114	989.2
APPLI 14	50	19.07	2.65	293	1210.4
APPLI 15	50	17.69	2.54	297	1032.2
APPLI 16	50	1:08.65	11.13	1314	1450
APPLI 17	50	24.87	4.15	450	1363
APPLI 18	50	1:10.40	34.88	408	1625
APPLI 19	50	47.90	9.81	900	1460.8
APPLI 20	10	7:02.50	2:50.38	2283	2085.6
APPLI 21	10	12.07	2.17	188	1289
APPLI 22	5	43.52	7.66	443	1541.4
APPLI 23	15	23.20	7.62	762	1475.6
APPLI 24	1	6:58.30	40.77	1450	1643
APPLI 25	1	5:43.31	1:26.46	5954	2331

NERE: Nombre de répétitions de la transaction pour une exécution du programme

TAB-4.10 Performance des programme/RDB pour la BD de taille réduite

NOM DE PROGRAM	NERE	TEMPS TOTAL	TEMPS CPU	NOMBRE E/S	DEFAUTS DE PAGES
APPLI 1	1	2:14.03	31.70	1617	1231
APPLI 2	100	28.38	3.72	376	1833
APPLI 3	1	2:17.15	32.17	1619	2000
APPLI 4	1	2:26.58	44.45	1612	1342
APPLI 5	1	2:26.39	44.34	1620	157
APPLI 6	50	2.90	1.67	14	538
APPLI 7	50	3.52	1.93	19	842
APPLI 8	10	1:08.76	58.02	188	752
APPLI 9	1	8.60	5.69	39	842
APPLI 10	50	4.50	3.45	13	534
APPLI 11	50	9.11	7.37	21	866
APPLI 12	50	3.21	2.00	14	671
APPLI 13	50	5.19	3.30	21	628
APPLI 14	20	8.64	0.93	149	668
APPLI 15	20	11.02	1.33	212	798
APPLI 16	10	21.37	2.19	344	1322
APPLI 17	20	43.06	4.73	698	1762
APPLI 18	3	4.58	0.76	62	796
APPLI 19	5	23.33	3.54	393	984
APPLI 20	1	6.69	1.31	100	876
APPLI 21	10	1.83	0.40	14	453
APPLI 22	2	14:05.64	1:57.25	13349	1497
APPLI 23	15	1:18.64	7.02	1390	1276
APPLI 24	1	33:02.37	6:28.06	24212	1592
APPLI 25	1	3:54:06.77	35:34.77	221254	1952

NERE: Nombre de répétitions de la transaction pour une exécution du programme

TAB-4.11 Performance des programmes/DBMS pour la BD de grande taille

NOM DE PROGRAM	NERE	TEMPS TOTAL	TEMPS CPU	NOMBRE E/S	DEFAUTS DE PAGES
APPLI 1	1	58.38	55.31	84	864
APPLI 2	100	15.37	3.50	322	949
APPLI 3	1	7.33	3.80	84	1002
APPLI 4	1	8:55.65	45.55	3859	420
APPLI 5	1	22.61	15.13	130	4078
APPLI 6	50	6.55	2.49	167	688
APPLI 7	50	2:13.40	53.04	1230	702
APPLI 8	10	31.66	11.02	285	957
APPLI 9	1	25.32	12.44	180	2042
APPLI 10	50	1:46.11	16.18	1227	760
APPLI 11	50	5:14.16	33.25	3179	789
APPLI 12	50	2:00.72	14.29	1278	1032
APPLI 13	50	02:44.21	16.33	1397	1135
APPLI 14	20	24.30	03.48	539	889
APPLI 15	20	15.34	2.89	446	941
APPLI 16	10	23.53	4.06	522	1043
APPLI 17	20	47.59	6.12	677	1264
APPLI 18	3	16:15.16	2:31.92	14049	1478
APPLI 19	5	44.53	7.10	1101	1557
APPLI 20	1	19:27.86	5:00.77	17170	2129
APPLI 21	10	71.02	1.28	65	1112
APPLI 22	2	35:58.98	51.11	9240	1242
APPLI 23	15	12:02.50	21.62	2556	1802
APPLI 24	1	2:17:00.73	14:29.24	90057	1626
APPLI 25	1	3:22:36.86	23:14.85	157565	2341

NERE: Nombre de répétitions de la trasaction
pour une exécutin du programme

TAB-4.12 Performance des programmes/RDB
pour la BD de grande taille

NOM DE PROGRAM	NERE	TEMPS TOTAL	TEMPS CPU	NOMBRE E/S	DEFAUTS DE PAGES
APPLI 4	1	20.62	15.16	17	177.6
APPLI 5	1	8.46	5.62	17	238.8
APPLI 6	50	23.05	13.96	160	814.2
APPLI 7	50	39.26	22.72	164	876.2
APPLI 8	10	13.05	5.40	82	745.2

NERE: Nombre de répétitions de la transaction
pour une exécution du programme

FIGURE-4.10bis Performance des programmes/RDB
avec optimisation explicite

NOM DE PROGRAM	NERE	TEMPS TOTAL	TEMPS CPU	NOMBRE E/S	DEFAUTS DE PAGES
APPLI 4	1	1:05:52.55	3:16.45	3967	191
APPLI 5	1	56.85	50.31	85	262
APPLI 6	50	7.25	2.91	167	718
APPLI 7	50	2:14.88	55.70	1230	1040
APPLI 8	10	28.70	11.48	287	713

NERE: Nombre de répétitions de la transaction
pour une exécution du programme

TAB-4.12bis Performance des programmes/RDB
avec optimisation explicite

4.4.2 Le calcul de l'efficacité des programmes

Les paramètres NRS, NRL, et NRP (voir 2.4.3) sont utilisés pour mesurer l'efficacité du programme d'application. Malheureusement, nous ne pouvons pas obtenir directement les valeurs de ces paramètres avec l'observation simple. Il nous faut tenir compte de la stratégie d'accès aux données du programme, utiliser certains paramètres intermédiaires et faire quelques calculs.

a) LRE (efficacité du programme en termes de records logiques):

D'après la nature des applications et de la définition de LRE, nous ne choisissons qu'une partie de nos applications pour mesurer ce point. Les raisons en sont les suivantes:

D'une part par exemple, si dans l'application1 on tend à lister tous les numéros d'OUVRAGE, il est évident que NRS et NRL pour les deux programmes fonctionnant avec les SGBDs differents sont identiques et que leur LRE est égal à une unité.

D'autre part, nous remarquons dans l'annexe-1 que NUMOU d'OUVRAGE en DBMS est une clé calculée, tandis que NUMOU d'OUVRAGE en RDB est une clé indéxée. Il est clair que selon cette définition d'efficacité, un programme qui manipule les données dans la BD en utilisant la clé calculée est plus efficace. Par exemple, nous avons une application "Etant donné un numéro d'OUVRAGE, fournir son titre". Avec la clé calculée, le programme DBMS/COBOL de cette application trouvera immédiatement l'endroit où est le record ayant le numéro donné, tandis que le programme RDB/COBOL doit tout d'abord faire référence aux index et trouvera ensuite les données désirées.

L'analyse ci-dessus exige que nous choissions dans nos applications certains types. Nous décidons de prendre les applications 3, 6, 8, 9, 15, 17, 21. Les résultats se trouvent dans le TAB-4.13.

nom de program.	DBMS	RDB
appli3	0.025	0.025
appli6	0.36	0.002
appli8	0.008	0.008
appli9	0.037	0.002
appli15	0.25	0.25
appli17	0.5	0.5
appli21	0.14	0.25

TAB-4.13 Efficacité du programme LRE

b) PRE (efficacité du programme en termes de records physiques):

La mesure sur ce point ne peut se faire qu'après que l'observation du point 4.4.1 ait été réalisée. En effet, pour calculer le NAP (nombre total de records physiques accédés), il faut disposer du nombre d'opérations E/S. Grâce à cet exemple nous avons remarqué qu'il faudrait aussi considérer l'ordre dans le temps des sessions de mesures, lorsqu'un ensemble de mesures est commencé.

Les résultats se trouvent dans le tableau-4.14.

nom de program.	DBMS	RDB	nom de program.	DBMS	RDB
appli1	0.002	0.007	appli13	0.015	1.40
appli2	0.0002	2.19	appli14	6.82	5.86
appli3	0.8	2	appli15	6.56	5.94
appli4	0.023	0.063	appli16	3.25	13.14
appli5	0.022	0.06	appli17	1.06	0.9
appli6	0.085	0.78	appli18	0.12	0.82
appli7	0.085	0.8	appli19	2.70	2.57
appli8	0.5	1.95	appli20	0.96	5.07
appli9	13	70	appli21	5.55	9.4
appli10	0.016	0.23	appli22	1.43	0.89
appli11	0.005	0.44	appli23	0.78	7.62
appli12	0.016	0.22	appli24	-	-

TAB-4.14 Efficacité des programmes PRE

4.5 MESURE DU VOLUME ET DU TAUX DE REMPLISSAGE

Après la création de la BD nous poursuivrons cette session de mesures. Les valeurs de volume et du taux de remplissage de la BD seront collectées à l'aide d'utilitaires de chaque SGBD:

Dans le VAX DBMS on dispose de l'instruction:

```
$> DBO/ANALYZE .
```

Elle nous donne une vue physique de notre base de données. Les sorties de cette instruction montrent comment la base de données utilise l'espace disponible. Elle montre aussi le taux de remplissage des pages de stockage. Les autres paramètres sont utiles pour le DBA.

Dans le RDB/VMS on a l'instruction:

rdo> ANALYZE OPTIONS (PAGES, RELATIONS OU INDEXS) .

Cette instruction nous donne une image de la BD qui a la forme et la signification des sorties à peu près équivalentes à celles des VAX DBMS.

Les résultats sont chiffrés au TAB-4.15 et TAB-4.16 pour les BD de taille réduite et TAB-4.17 et TAB-4.18 pour celles de grande taille.

type d'article	occurrences d'article	DBMS (bytes)	RDB (bytes)
AUTEUR	501	33234	14529
OUVRAGE	1001	105484	33033
OUVAUT	1950	55848	25350
MCOUV	1950	109377	3900
EXEMPLAIRE	2100	76517	39900
EMPRUNTEUR	100	9054	2900
EMPRUNT	81	2590	1377
ENPRUNT-ARCH	280	10314	5880

TAB-4.15 Volume des fichiers des BD de taille réduite

	DBMS		RDB	
AREA	volume(pages)	taux de remp.	volume(pages)	taux de remp.
FAUTEUR	200	56%	2052	68%
FOUV	300	79%		
FEXEM	200	53%		
FEMPR	200	13%		
TOTAL	900	50.2%		

TAB-4.16 Volume et taux de remplissage des BD de taille réduite

type d'article	occurrences d'article	DBMS (bytes)	RDB (bytes)
AUTEUR	3000	251235	93029
OUVRAGE	10000	761208	350034
OUVAUT	14650	441715	249029
MCOUV	14650	856483	322300
EXEMPLAIRE	25860	1004497	594773
EMPRUNTEUR	300	26479	9298
EMPRUNT	200	7063	4221
ENPRUNT-ARCH	2000	78516	49922

TAB-4.15 Volume des fichiers des BD de grande taille

	DBMS		RDB	
AREA	volume(pages)	taux de remp.	volume(pages)	taux de remp.
FAUTEUR	900	88 %	8028	76 %
FOUV	2100	91%		
FEXEM	1600	89%		
FEMPR	400	32%		
TOTAL	5000	77.5%		

TAB-4.16 Volume et taux de remplissage des BD de grande taille

CHAPITRE 5
ANALYSE ET COMPARAISON DES RESULTATS

L'objectif de ce chapitre est d'exploiter les résultats obtenus au chapitre précédent et d'aider le lecteur dans l'utilisation des résultats et, si nécessaire, dans la définition de nouveaux résultats permettant de prendre une décision.

première partie: L'ASPECT LOGIQUE

5.1 PUISSANCE DU MODELE DE DONNEES

1) A partir des représentations graphiques et logiques étudiées dans la section 4.1, il est évident que le modèle DBMS semble plus complexe. En effet, pour modéliser les données il faut disposer des concepts de type record, de type set (liaison entre deux types de record) et d'item, tandis qu'en RDB, la forme de representation des données est très simple. Il suffit de disposer des concepts de relation (tableau) et d'attribut.

La complexité de la structure de données en SGBD CODASYL implique une complexité du DML. Pour un programmeur d'application de BD, il faut tenir compte des concepts de clé de la BD, d'occurrence d'un type de record, d'occurrence d'un set, d'OWNER records et MEMBER records, de courant d'un type de record, de courant d'un type de set etc (cfr. VAX-11 COBOL USER GUIDER). Au contraire, en RDB, aucun de ces concepts n'est vu par les utilisateurs.

2) A propos des représentations logiques, nous remarquons qu'en DBMS les utilisateurs (du moins ceux qui veulent stocker des données dans la BD) sont obligés d'avoir connaissance de la localisation de chaque type de record. Ceci est dû à la clause WITHIN qui apparaît dans le texte du schéma et l'instruction STORE qui a la forme suivante:

STORE nom-record WITHIN nom-area.

Il n'y a aucune description concernant la localisation de données dans la représentation logique de RDB.

3) Du point de vue de la représentation physique, en RDB, la définition de la clé d'accès est, en effet, une option. Le DBA peut définir selon les besoins, par exemple, la performance, les clé d'accès pour certaines relations dans le texte du schéma de la BD. Mais utilisateurs et programmeurs peuvent complètement ignorer ces clés d'accès prédéfinies quand ils manipulent les données de la BD. Le RDB dispose d'un "optimiser" qui offre la stratégie d'accès optimale selon les prédéfinitions des clés d'accès.

Prenons un exemple contraire dans le DBMS. Supposons qu'on ait une application transactionnelle "Etant donné un numéro d'auteur, fournir son origine." et que DBA ait défini une clé indexée sur le numéro d'auteur NUMAU. Un programmeur commence à coder cette application en COBOL. Mais il ignore complètement le NUMAU qui est une clé d'accès et programme de la façon suivante:

```
INPUT ( XNUMAU ).
MOVE 0 TO TROUV-AUT.
MOVE 0 TO FIN-SET.
FIND FIRST AUTEUR WITHIN SYSTAUTEUR.
      IF NUMAU OF AUTEUR = XNUMAU
         MOVE 1 TO TROUV-AUT
      ELSE
          PERFORM FIND-NEXT-AUT UNTIL
             FIN-SET = 1 OR TROUV-AUT = 1.
 séquence d'instructions.

FIND-NEXT-AUT.
   FIND NEXT AUTEUR WITHIN SYSTAUTEUR
         AT END MOVE 1 TO FIN-SET.
   IF NUMAU OF AUTEUR = XNUMAU
      MOVE 1 TO TROUV-AUT.
```

Nous pouvons imaginer qu'il y aura une dégradation très forte de la performance si AUTEUR contient des millions de records. Le SGBD doit parcourir en moyenne un demi-million de records pour trouver le record désiré.

La façon correcte de programmer est la suivante:

```
INPUT (XNUMAU).
MOVE XNUMAU TO NUMAU OF AUTEUR.
FIND FIRST AUTEUR WITHIN SYSTAUTEUR USING NUMAU.
séquence d'instructions.
```

4) Nous remarquons que les deux modèles sont capables de représenter les concepts fondamentaux de l'E/A. DBMS possède la capacité de représenter effectivement les associations complexes (par exemple, associations de type "many-to-many") sans introduire de redondances de données (voir figure-4.1 et figure-4.3). Prenons par exemple un type d'association entre deux types d'entité avec la cardinalité N-N. Certains auteurs considèrent que l'introduction d'un nouveau type de record est peu naturelle [DELO 82]. Certains pensent qu'une association "many-to-many" doit contenir certaines informations à propos de' l'association elle-même [ROBE 71]. Pour l'association N-N, supposons un programme d'application qui va modifier un identifiant de X (voir la figure-4.1). En DBMS, on trouve le record x et on le modifie tout simplement. En RDB, il faut non seulement modifier la valeur de l'identifiant de la relation X mais aussi la(es) valeur(s) d'identifiant de la relation XY. Il s'agit du même problème que pour la suppression d'un identifiant de x.

5) En ce qui concerne la représentation d'un identifiant d'un type d'article (voir TAB-4.6), deux choses sont à noter:

La première est qu'en DBMS, il n'y a pas moyen de définir un identifiant qui soit une clé calculée. Nous avons montré au point 4.4.2, que l'utilisation d'une clé calculée est plus efficace que l'utilisation de la clé indexée. Ceci signfie qu'il faut faire un compromis dans son choix, entre, soit la performance des programmes, soit l'intégrité de la BD.

La seconde concerne la puissance du modèle. Lorsqu'un identifiant ne peut être défini que comme une clé indexée dans les deux SGBD, nous pouvons comparer la puissance du modèle en posant la question suivante:

Supposons qu'on veuille imposer une contrainte d'intégrité sur chaque type d'article de la BD, le modèle peut-t-il réaliser cette contrainte dans toutes les circonstances ?

La réponse est positive pour RDB et négative pour DBMS. Nous l'expliquons dans ce qui suit.

Regardons le schéma-3. Nous remarquons que le type de record OUVAUT du schéma est à l'origine d'un type d'association "many-to-many" entre type d'article OUVRAGE et AUTEUR. En effet, DBMS ne peut pas réaliser l'identification de ce type de record. Normalement il doit être identifié par les sets OO et AO. Lorsqu'il n'a pas d'item, l'utilisateur ne peut pas établir l'unicité des records OUVAUT.

Dans les limites de nos moyens, nous pouvons réaliser la contrainte en introduisant des données redondantes. Par exemple, définissons deux items NUMOU et NUMAU pour OUVAUT. L'unicité des sets OO et AO peuvt alors être établi sur ces items. Puis nous définissons une clause DUPLICATES ARE NOT ALLOWED sur ces sets. OUVAUT peut par conséquent, être identifié.

Notons qu'un auteur a écrit 8 ouvrages au plus. Si l'on définit les index sur les sets OO et AO, il n'est évidemment pas économique du point de vue de la place du disque et de la performance.

En revanche, dans le cas d'RDB, grâce a l'uniformité des définitions des données, il n'y a pas de problème pour garantir l'unicité d'un type de record (relation). A la limite, l'ensemble des items de la relation peut jouer le rôle d'identifiant.

5.2 INDEPENDANCE DE PROGRAMMES PAR RAPPORT AU CHANGEMENT DE LA STRUCTURE DES DONNEES

Nous allons maintenant comparer l'indépendance de données offerte par les SGBD différents en faisant référence au TAB-4.6.

5.2.1 En DBMS

Un utilisateur peut définir sa vue comme un sous-schéma. Ce sous-schéma doit être limité à un sous-ensemble du schéma de la BD. C'est-à-dire que ce que l'on définit dans le sous-schéma (les types de record, les types de set et les items) doit se trouver dans le texte du schéma de la BD. Bien que cette restriction soit gênante pour certaines applications, par exemple renommer (changer) le nom d'un type de record, d'un item etc., il n'a aucun impact sur les programmes d'application dans le cas où l'on ajoute un type de record, un item, un set (voir TAB-4.7) dans le schéma. De toute façon, la BD peut s'agrandir au cours du temps sans affecter les programmes d'application.

Notons que pour atteindre un degré d'indépendance de haut niveau, le SGBD doit offrir des composants virtuels d'objets afin de faire un "mapping" entre le schema et sous-schéma [WAGH 75]. DBMS actuel n'offre pas cette facilité.

Nous avons vu dans les points 4.1 et 5.1 qu'une partie concernant l'indépendance physique a été déjà évoquée et nous avons remarqué que le DBMS est plutôt faible sur ce plan. Ceci peut s'expliquer par les faits suivants:

Le programmeur doit rendre compte du mécanisme de cle d'accès (CALC, CHAIN, ou INDEX) de chaque type de record, de l'AREA où les records sont stockés, de la méthode de représentation d'association entre types de record et des caractéristiques de set. En plus, lorsque les définitions de clé d'accès doivent être connues par l'utilisateur, si celui-ci les utilise dans ses programmes, la modification des définitions de clés d'accès dans le schéma entraîne inévitablement la modification des programmes.

En effet, du côté du SGBD lui-même, la possibilité de changement du mode d'une clé d'accès est très limitée. Par exemple, pour définir une clé indexée d'un type de record en DBMS , il faut d'abord définir le type de set où le type de record est MEMBER, comme un set trié (SORTED) dans le schéma. Donc, si l'on veut changer le mode de la clé, par exemple, d'une clé indexée en une clé calculée, il est impossible de le faire sans recompiler le schéma. Par conséquent, une fois que le schéma est recompilé, les programmes d'application existants, eux aussi, doivent l'être obligatoirement.

Le cas contraire est permis, c'est-à-dire la transformation d'un item en une clé calculée. C'est une flexibilité offerte par DBMS. Mais, signalons que, dans ce cas-ci, cela a un impact sur le programme d'application (voir exemple de 5.1).

5.2.2 En RDB

Une capacité remarquable du SGBD RELATIONNEL RDB est de définir une vue (VIEW) qui est elle-même une relation. Cette vue n'est pas obligatoirement définie dans le texte du schéma de la BD. Elle peut être beaucoup plus générale. Par exemple, elle peut être une jointure de deux relations du schéma de la BD. En général un sous-schéma est constitué de ces types de record "virtuels" qui peuvent être définis par de vrais types de record dans le schéma de la BD. Prenons une application de notre système d'information BIBLIOTHEQUE "Fournir une liste des informations des OUVRAGES qui ont au moins un exemplaire" Pour celle-ci, nous pouvons définir une vue comme suit:

```
DEFINE VIEW EXEMPLAIRE-COMPUTE OF
        EX IN EXEMPLAIRE CROSS O IN OUVRAGE
        OVER NUMOU
        O.TITRE.
        O.ANNEE.
        EX.NUMOU.
        EX.NUMEX.
END VIEW.
```

Où: L'EXEMPLAIRE-COMPUTE s'appelle le type de record "virtuel" et l'EXEMPLAIRE et l'OUVRAGE les vrais types de record.

Grâce à cette capacité, nous pouvons dynamiquement définir les associations entre les relations du schéma et facilement ajouter un domaine dans une relation ou une relation dans le schéma sans affecter les programmes d'application.

Notons qu'il est difficile de supporter les opérations de mise à jour d'une vue sur les records stockés dans la BD [DELO 82]. RDB ne le supporte pas non plus.

En ce qui concerne l'indépendance physique, RDB offre une grande flexibilité pour implanter les relations et les domaines et changer le mode d'accès. Lorsque le programmeur d'application de BD ne voit pas les chemins d'accès et ne doit pas avoir une conscience de la localisation des données, tous les changements de la structure de stockage des données sont supportés.

Il est à noter qu'RDB ne gère pas la notion de version de la BD. C'est-à-dire que, bien que l'on modifie les objets du schéma conceptuel et que l'on doive ensuite recompiler le schema de la BD, les programmes qui utilisent des objets qui n'ont pas été modifiés peuvent fonctionner tout de suite sans être recompilés après la création de la novelle BD.

5.3 EVALUATION DU DEGRE ALGORITHMIQUE

Le tableau TAB-4.8 donne les résultats des mesures du degre algorithmique des programmes. Il ne s'agit que d'une expérience de mesure quantitative. Il faut les interpréter en relevant certains facteurs qui affectent la procéduralité des programmes d'application.

Nous reprenons l'exemple développé dans la section 4.3. Nous l'analysons aussi à deux niveau.

5.3.1 Au niveau de l'algorithme conforme:

Au vu de l'ALG-4.1 et l'ALG-4.2, nous observons que:

- lors du premier pas de transformation de l'algorithme prédicatif en algorithmes conformes, la structure de l'algorithme conforme à DBMS devient déjà plus complexe: une structure "FOR" dans l'algorithme prédicatif est divisé en quatre structures "FOR" et "IF", tandis que l'algorithme conforme à RDB n'en contient qu'une seule.

Ceci vient des faits suivants:

- l'origine d'AUTEUR n'est pas une clé d'accès. La condition pour obtenir l'objet désiré n'est ni évaluable ni développable. DBMS n'offre pas de facilité d'accès par filtre. Il faut donc avoir recours au connecteur booléen "IF".

- dans le schéma-3 nous voyons que pour obtenir l'objet désiré (ouvrage et auteur) il faut passer le set AO, le type de record OUVAUT et l'autre set OO. Nous regardons le schéma-4 dans lequel aucun chemin n'apparaît. Ce sont les domaines en commun qui font la liaison entre les types d'entité. Mathématiquement, le RDB peut réaliser le "for" dans les algorithmes prédicatifs sans le développer ni le décomposer.

Dans la section 5.1, nous avons implicitement fait remarquer que le domaine en commun pour le modèle relationnel introduit des donnees redondantes. Du point de vue la proceduralite, est-ce que ce concept de domaine est "naturel" ? Que pense l'utilisateur lui-même?

5.3.2 Au niveau du programme exécutable

Au vu des programmes ALG-4.1 et ALG-4.2, nous remarquons que :

- dans le PRG-4.1, un "for" dans l'algorithme conforme ALG-4.1 est divisé en deux instructions de DML FETCH FIRST... et FETCH NEXT.... Ceci revient à considérer la logique adaptée par les SGBD pour accéder aux données de la BD:

Le SGBD CODASYL ne permet que l'accès au niveau logique "one-record-at-a-time". Donc, pour parcourir un fichier (un type de record) de la BD, il faut d'abord positionner le premier record du fichier (FETCH FIRST), puis faire une boucle (PERFORM ... UNTIL), afin d'accéder aux autres records (FETCH NEXT) jusqu'à la fin du fichier.

Nous voyons que le résultat de ce genre de logique est véritablement d'augmenter le code source du programme et que le programmeur doit programmer d'une manière très explicite pour manipuler les données de la BD.

Au vu du PRG-4.2, lorsque le SGBD RELATIONNEL permet la logique dite "multi-records-at-a-time", le programme est donc très concis.

- dans la colonne NDZDS du tab-4, nous pouvons nous apercevoir que le nombre de déclarations spécifiques de zone de données dans les programmes/RDB/COB est plus élevé que dans les programmes/DBMS/COB. On peut étudier ce problème sous l'aspect de la compatibilité entre DML et COBOL. Nous voudrions dire en plus que si un DML n'est pas compatible avec le "host" langage, il risque d'augmenter le niveau procédural des programmes d'application. Nous illustrerons ceci à l'aide des quelques exemples suivants:

a) Dans le PRG-4.2, lorsque les objets du DML/RDB ne sont pas connus et ne peuvent pas être référencés dans l'environnement COBOL, il faut explicitement déclarer les variables intermédiaires pour que l'utilisateur puisse obtenir les objets de la BD (voir WORKING-STORAGE SECTION et l'instruction GET dans le PRG-4.2).

b) Un autre exemple montre que RDB n'a pas exploité une bonne structure de contrôle dans l'environnement COBOL, si bien que pour certaines applications/RDB le niveau "multi-records-at-a-time" doit descendre au niveau "one-record-at-a-time". Nous verrons pourquoi en faisant référence au TAB-4.8 à ce même moment:

Nous nous sommes aperçu dans le TAB_4.8 qu'en RDB, le nombre total d'instructions exécutables (NTIE) dans l'application 18 est plus grand que pour la même application en DBMS.

En réalité, pour retirer un exemplaire (supposons qu'il existe), étant donné un numéro d'ouvrage, il faut d'abord vérifier que l'exemplaire à retirer est déjà prêté ou non. S'il est prête, alors on ne peut pas le retirer, sinon on le retire et on arrête le programme. A la sortie, on voudrait aussi savoir s'il y a un exemplaire qui a été vraiment supprimé.

Nous commençons à programmer en COBOL de la façon suivante:

```
&RDB&     START-TRANSACTION.
    INPUT (XNUMOU).
    MOVE 0 TO UNE-SUPP-EFFECT.
    PERFORM  BIBLIO THRU BIBLIO-EXIT.
    IF UNE-EUPP-EFFECT = 0
       DISPLAY "pas d'exemplaire  supprimé".
&RDB& COMMIT
    STOP RUN.

BIBLIO.
 &RDB& FOR EX IN EXEMPLAIRE WITH
 &RDB&       EX.NUMOU = XNUMOU
 &RDB&       GET NUMEX1 = EX.NUMEX END_GET
    MOVE 1 TO DELET-OK
    PERFORM VERIF-DEL-OK  (* procédure qui a pour but de donner
                            une valeur 0 à delet-ok si l'exemplaire
                            n'est pas prêté*)
```

```
    IF DELET-OK = 1
      PERFORM DELT-EX      (* procédure qui supprime l'exemplaire *)
      MOVE 1 TO UNE-SUPP-EFFECT
      GO TO BIBLIO-EXIT
&RDB& END-FOR.

BIBLIO-EXIT
   EXIT.
```

PRG-5.3 Exemple de programme de l'application 18

En effet, les instructions IF et GO TO ne peuvent pas être emboitées dans la boucle "FOR" du DML/RDB. Si on le fait comme dans le prg-5.3, la compilation va échouer. On est donc obligé de programmer en utilisant l'instruction START-STREAM du DML qui accède à la BD de façon "one-record-at-a-time" (cfr. appli18 en annex-4).

Il est à noter que l'instruction FOR de DML/RDB ne traite pas certains cas exceptionnels, comme par exemple la fin d'une agrégation. Cela peut être très gênant dans le cas où l'on veut vérifier, par exemple, qu'un auteur existe ou non, étant donné son identifiant. Le programme avec l'instruction "FOR" parcourt la relation AUTEUR. S'il y en a un, le programme franchit la boucle et passe à l'instruction suivante, sinon il continue à chercher jusqu'à la fin du fichier AUTEUR. Après l'instruction FOR, on ne sait pas si l'objet a été trouvé ou non. Pour le savoir, il faut au moins ajouter deux instructions COBOL (voir la ligne 2 et la ligne 5 du programme suivant):

```
1          INPUT (XNUMAU).
2          MOVE 0 TO AUT-TROUVE.
3 &RDB&   FOR A IN AUTEUR WITH
4 &RDB&          A.NUMAU = XNUMAU
5             MOVE 1 TO AUT-TROUVE
6 &RDB&   END-FOR
7          instructions suivantes
           :
           :
```

PRG-5.4 Exemple d'un programme

Nous terminons ce paragraphe en signalant que, lorsque toutes les applications que nous avons faites ont pour objet de manipuler les données de la BD, le DML a d'ailleur le même but, la proportion NINSD/NTIE dans le tab-4.8 montre d'une part la puissance et d'autre part la procéduralité d'un DML. Cela veut dire que pour ce genre d'applications, leurs programmes/COB doivent avoir le moins d'instructions COBOL possible, utilisées pour contrôler le processus de traitement.

Il est clair que le DML/RDB est beaucoup plus intéressant a ce point de vue que le DML/DBMS.

deuxième partie: L'ASPECT TECHNIQUE

5.4 COMPARAISON DE LA PERFORMANCE

Notre analyse se concentre sur les tableaux TAB-4.11 et TAB-4.12 pour les BD de grande taille. Quant aux TAB-4.9 et TAB-4.10 pour les BD de taille réduite, nous ne ferons pas beaucoup de commentaires. Le lecteur peut faire la comparaison des résultats avec ceux des TAB-4.11 et TAB-4.12.

Pour mieux comparer les résultats obtenus au point 4.4, nous donnons la liste suivante, laquelle donne en détail les applications qui correspondent aux types d'applications.

type d'application	numéros des applications
consultation	1,2,3,4,5,6,7,8,9,10,11,12,13,24
mise-à jour	14,15,16,17,18,19,20,21,22,23,25
transactionnel	2,6,7,8,9,14,15,16,17,18,19,20,21
de masse	1,3,4,5,10,11,12,13,22,23,24,25
élémentaire	1,2,3,14,15,16,
moy.complexe	4,5,6,7,8,17,18,19,22
complexe	9,10,11,12,13,20,21,23,24,25
accès séquent.	1,4,5,10,11,12,13,24,25
par clé	2,6,7,9,14,15,16,17,18,19,20,21,23
séq.filtré	3,8,22

LISTE-5.1 Les numéros des applications et leur typologie

5.4.1 Temps d'exécution

En général, le temps total écoulé des programmes d'application en DBMS est plus favorable qu'en RDB, bien qu'il y ait des variations à propos du temps CPU.

Le fait s'explique par la quatrième colonne des tab-4.11 et tab-4.12. Nous voyons qu'en DBMS, les programmes produisent presque toujours moins d'opérations d'E/S (l'explication en sera donnée plus loin) par comparaison avec les programmes/RDB qui effectuent le même type d'application. Dans ce sens-la, si l'on considere que le temps total écoulé (temps de réponse) est un des indices les plus importants de la performance pour des applications de nature interactive, on pourrait dire que le temps CPU dans les mesures de performance est moins significatif par rapport au temps des opérations d'E/S. C'est pourquoi on suppose souvent que les E/S sont une cause majeure des goulots d'étranglements dans les SGBD [HAWT 79].

En comparant le TAB-4.11 et TAB-4.12, nous remarquons qu'il y a une dégradation très forte pour les programmes/DBMS des applications 1, 2, 3 et 5. Nous avons cherché les causes de ceci et trouvé:

- qu'il n'est pas adéquat de définir (nous avous défini) une clé calculée au type d'article OUVRAGE pour des applications qui lisent en une exécution, tous les records d'OUVRAGE, parce qu'avec la clé calculée, les records sont distribués partout sur le disque. Notons qu'une operation E/S contient plusieurs pages disque consécutives. Grâce à cet exemple, nous pouvons dire, de manière générale, qu'il ne faut pas définir une clé calculée pour les objets auxquels les applications de masse accèdent;

- qu'il y a 75% de records d'OUVRAGE qui sont fragmentés lors du chargement des données dans la BD. Il est normal que le programme produise plus d'E/S pour trouver un record qui est divisé en deux ou trois endroits du disque. Nous avons essayé d'améliorer cette situation en changeant les paramètres de la création de la BD, mais sans beaucoup de succès. Ceci nous pose donc le problème de la validation des mesures. Nous en parlerons au chapitre 6.

5.4.2 Temps CPU

En ce qui concerne le temps CPU, les programmes d'application en DBMS qui ont la typologie: consultation ou mise à jour, transactionnel, par clé et que ce soit élémentaire, moyennement complexe, ou complexe, ont un très net avantage sur les mêmes types d'applications en RDB. Mais la situation s'inverse pour des applications de typologie consultation, de masse, élémentaire ou moyennement complexe et accès séquentiel ou séquentiel filtré.

Selon la définition d'application de type de masse (voir chapitre 2), il est évident que l'application de masse est bien accordée à la nature du DML relationnel qui est dite ensembliste. En revanche, si l'application est de type transactionnel et élémentaire, il faut voir quelle technique d'accès adopte le SGBD. Nous notons que le RDB n'offre que la technique d'accès index et qu'il n'existe pas d'accès calculé ou par chemin. L'article [DONA 81] montre que l'index sera efficace dans le cas où la transaction accède à beaucoup de records, tandis que "hashing" (accès calculé) et accès par chemin renforcent la performance de la transaction qui accède à peu de records. Ce qui explique pourquoi les applications en DBMS de type transactionnel et surtout complexe l'emportent sur les applications en RDB.

5.4.3 Accès physique

Nous portons maintenant notre attention sur la quatrième et la cinquième colonne des tab-4.9 et tab-4.10 (aussi des TAB-4.11, TAB-4.12). Nous remarquons tout de suite que les applications en DBMS, surtout pour ceux d'objets de taille complexe, produisent moins d'opérations d'E/S et de défauts de pages disque qu'en RDB. Ceci nous conduit à obtenir le résultat du TAB-4.14. Il est évident qu'étant donné une requête, le nombre de records qui satisfont cette requête est déterminé. Si deux programmes fonctionnent avec les SGBD différents, alors le programme qui produit moins d'opérations E/S est plus efficace. Ce phenomène s'explique par le fait suivant: DBMS offre aux utilisateurs la possiblité de définir l'organisation des données de la BD. Grâce à cette possibilité, une efficacité de haut niveau du programme est réalisable.

Par exemple en DBMS, on peut définir un set de telle façon que les records membres d'un set soit "clustered" via OWNER records du set, via un set index ou un chemin. Quand on essaye de trouver les records dans un "clustered" set, il y a beaucoup de chance de les trouver sans avoir plus d'un accès E/S, et sans défaut de page.

Par opposition, RDB, bien qu'il dispose d'un "optimiser" qui a pour but d'optimiser des accès à la BD selon des index prédéfinis, n'offre pas aux index, comme en SYSTEM R [DONA 81], la propriété de "clustering". On ne sait donc pas où sont les records de même type. Il semble alors qu'en RDB la possiblité de produire plus d'operations E/S et plus de défauts de pages soit plus élevée qu'en DBMS.

En vue de faire une comparaison nous prendrons les deux programmes de l'application 6 dont la description est la suivante:

Fournir le(s) ouvrage(s) qui est(sont) écrit(s) par un auteur donné.

Pour le programme/DBMS/COB de cette application, nous le simplifions comme suit:

```
1   READY.
2   MOVE 245 TO NUMAU OF AUTEUR.
3   FIND FIRST AUTEUR WITHIN SYSTAUTEUR USING NUMAU.
4   FIND FIRST OUVAUT WITHIN AO.
5   PERFORM FET-OU.
6   MOVE 0 TO FIN-SET.
7  PERFORM NEXT-OUVAUT THRU NEXT-EXIT UNTIL FIN-SET=1.
8   COMMIT.

NEXT-OUVAUT.
9    FETCH NEXT OUVAUT WITHIN AO AT END
10           MOVE 1 TO FIN-SET
11           GO TO NEXT-EXIT.
12   PERFORM FET-OU.

NEXT-EXIT.
13      EXIT.

FET-OU.
14     FETCH OWNER WITHIN OO.
15    DISPLAY  OUVRAGE.
```

PRG-5.4 Programme/DBMS/COB de l'application 6

En RDB:

```
1 &RDB&  START-TRANSACTION
2          PERFORM BOUCLE
3 &RDB&  COMMIT.

BOUCLE.
4 &RDB&  FOR O IN OUVRAGE CROSSE OU IN OUVAUT WITH
5 &RDB&        O.NUMAU = OU.NUMOU AND
6 &RDB&        OU.NUMAU = 245
7 &RDB&        GET O END-GET
8          DISPLAY O
9 &RDB&  END-FOR.
```

PRG-5.5 Programme/RDB/COB de l'application 6

a) Dans le prg-5.4:

(1) Regardons la ligne 3, avec 3 accès au disque (2 niveaux d'index). On peut obtenir les informations concernant l'auteur donné. On a aussi la possiblité de définir le NUMAU comme clé calculée. Dans ce cas-là, un seul accès est suffisant.

(2) Il y a un pointeur (chain) pour le set AO. Chaque record OUVAUT contient un pointeur qui pointe vers le record suivant du même set. Il est possible d'obtenir les records OUVAUT (voir la ligne 7) qui sont liés à l'auteur donné sans avoir les accès auxiliaires lorsque le set est défini de manière "clustered" ou organisé via la clé NUMAU.

b) Dans le prg-5.5:

(1) "Clustered" n'est pas supporté par RDB. Dans la ligne 4, le NUMAU de OUVAUT est une clé index dupliquée. Il est donc possible pour chaque ouvrage écrit par le même auteur donné, d'avoir 3 accès au disque.

(2) Remarquons la ligne 4. Il est très difficile pour un "optimiser" de faire intelligemment la jointure entre OUVRAGE et OUVAUT. S'il prend, par exemple, d'abord l'OUVRAGE et le joint avec OUVAUT, alors selon la nature de la jointure (production + projection), il va lier chaque tuplet d'OUVRAGE avec les records dont NUMAU vaut 245 dans OUVAUT, et puis faire la projection sur NUMOU. Par cet exemple, nous pouvons voir que les records à référencer sont considérablement plus nombreux.

Si l'"optimiser" prend d'abord OUVAUT et fait la projection sur le NUMAU de valeur 245, et enfin fait la jointure sur le domaine commun NUMOU d'OUVRAGE, les records références seront beaucoup moins nombreux.

Même si RDB choisit la meilleure des deux façons décrites ci-dessus, le prg-5.5 est toujours plus lent que le prg-5.4. C'est parce que l'accès par index a besoin de plus d'accès physiques pour parcourir l'arbre d'index que l'accès par chemin qui a un poniteur direct. Ceci peut expliquer les résultats obtenus dans le TAB-4.13.

c) En cas de stockage des données dans la BD, remarquons que le programme de l'application 25 (programme de chargement) en DBMS est effectivement plus coûteux que son équivalent en RDB.

Quatre facteurs concernant le stockage des records dans la BD en DBMS doivent être notés:

(1) On doit connaître le mode (automatic ou manual) du set ou les records vont être stockés. Une restriction, cependant dévalorise la performance du stockage des records: il n'y a aucun type de record qui puisse être un OWNER record d'un set sans être un record membre d'autre set (cfr. la remarque du schéma-3). Quand on stocke un ouvrage par exemple, dans le type de record OUVRAGE, le SGBD doit établir en même temps le set du système où l'OUVRAGE est un membre record.

(2) Trois pointeurs seront établis (OWNER, PRIOR et NEXT) au moment où l'on stocke les records dans un set. Le SGBD doit donc à ce moment-là tenir compte de l'ordre d'insertion de ces records (FIRST ou LAST).

(3) Lorsque le set est défini en mode index il faut mettre à jour l'index.

(4) Le SGBD doit vérifier la contrainte d'intégrité imposée par l'utilisateur sur le set est les items.

En RDB, le SGBD prend les derniers facteurs ((3) et (4)) en charge ainsi que la mise à jour du pointeur des relations du système. Bref, l'ajoût des records dans la BD en RDB est moins coûteux qu'en DBMS.

L'analyse ci-dessus nous permet de dire que, bien que la structure de données en CODASYL soit complexe, elle permet de renforcer la performance (surtout pour les applications de type complexe).

Il semble que nous ayons aussi expliqué pourquoi en RDB les programmes de certains types ont gagné du temps CPU, mais n'ont pas gagné de temps total écoulé.

5.4.4 Comportement des SGBD

Nous remarquons dans les tableaux du point 4.4, qu'il y a une forte dégradation du temps pour les applications en RDB 18, 20 et 23 par rapport aux mêmes applications en DBMS. Ceci est dû aux comportements des SGBD vis-à-vis de la mise à jour de la BD par le programme.

Dans le schéma-4, la contrainte 7 exige de l'utilisateur de supprimer explicitement dans son programme les records EMPRUNT-ARCH qui ont un même numéro d'un exemplaire qui va être supprimé.

Néanmoins, en DBMS, (voir chéma-3), lorsque l'utilisateur décide de supprimer un exemplaire, il n'est pas obligé d'utiliser explicitement l'instruction ERASE pour supprimer les records EMPRUNT-ARCH qui sont liés à cet exemplaire. C'est le SGBD qui s'occupe de l'intégrité de la BD dans ce cas-là. Il lui suffit que le SGBD retire le repère de cette occurrence du set EMP et non pas de les supprimer physiquement. Cela peut donc être très efficace.

Nous signalons, à la fin de notre comparaison de la performance, que dans le cadre d'RDB, il y a un choix qui est offert au programmeur:

- optimisation des accès confiée au SGBD (optimisation implicite);

- optimisation des accès décidée par lui-même (optimisation explicite).

Nous avons choisi quelques applications (voir TAB-4.10bis et TAB-4.12bis) pour tester avec quelle optimisation le programme est le plus efficace. Nous trouvons que les résultats sont sensiblement les mêmes. Le programme de l'application 5 avec l'optimisation explicite pour la BD de grande taille produit moins d'E/S. Mais le temps CPU pour ce programme est plus important. L'optimisation explicite n'entraîne pas donc de gain.

Le fait que les résultats obtenus sont sensiblement les mêmes provient certainement plus du choix des applications tests que d'une similitude entre les résultats des deux méthodes d'optimisation proposées.

Il existe en effet de bonnes raisons pour préférer dans certain cas une optimisation explicite [HAIN 86].

Nous pensons qu'il serait intéressant de comparer de manière plus approfondie ces deux possiblités. Etant donné que le sujet principal de ce mémoire est la comparaison de 2 SGBD de types différents et du manque de temps, nous n'avons pas pu réaliser d'autre tests qui permettraient de comparer plus en profondeur l'influence du choix entre l'une et l'autre méthode.

5.5 ESTIMATION DU VOLUME ET DU TAUX DE REMPLISSAG DE LA BD

Au vu de TAB-4.13 et TAB-4.15, nous remarquons que:

- les fichiers (types d'article) en RDB occupent effectivement moins de place qu'en DBMS. En effet le SGBD compresse automatiquement les données quand on les charge dans la BD.

En DBMS, on peut condenser les chaînes de caractères si l'on définit dans le STORAGE-SCHEMA l'item avec la clause "LOCATION IS DYNAMIC". Mais pour des raisons de performance, il vaut mieux ne pas condenser l'item qui est une clé calculée.

- Si l'on consulte le TAB-4.14 et le TAB4-.16 on peut voir que le volume total de la BD/DBMS est beaucoup plus petit que la BD/RDB. Cela signifie que le SGBD/RDB a besoin de plus de donnees de gestion, par exemple des données pour implanter les index, les relations systèmes, les contraintes d'intégrité etc.

En examinant les volumes statiques des BD, il nous est permis de dire que la simplicité de la structure en RDB (voir 4.1 et 5.1) envisagée par les utilisateurs est au détriment des places de stockage. Ceci signifie implicitement une complexité de la gestion du système lui-même.

Nous notons que, malheureusement les informations que nous avons obtenues ne nous permettent pas d'analyser plus en profondeur la structure de données internes du RDB. Dans le manuel [RDB 84], les informations suffisantes pour calculer et évaluer rigoureusement le volume de la BD ne sont pas données.

On peut toutefois de façon assez réaliste calculer le volume statique de la BD/DBMS selon le schéma logique, le schéma physique, les éléments de quantification prédéfinis, et la représentation des données physiques qui est décrite dans le manuel [DBMS 84]. Si le lecteur est intéressé, il peut faire référence à l'exemple de calcul du volume de la BD/DBMS pour la BD de taille réduite développé à l'annexe-2.

En ce qui concerne le taux de remplissage, nous trouvons qu'il est raisonnable pour les deux BD juste après la création de la BD.

En effet, il est facile pour DBA/DBMS de charger les données, que le taux de remplissage soit élevé ou non. Il suffit de définir dans le STORAGE-SCHEMA le mode de placement comme "clustered" ou "scattered".

Il faut le faire bien entendu selon les caractéristiques de ses applications. Nous avons par exemple, dans notre expérience, volontairement laissé le taux de remplissage de l'AREA FEMPR assez faible, parce qu'au cours du temps nous allons charger les records de EMPRUNT-ARCH qui se trouve dans cette AREA.

En revanche, nous n'avons pas le choix quant au contrôle du taux de remplissage en RDB: le SGBD en prend soin.

CHAPITRE 6
CONCLUSIONS

Il est temps de synthétiser notre expérience .

Dans la section 6.1, nous résumerons très brièvement les analyses du chapitre précédent sur les cinq critères.

Nous essayerons dans la section 6.2, d'estimer l'adéquation des SGBD RELATIONNEL (RDB) et SGBD CODASYL (DBMS) à différents domaines d'application.

La section 6.3 est consacrée à un résumé de ce que nous avons fait afin d'évaluer le travail réalisé.

La section 6.4 va proposer quelques possibiltés d'extension de notre travail.

6.1 SYNTHESE DES COMPARAISONS

Après avoir analysé les résultats des mesures au chapitre précédent, nous voudrions encore dire quelques mots en guise de conclusion sur les cinq critères de comparaison.

1) Puissance du modèle de données

La structure de données de DBMS semble plus complexe qu'en RBD. Mais il est facile de représenter les associations dites complexes sans introduire de données redondantes.

La représentativité de la structure de données à partir du schéma E/A est la même pour les deux modèles. En effet les deux modèles peuvent être transformés l'un dans l'autre.

Il est évident que:

a) concernant le modèle RDB à DBMS: un ensemble de relations normalisées en RDB est aussi un ensemble de types de records en DBMS et les tuplets d'une relation sont les instances d'un type de record.

b) et le modèle DBMS à RDB: chaque set en DBMS peut être vu comme une association binaire entre l'identifiant d'un OWNER record et l'identifiant d'un MEMBER record. Quand un set a plus d'un MEMBER records il faut définir une relation entre chacun de ces MEMBER records et leur même OWNER record.

Avec RDB, on peut facilement réaliser l'unicité d'un type de record. Par opposition, il est irréaliste en DBMS, d'implanter un identifiant dans un type de record qui soit à l'origine une association de type "many-to-many".

2) Indépendance des données

Les deux SGBD permettent d'élargir la BD sans affecter les programmes d'application.

RDB offre plus d'indépendance, tandis que DBMS est très modeste, surtout sur le plan de l'indépendance physique.

3) Degré algorithmique des programmes

En ce qui concerne le degré algorithmique des programmes d'application, nous nous sommes aperçu que:

a) la procéduralité du DML en DBMS découle de plusieurs facteurs:

- de l'utilisation du connecteur booléen "if";

- de la connaissance de la façon d'accéder aux objets de BD;

- de la conscience de la localisation des objets désirés;

- à cause de la logique d'accès aux données "one-record-at-a-time", l'instruction du DML FIND ne trouve qu'une instance de record chaque fois qu'elle est appelée par la programme. Par conséquent, le programmeur doit exécuter une séquence d'instructions FIND afin d'obtenir les informations désirées.

b) RDB offre un langage DML qui est un langage de haut niveau, non procédural et très puissant. Mais le haut niveau peut être amené à un niveau plus bas à cause de la mauvaise compatibilté avec le COBOL. Cela signifie d'ailleurs qu'il n'est pas approprié pour toutes les applications.

4) Performance des SGBD

Il y a une tendance très évidente à remarquer: que ce soit du point de vue du temps d'exécution, du temps CPU ou de l'accès physique, pour une application, plus complexe est la taille de l'objet à traiter, plus avantageux est le programme/DBMS de cette application. Ceci est dû aux faits suivant:

- La structure de données complexe permet d'accéder aux données de manière plus efficace.

- Le DBMS offre plus de possibilités de définir les clés d'accès.

En ce qui concerne l'efficacité des programmes, normalement DBMS l'emporte sur RDB. Cela vient de ce que ces programmes produisent moins d'opérations d'E/S et qu'ils peuvent choisir le mécanisme d'accès le plus performant, par exemple accès calcul au lieu d'accès par index.

Selon l'analyse du point 5.4, nous ne pouvons pas simplement dire que les programmes/DBMS l'emportent sur les programmes/RDB. En effet, pour ce qui est "élémentaire, de masse, et avec accès séquentiel filtré", les programmes/DBMS sont plus couteux.

Nous croyons que, dès lors qu'une méthode qui réduirait le temps d'E/S [HAWT 79] sera réalisée et, dès lors que l'on pourrait créer un "optimiser" dit plus "intelligent", la performance des programmes/RDB pourrait s'améliorer.

5) Volume et taux de remplissage de la BD

a) RDB permet de condenser automatiquement les données dans les fichiers de la BD. Quand au DBMS, on doit spécifier explicitement les items que l'on veut condenser.

b) DBMS offre à l'utilisateur la possibilté de contrôler le taux de remplissage selon ses propres besoins. Ce n'est pas le cas d'RDB.

c) La simplicité de la structure de données observée par les utilisateurs se solde par une augmentation de place sur le disque pour les données de gestion.

6.2 DOMAINES D'APPLICATION DE CHAQUE SGBD

Au vu des analyses sur l'aspect logique presentees au chapitre précédent, nous pouvons dire que:

- lorsque DBMS ne permet pas la restructuration du schéma de la BD, il convient d'utiliser DBMS quand on possède un schéma conceptuel relativement stable; c'est-à-dire, lorsqu'au cours du temps, la possibilité de modification du schéma est relativement faible.

Beaucoup d'entreprises changent au cours du temps leur structure organisationnelle. Chaque fois qu'une modification y est apportée, le schéma conceptuel du système informatique risque d'être changé. Le RDB offre plus de souplesse pour la réalisation de ces modifications. Les utilisateurs ne doivent pas refaire ce qui a déjà été fait pour certaines applications.

- Selon leurs besoins, les utilisateurs peuvent ajouter dynamiquement des index, des relations ainsi que les supprimer dans l'environnement RDB.

Lorsque les utilisateurs de RDB voient les données représentées sous la forme de tableau, ils peuvent n'avoir aucune connaissance du SGBD lui-même. Il suffit donc de peu d'entraînement de la part des utilisateurs pour maintenir une BD/RDB.

Une entreprise qui n'a pas d'administrateur professionnel et voudrait informatiser son système d'information sera intéressée par le RDB.

- Un schéma conceptuel d'une entreprise est constitué d'objets entre lesquels il existe des associations complexes. Si la proportion d'applications de type transactionnel et complexe est élevé par rapport à des applications de type de masse et simple, il est conseillé aux utilisateurs d'utiliser le SGBD CODASYL. A long terme, le gain de performance sera considérable.

6.3 CONCLUSION SUR LA DEMARCHE

6.3.1 Conclusion générale

L'évaluation comparative de SGBD de type distincts diffère de l'évaluation d'un seul SGBD.

Dans un premier temps, nous devons contrôler chaque SGBD à comparer tout en respectant ses structures de données et les restrictions d'implémentation de la BD à partir d'une référence commune. Notre étude montre que, étant donne un systeme d'information réel qui est, par exemple, décrit par le modèle E/A en tant que référentiel commun, le MAG permet une approche homogène, identique et automatique pour établir les schémas des SGBD différents sans perdre les informations du système réel.

Dans un second temps, nous devons déterminer sous quel angle et de quelle façon la comparaison se poursuit. Nous croyons et nous avons montré expérimentalement qu'il est pertinent de faire la comparaison en exploitant les applications réelles (les besoins d'utilisateurs) et leurs effets sur les activités gérées par le SGBD.

Dans un troisième temps, nous devons nous servir des mesures ainsi effectuées sur les systèmes réels. Pour fournir les charges aux systèmes, nous devons définir des charges qui soient assez générales et indépendantes des systèmes. Dans notre expérience nous avons classifié les applications qui sont liées à la manipulation des données de BD.

N'ayant pas la prétention de faire une comparaison complète et exhaustive, nous avons examiné les SGBD RELATIONNEL et CODASYL en particulier RDB et DBMS sur deux aspects.

a) L'aspect logique

Après les analyses conceptuelles, nous avons essayé de faire des mesures sur l'aspect logique de la façon la plus "quantitative" possible. Il vaut mieux expliquer les différences et extrapoler les conclusions en partant des chiffres et des faits. Nous avons, par exemple, défini quelques paramètres quantitatifs pour mesurer le degré algorithmique des programmes. Les mesures "parlent" donc assez d'elles-mêmes.

b) L'aspect technique

Nous avons présenté une méthodologie pour evaluer la performance des SGBD dans un environnement mono-utilisateur. Nous avons identifié quatre types d'analyse essentielles pour classifier les applications: type d'opération, taille de l'objet à traiter, nombre d'objets à traiter et mécanisme d'accès utilisé. Selon ces types, nous avons construit une série de programmes d'application dans le but de mesurer la performance des SGBD.

6.3.2 L'évaluation du travail réalisé

L'évaluation du travail consiste à relever les difficultés que nous avons rencontrées et valider les acquis.

a) Les difficultés de la démarche

- La première difficulté que nous avons rencontrée est la détermination des critères de comparaison correspondant à ce que nous voulions mesurer, confrontés à tant de caratéristiques du SGBD.

Nous avons finalement opté pour deux aspects de comparaison: les aspects logique et technique. Représentent-t-ils largement et principalement des besoins d'utilisateurs?

- Le choix des paramètres significatifs qui correspondent aux critères de comparaison est une autre difficulté. D'une part, il en est ainsi, quant à l'aspect logique du SGBD. La détermination des paramètres dépend fortement de la connaissance des concepts que l'on possède. D'autre part, l'évaluation de la performance pose des problèmes de mesurabilité. Nous avons dû chercher des utilitaires que nous pouvions employer dans une installation spécifique pour mesurer la performance. Cela veut dire que si l'on veut faire les expériences les plus générales possible par souci de portabilité, on est limité par l'environnement spécifique.

- Nous avons réussi, à partir d'un modèle de référence, à contrôler des conditions de comparaison à un plus haut niveau: la transformation des modèles de données spécifiques. Mais la recherche du contrôle des conditions expérimentales vaut toujours la peine d'être poursuivie.

Le choix des paramètres physiques a été particulièrement délicat. Par exemple, en DBMS, il y a deux paramètres BUFFERS et LENGTH_BUFFER qui déterminent la taille du pool des tampons de la mémoire centrale et qui conditionnent la performance du système. Pour avoir une performance raisonnable, il faut faire des essais afin de déterminer les valeurs de ces paramètres.

Au début de notre expérience, nous avons chargé la BD BIBLIO/DBMS de taille réduite. En examinant le temps consommé par le programme de chargement, nous avons remarqué qu'il s'écoule presque une demi-heure. Par opposition, le programe/RDB qui a chargé la même quantité de données ne consomme que quelques minutes. Après avoir changé simplement deux valeurs de ces paramètres pour le programme/DBMS, nous avons pu obtenir un temps qui l'emporte sur celui du programme/RDB.

b) La validation des mesures

Nous ne parlerons ici que des mesures de performance. Nous l'examinerons en deux phases:

- La première est la validation de la méthode. Nous voudrions dire que la méthode est valide si les types d'analyse sont indépendants des systèmes. C'est-à-dire que les types sont applicables d'un système à l'autre.

Du point de vue des applications qui manipulent les données de la BD, la plupart des types d'application que nous avons définies au chapitre 3 sont indépendants des systèmes.

Mais en ce qui concerne le mécanisme d'accès aux données, remarquons que RDB n'offre qu'un mécanisme d'accès par index, tandis que DBMS en offre plusieurs (accès calcul, accès par index et accès par chemin). Or, nous devons baser notre comparaison sur la même possibilité d'utilisation du mécanisme d'accès. C'est-à-dire que le programme d'application en DBMS qui accède aux données par un chemin ou par clé calculée est comparable au programme d'application en RDB, qui accède aux mêmes données par clé indexée. Notons que ceci est garanti au stade de la transformation des modèles de données.

- La deuxième est la validation qui consiste à tester les résultats obtenus. Nous considérons qu'un résultat comparatif sur une application z est valide, si, dès lorsque l'on exécute deux programmes x, y de cette application, en changant le moment des exécutions des programmes dans des conditions équivalentes, on peut obtenir le même résultat. Le résultat comparatif a la forme suivante:

le programme x de l'application z l'emporte sur le programme y de cette même application.

L'inverse donnera le résultat différent.

Pour vérifier finalement les résultats présentés aux TAB-4.9 et TAB-4.10, nous avons attendu quelques semaines pour refaire les mêmes mesures dans des conditions identiques. Nous nous sommes aperçu que sauf pour deux ou trois applications, par exemple, 14 et 15, nous avons obtenu le même résultat comparatif.

6.4 EXTENSIONS

Il sera intéressant de voir, à présent, les possibilités d'extension de notre travail.

6.4.1 Environnement multi-utilisateurs

Lorsque plusieurs programmes fonctionnent avec le même SGBD, le problème de concurrence appparaît. Nous pouvons imaginer dans cet environnement que le temps écoulé pour chaque programme sera plus élevé que pour celui qui fonctionne tout seul dans l'environnement spécifique. Il est donc intéressant pour certains utilisateurs de savoir quel SGBD est plus puissant et peut accepter plus d'utilisateurs dans une tranche de temps déterminée.

Du point de vue des effets d'applications sur le système, les types d'analyse que nous avons classés dans le chapitre 3 sont évidemment aussi applicables. Nous pouvons exécuter un nombre différent de programmes de mêmes types, de types mélangés ou de types tout à fait différents dans un environnement spécifique. En notant le nombre de programmes en concurrence, après chaque exécution, il est possible de mesurer le temps de réponse de chaque programme soit grâce à l'horloge soit grâce aux utilitaires du système existant. Les résultats de ce genre de mesure peuvent être illustrés dans un graphe à deux dimensions. L'abscisse représente le nombre de programmes en concurrence et l'ordonnée représente le temps de réponse de chaque exécution. Le lecteur peut se référer à [BORA 84].

6.4.2 Mesure du degré de données partagées

Il est possible de mesurer le degré de données partagées pour les différents SGBD avec notre méthode dans l'environnement multi-utilisateurs. Par exemple, étant donné une application de type déterminé, plusieurs copies du même programme de cette application sont produites et exécutées concurremment. Dans ce cas-là, on sait que tous les programmes exécutés accèdent aux même données: données fortement partagées. Le temps de reponse de chaque exécution est ensuite enregistré. On peut alors établir les performances selon le degré de partage des données.

6.4.3 Au niveau physique

Lorsque chaque type de transaction est caractérisé par le mécanisme d'accès aux données, une fois la stratégie d'accès aux données déterminée pour un fichier de BD, il est possible de comparer les index implementés dans des SGBD différents. Si les index sont les mêmes (par exemple B-arbre), il est possible de voir quel SGBD utilise l'index de façon la plus efficace dans des conditions équivalentes.

6.4.4 Choix des optimisations d'accès des programmes/RDB

Nous avons abordé ce point au chapitre 5. Pour approfondir cette étude, il faudrait choisir des applications de type plus complexe, et les développer avec deux optimisations: implicite ou explicite.

REFERENCES ET BIBLIOGRAPHIE

[BORA 84] Haran BORAL
"A methodology for database system performance evaluation"
ACM SIGMOD 1984

[BRAC 75] G. BRACCHI, P.PAOLINI, G. PELAGATTI
"Data independent descriptions and the DDL specifications"
Data Base Descriptions. C.M DOUQU2 and G.M.NIJSSEN (eds)
North-Holland Publishing Company, 1975

[CHAR 71] Donald.D. CHAMBERLIN; Arthur.M. GILHERT and Robert.A. YOST
"A history of system R and SQL/DATA system"
IBM Research Laboratory San Jose. California 95193
IEEE 1981

[COBO 82] VAX-11 COBOL User guider
Digital Equipement Corporation Maynard, Massachusetts 1982

[CODD 74] E.CODD and C.J DATE
"The relational and network approches: Comparison of the application programming interfaces"
Proc. ACM-SIGFIDET Workshop on Data Description, Acces and Control. Ann Arbort, Mich., May 1974

[DATE 84] C.J. DATE
"Some principles of good language-with special reference to design of database languages"
ACM SIGMOD Record volume 14 number 3 1984

[DATE 81] C.J. DATE
" An introduction to database systems "
Third edition VOL.1
Addison-wesley Publishing Company 1981

[DBMS 84] VAX DBMS Manual
Digital Equipement Corporation, Maynard, Massachusetts 1984.

[DELC 87] A. DELCOURT, M.CADELLI, C.CHARLOT, J-L HAINAUT
"Conception d'une base de données - éléments méthodologiques"
Namur, Mars 1987

[DELO 82] Claude DELOBEL et Michel ADIBA
"Base de données et systèmes relationnels"
Bordas Paris 1982.

[ENGL 71] Robort W. ENGLES
"An analyses of the April 1971 data base task group report"
ACM SIGFIDET Workshop Data Description, Acces and Control
Edited by E.F CODD and A.L DEAN,
San Diego California , NOV.1971

[FERR 81] Domenico FERRARI, Giuseppe SERAZZI, Alessandro ZEIGNER
"Measurement and turning of computer system"
Prentice-Hall. INC.Englewood Cliffs N.J. 0763 1981

[HAIN 77] J-L HAINAUT
"Some tools for data indepence in multilevel
data base system"
Architecture and Model in Data Base Management System
North-Holland Publishing Company 1977

[HAIN 81] J-L HAINAUT
"Production de programmes ADL en COBOL/DML CODASYL 71
application au système DBMS-20"
Institut d'Informatique à Namur 1985

[HAIN 85] J-L HAINAUT
"Introduction aux systèmes de gestions
de base de données CODASYL 71"
Namur Octobre 1985

[HAIN 84] J-L HAINAUT
"Programmation d'application sur base de données"
Institut d'Informatique à Namur 1984

[HAIN 86] J-L HAINAUT
"Conception assistée des applications informatiques
2 conception de la base de données"
MASSON Presses Universitaires de NAMUR 1986

[HAIN 87] J-L HAINAUT
"Introduction à la théorie relationnelle des bases de données"
Note du cours
Facultés Universitaires Notre-Dame de la Paix à Namur 1987

[HAWT 79] Paul HAWTHORN and Michael STONEBRAKER
"Performance analyse of a relational data base management system"
Proc. ACM-SIGMOD International Conference On Management of Data . May 30 - Juni 1979

[HUTT 79] A.T.F. HUTT
"A relational data base management system"
Tohn Wiley and Sons 1979

[KERS 76] L. KERSCHBERG, A.KING and D.TSICHNTZIS
"A taxonomy of data models"
System for Large Data Base, P.C LOCKEMANN and E.J NEUHOLD (eds). North-Holland Publishing Company 1976

[KIN 79] Won KIN
"Relational database systems"
ACM Computing Surveys Vol.11 N° 3 September 1979

[LOCH 77] F.H. LOCHOUSKY, D.C.TSICHRITZIS
"User performance considerations in DBMS selection"
ACM SIGMOD Proceeding International Coference On Management Of Data, Toronto, Canada 1977

[MICH 76] A.S MICHAELS, B.MITTMAN and C.R CARLSON
"A comparison of the relational and CODASYL approaches to data base management"
ACM Computing Survey 8, N°.1 (March 1976)

[NIJS 74] C.M NIJSSEN
"Data structuring in DDL and the relational data model"
Proc. IFIP TC-2 Working Conference on Data Base Management System , April 1974

[PERR 80] Robert PERRON
"Design guid for CODASYL data base management systems"
Q.E.D. Information Sciences. INC.
Welleley, Massachusetts 1980

[RDB 84] VAX-11 RDB/VMS Manual
Digital Equipement Corporation, Maynard,
Massachusett s 1984.

[STON 75] Michel STONEBRAKER and Gerald HELD
"Networks, hierarchies and relations in data base
management system"
ACM Pacific 1975

[SHEL 78] A.SHELDON ,'
"Data model equivalence"
Proceeding Very Large Data Base 1978

[WAGH 75] W.J. WAGHON
"The DDL as an industry standard ?"
Data Base Description B.C.M. DOUQUE and G.M.NIJSSEN (eds)
North-Holland Publishing Company 1975

[WILL 74] J. WILLIAM
"Data definition spectrum and procedurality spectrum"
Data Base Management System
North-Holland Publishing Company 1974

[WONG 81] Patrick M.K. WONG
"Performance evaluation of data base systems"
UMI Research Press 1981

ANNEXE-1

TEXTES DES SCHEMAS DES BASES DE DONNEES

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*
*     A)  BIBLIO/DBMS
*
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* 1) SCHEMA LOGIQUE DE LA BD/DBMS
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

SCHEMA NAME IS BIBLIO1

AREA NAME IS FAUTEUR
AREA NAME IS FOUV
AREA NAME IS FEXMP
AREA NAME IS FEMPR

***************************************************
* DEFINITION DES RECORDS
***************************************************
RECORD NAME IS AUTEUR
    WITHIN FAUTEUR
    ITEM NUMAU TYPE SIGNED WORD
    ITEM NOMAU TYPE CHARACTER 30
    ITEM ORIGINE TYPE CHARACTER 30

RECORD NAME IS OUVAUT
    WITHIN FAUTEUR

RECORD NAME IS OUVRAGE
    WITHIN FOUV
    ITEM NUMOU TYPE SIGNED WORD
    ITEM TITRE TYPE CHARACTER 30
    ITEM ANNEE TYPE CHARACTER 4
    ITEM EDITEUR TYPE CHARACTER 30

RECORD NAME IS MCOUV
    WITHIN FOUV
    ITEM MOTCLE TYPE CHARACTER 30

RECORD NAME IS EXEMPLAIRE
    WITHIN FEXMP
    ITEM NUMEX TYPE SIGNED WORD
    ITEM ETAGE TYPE SIGNED WORD
    ITEM TRAVEE TYPE SIGNED WORD
    ITEM RAYON TYPE  SIGNED WORD

RECORD NAME IS EMPRUNT
    WITHIN FEMPR
    ITEM DATEDEBUT TYPE SIGNED LONGWORD

RECORD NAME IS EMPRUNTARCH
    WITHIN FEMPR
    ITEM DATE_DEBUT TYPE SIGNED LONGWORD
    ITEM DATE_FIN TYPE SIGNED LONGWORD

RECORD NAME IS EMPRUNTEUR
    WITHIN FEMPR
    ITEM NUMEM TYPE SIGNED WORD
    ITEM NOMEM TYPE CHARACTER 30
```

```
    ITEM ADRESSE TYPE CHARACTER 30


*****************************************************
* DEFINITION DES SETS
*****************************************************
SET NAME IS SYSTAUTEUR
    OWNER IS SYSTEM
    MEMBER IS AUTEUR
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED
    ORDER IS SORTED BY ASCENDING NUMAU
    DUPLICATES ARE NOT ALLOWED

SET NAME IS SYSTNOMAU
    OWNER IS SYSTEM
    MEMBER IS AUTEUR
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED

SET NAME IS SYSTMOTCLE
    OWNER IS SYSTEM
    MEMBER IS MCOUV
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED

SET NAME IS SYSTOUV
    OWNER IS SYSTEM
    MEMBER IS OUVRAGE
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED

SET NAME IS SYSTEXEMP
    OWNER IS SYSTEM
    MEMBER IS EXEMPLAIRE
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED
    ORDER IS SORTED BY ASCENDING NUMEX
    DUPLICATES ARE NOT ALLOWED

SET NAME IS SYSTEMPR
    OWNER IS SYSTEM
    MEMBER IS EMPRUNTEUR
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED
    ORDER IS SORTED BY ASCENDING NUMEM
    DUPLICATES ARE NOT ALLOWED

SET NAME IS AO
    OWNER IS AUTEUR
    MEMBER IS OUVAUT
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY

SET NAME IS OO
    OWNER IS OUVRAGE
    MEMBER IS OUVAUT
    INSERTION IS AUTOMATIC
```

```
    RETENTION IS MANDATORY

SET NAME IS OMC
    OWNER IS OUVRAGE
    MEMBER IS MCOUV
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY

SET NAME IS OE
    OWNER IS OUVRAGE
    MEMBER IS EXEMPLAIRE
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY

SET NAME EXE
    OWNER IS EXEMPLAIRE
    MEMBER IS EMPRUNT
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY

SET NAME IS EMP
    OWNER IS EMPRUNTEUR
    MEMBER IS EMPRUNT
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY

SET NAME IS EXEA
    OWNER IS EXEMPLAIRE
    MEMBER IS EMPRUNTARCH
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY

SET NAME IS EMPA
    OWNER IS EMPRUNTEUR
    MEMBER IS EMPRUNTARCH
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY
```

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*   2)  SOUS-SCHEMA DE LA BD/DBMS
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

SUBSCHEMA NAME IS BIBLSUB FOR BIBLIO1 SCHEMA

REALM FAUTEUR
    IS FAUTEUR

REALM FOUV
    IS FOUV

REALM FEXMP
    IS FEXMP

REALM FEMPR
    IS FEMPR

RECORD NAME IS AUTEUR
    ITEM NUMAU TYPE IS SIGNED WORD
    ITEM NOMAU TYPE IS CHARACTER 30
    ITEM ORIGINE TYPE IS CHARACTER 30

RECORD NAME IS OUVAUT

RECORD NAME IS OUVRAGE
    ITEM NUMOU TYPE IS SIGNED WORD
    ITEM TITRE TYPE IS CHARACTER 30
    ITEM ANNEE TYPE IS CHARACTER 4
    ITEM EDITEUR TYPE IS CHARACTER 30

RECORD NAME IS MCOUV
    ITEM MOTCLE TYPE IS CHARACTER 30

RECORD NAME IS EXEMPLAIRE
    ITEM NUMEX TYPE IS SIGNED WORD
    ITEM ETAGE TYPE IS SIGNED WORD
    ITEM TRAVEE TYPE IS SIGNED WORD
    ITEM RAYON TYPE IS SIGNED WORD

RECORD NAME IS EMPRUNT
    ITEM DATEDEBUT TYPE IS SIGNED LONGWORD

RECORD NAME IS EMPRUNTARCH
    ITEM DATE_DEBUT TYPE IS SIGNED LONGWORD
    ITEM DATE_FIN TYPE IS SIGNED LONGWORD

RECORD NAME IS EMPRUNTEUR
    ITEM NUMEM TYPE IS SIGNED WORD
    ITEM NOMEM TYPE IS CHARACTER 30
    ITEM ADRESSE TYPE IS CHARACTER 30

SET NAME IS SYSTAUTEUR

SET NAME IS SYSTNOMAU

SET NAME IS SYSTMOTCLE

SET NAME IS SYSTOUV
```

```
SET NAME IS SYSTEXEMP

SET NAME IS SYSTEMPR

SET NAME IS AO

SET NAME IS OO

SET NAME IS OMC

SET NAME IS OE

SET NAME IS EXE

SET NAME IS EMP

SET NAME IS EXEA

SET NAME IS EMPA
```

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*   3)  SCHEMA PHYSIQUE DE LA BD/DBMS
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

STORAGE SCHEMA NAME IS BIBLST FOR BIBLIO1 SCHEMA

RECORD NAME IS AUTEUR
    PLACEMENT IS CLUSTERED VIA SYSTAUTEUR
    ITEM NUMAU TYPE IS SIGNED WORD
    ITEM NOMAU ALLOCATION IS DYNAMIC
         TYPE IS CHARACTER 30
    ITEM ORIGINE ALLOCATION IS DYNAMIC
         TYPE IS CHARACTER 30

RECORD NAME IS OUVAUT
    PLACEMENT IS CLUSTERED VIA AC

RECORD NAME IS OUVRAGE
    PLACEMENT IS SCATTERED USING NUMOU
    ITEM NUMOU TYPE IS SIGNED WORD
    ITEM TITRE TYPE IS CHARACTER 30
    ITEM ANNEE TYPE IS CHARACTER 4
    ITEM EDITEUR TYPE IS CHARACTER 30

RECORD NAME IS MCOUV
    PLACEMENT IS CLUSTERED VIA OMC
    ITEM MOTCLE TYPE IS CHARACTER 30

RECORD NAME IS EXEMPLAIRE
    PLACEMENT IS CLUSTERED VIA SYSTEXEMP
    ITEM NUMEX TYPE IS SIGNED WORD
    ITEM ETAGE TYPE IS SIGNED WORD
    ITEM TRAVEE TYPE IS SIGNED WORD
    ITEM RAYON TYPE IS SIGNED WORD

RECORD NAME IS EMPRUNT
    PLACEMENT IS CLUSTERED VIA EMP
    ITEM DATEDEBUT TYPE IS SIGNED LONGWORD

RECORD NAME IS EMPRUNTARCH
    PLACEMENT IS CLUSTERED VIA EMPA
    ITEM DATE_DEBUT TYPE IS SIGNED LONGWORD
    ITEM DATE_FIN TYPE IS SIGNED LONGWORD

RECORD NAME IS EMPRUNTEUR
    PLACEMENT IS CLUSTERED VIA SYSTEMPR
    ITEM NUMEM TYPE IS SIGNED WORD
    ITEM NOMEM TYPE IS CHARACTER 30
    ITEM ADRESSE TYPE IS CHARACTER 30


************************************************************
* LES SETS ET SES MODES
************************************************************

SET NAME IS SYSTAUTEUR
    MODE IS INDEX
         NODE SIZE IS 240 BYTES
```

```
SET NAME IS SYSTNOMAU
    MODE IS CALC
        MEMBER IS AUTEUR
            KEY IS NOMAU

SET NAME IS SYSTMOTCLE
    MODE IS CALC
        MEMBER IS MCOUV
            KEY IS MOTCLE

SET NAME IS SYSTOUV
    MODE IS CALC
        MEMBER IS OUVRAGE
            KEY IS NUMOU

SET NAME IS SYSTEXEMP
    MODE IS INDEX
        NODE SIZE IS 475 BYTES

SET NAME IS SYSTEMPR
    MODE IS INDEX
        NODE SIZE IS 116 BYTES

SET NAME IS AO
    MODE IS CHAIN

SET NAME IS OO
    MODE IS CHAIN

SET NAME IS OMC
    MODE IS CHAIN

SET NAME IS OE
    MODE IS CHAIN

SET NAME IS EXE
    MODE IS CHAIN

SET NAME IS EMP
    MODE IS CHAIN

SET NAME IS EXEA
    MODE IS CHAIN

SET NAME IS EMPA
    MODE IS CHAIN
```

```
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!     B)  BIBLIO/RDB
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

!***********************************************
! Creation de la base de donnees biblio
!************************************************

DEFINE DATABASE "BIBLIO" IN "CDD$TOP.INF.MEM.SWU.BIBLIO"
                 NUMBER OF BUFFERS IS 18
                 BUFFER SIZE IS 28.

!**************************************************
! Define fields for the biblio database
!**************************************************

DEFINE FIELD TITRE
       DESCRIPTION IS /* TITRE DE L'OUVRAGE */
       DATATYPE IS TEXT SIZE IS 30.

DEFINE FIELD ANNEE
       DESCRIPTION IS /* ANNEE D'EDITION DE L'OUVRAGE */
       DATATYPE IS TEXT SIZE IS 4.

DEFINE FIELD NUMOU
       DESCRIPTION IS /* NUMERO DE L'OUVRAGE */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD EDITEUR
       DESCRIPTION IS /* EDITEUR DE L'OUVRAGE */
       DATATYPE IS TEXT SIZE IS 30.

DEFINE FIELD NOMAU
       DESCRIPTION IS /* NOM DE AUTEUR */
       DATATYPE IS TEXT SIZE IS 30.

DEFINE FIELD NUMAU
       DESCRIPTION IS /* NUMERO DE L'AUTEUR */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD NUMEX
       DESCRIPTION IS /* NUMERO DE L'EXEMPLAIRE */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD NUMEM
       DESCRIPTION IS /* NUMERO DE L'EMPRUNTEUR */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD DATEDEB
       DESCRIPTION IS /* DATE STANDARD */
       DATATYPE IS SIGNED LONGWORD.

DEFINE FIELD NOMEM
       DESCRIPTION IS /* NOM DE L'EMPRUNTEUR */
       DATATYPE IS TEXT SIZE IS 30.

DEFINE FIELD ORIGINE
       DESCRIPTION IS /* ORIGINE DE L'AUTEUR */
       DATATYPE IS TEXT SIZE IS 30.
```

```
DEFINE FIELD ADRESSE
       DESCRIPTION IS /* ADRESSE DE L'EMPRUNTEUR */
       DATATYPE IS TEXT SIZE IS 30.

DEFINE FIELD ETAGE
       DESCRIPTION IS /* ETAGE OU SE TROUVE L'EXEMPLAIRE */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD TRAVEE
       DESCRIPTION IS /* TRAVEE DE L'EXEMPLAIRE */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD RAYON
       DESCRIPTION IS /* RAYON OU SE TROUVE L'EXEMPLAIRE */
       DATATYPE IS SIGNED WORD.

DEFINE FIELD MOTCLE
       DESCRIPTION IS /* MOT CLE DE L'OUVRAGE */
       DATATYPE IS TEXT SIZE IS 30.

COMMIT

!*****************************************************************
! Define relation
!*****************************************************************

DEFINE RELATION OUVRAGE.
       TITRE.
       ANNEE.
       NUMOU.
       EDITEUR.
END OUVRAGE RELATION.

DEFINE RELATION OUVAUT.
       NUMOU.
       NUMAU.
END OUVAUT RELATION.

DEFINE RELATION MCOUV.
       NUMOU.
       MOTCLE.
END MCOUV RELATION.

DEFINE RELATION EXEMPLAIRE.
       NUMOU.
       NUMEX.
       ETAGE.
       TRAVEE.
       RAYON.
END EXEMPLAIRE RELATION.

DEFINE RELATION EMPRUNT.
       NUMEX.
       NUMEM.
       DATE-DEBUT BASED ON DATEDEB.
END EMPRUNT RELATION.

DEFINE RELATION EMPRUNTEUR.
       NUMEM.
       NOMEM.
```

```
        ADRESSE.
END EMPRUNTEUR RELATION.

DEFINE RELATION AUTEUR.
        NUMAU.
        NOMAU.
        ORIGINE.
END AUTEUR RELATION.

DEFINE RELATION EMPRUNT-ARCH.
        NUMEX.
        NUMEM.
        DATE-DEBUT BASED ON DATEDEB.
        DATE-FIN BASED ON DATEDEB.
END EMPRUNT-ARCH.

COMMIT

!*****************************************************************
!  Define constraints
!*****************************************************************

DEFINE CONSTRAINT NUM-OU-EXIST
        FOR OA IN OUVAUT
        REQUIRE ANY OU IN OUVRAGE
              WITH OU.NUMOU = OA.NUMOU.

DEFINE CONSTRAINT NUM-AU-EXIST
        FOR OA IN OUVAUT
        REQUIRE ANY A IN AUTEUR
              WITH A.NUMAU = OA.NUMAU.

DEFINE CONSTRAINT NUM-OU-MC-EXIST
        FOR MC IN MCOUV
        REQUIRE ANY OU IN OUVRAGE
              WITH OU.NUMOU = MC.NUMOU.

DEFINE CONSTRAINT NUM-OU-EX-EXIST
        FOR E IN EXEMPLAIRE
        REQUIRE ANY OU IN OUVRAGE
              WITH OU.NUMOU = E.NUMOU.

DEFINE CONSTRAINT NUM-EX-EXIST
        FOR EM IN EMPRUNT
        REQUIRE ANY EX IN EXEMPLAIRE
              WITH EX.NUMEX = EM.NUMEX.

DEFINE CONSTRAINT NUM-EMP-EXIST
        FOR EM IN EMPRUNT
        REQUIRE ANY EMP IN EMPRUNTEUR
              WITH EMP.NUMEM = EM.NUMEM.

DEFINE CONSTRAINT NUM-EX-EM-EXIST
        FOR EA IN EMPRUNT-ARCH
        REQUIRE ANY EX IN EXEMPLAIRE
              WITH EX.NUMEX = EA.NUMEX.

DEFINE CONSTRAINT NUM-EM-EA-EXIST
        FOR EA IN EMPRUNT-ARCH
        REQUIRE ANY EMP IN EMPRUNTEUR
```

```
            WITH EMP.NUMEM = EA.NUMEM.

COMMIT

!*****************************************************
! Define index
!*****************************************************

DEFINE INDEX AUTEUR-NUM-AU FOR AUTEUR
       DUPLICATES ARE NOT ALLOWED.
       NUMAU.
END AUTEUR-NUM-AU INDEX.

DEFINE INDEX OUVAUT-NUM-AU FOR OUVAUT
       DUPLICATES ARE ALLOWED.
       NUMAU.
END OUVAUT-NUM-AU INDEX.

!-------------------------------------------------------
! CONTRAINTE 9 (cfr. SCHEMA-4 DU TEXTE DU MEMOIRE)
!-------------------------------------------------------
DEFINE INDEX OUVAUT-IDENTIF FOR OUVAUT
       DUPLICATES ARE NOT ALLOWED.
       NUMOU.
       NUMAU.
END OUVAUT-IDENTIF INDEX.

DEFINE INDEX OUV-NUM-OU FOR OUVRAGE
       DUPLICATES ARE NOT ALLOWED.
       NUMOU.
END OUV-NUM-OU INDEX.

DEFINE INDEX MCOUV-NUM-OU FOR MCOUV
       DUPLICATES ARE ALLOWED.
       MOTCLE.
END MCOUV-NUM-OU INDEX.

!-------------------------------------------------------
! CONTRAINTE 10  (cfr. SCHEMA-4 DU TEXTE DU MEOIRE)
!-------------------------------------------------------
DEFINE INDEX MCOUV-IDENTIF FOR MCOUV
       DUPLICATES ARE NOT ALLOWED.
       NUMOU.
       MOTCLE.
END MCOUV-IDENTIF INDEX.

DEFINE INDEX EXEMP-NUM-EX FOR EXEMPLAIRE
       DUPLICATES ARE NOT ALLOWED.
       NUMEX.
END EXEMP-NUM-EX INDEX.

DEFINE INDEX EXEMP-NUM-OU FOR EXEMPLAIRE
       DUPLICATES ARE ALLOWED.
       NUMOU.
END EXEMP-NUM-OU INDEX.

DEFINE INDEX EMPR-NUM-EM FOR EMPRUNTEUR
       DUPLICATES ARE NOT ALLOWED.
       NUMEM.
END EMPR-NUM-EM INDEX.
```

```
DEFINE INDEX EMPRUNT-NUM-EX FOR EMPRUNT
       DUPLICATES ARE NOT ALLOWED.
       NUMEX.
END EMPRUNT-NUM-EX INDEX.

DEFINE INDEX EMPRUNT-NUM-EM FOR EMPRUNT
       DUPLICATES ARE ALLOWED.
       NUMEM.
END EMPRUNT-NUM-EM INDEX.

DEFINE INDEX EMPRUNT-ARCH-NUMEX FOR EMPRUNT-ARCH
       DUPLICATES ARE ALLOWED.
       NUMEX.
END EMPRUNT-ARCH-NUMEX INDEX.

DEFINE INDEX EMPRUNT-ARCH-NUMEM FOR EMPRUNT-ARCH
       DUPLICATES ARE ALLOWED.
       NUMEM.
END EMPRUNT-ARCH-NUMEM INDEX.

COMMIT
FINISH
```

ANNEXE-2

CHARGEMENT DES BASES DE DONNEES

ANNEXE-2

CHARGEMENT DES BASES DE DONNEES

A) PRINCIPES

Dans cette annexe, nous allons décrire la manière dont nous avons réalisé la génération des données. Nous présenterons d'abord les principes et les objectifs du chargement des bases de données. Ensuite, nous décrirons de manière générale l'architecture du programme de chargement et les descriptions et les algorithmes des modules du programme. Enfin, nous donnerons un exemple de détermination des paramètres physiques des bases de données.

1) Objectifs

Après avoir terminé la conception de la BD et écrit un schéma logique représentant la structure du système d'information, il faut y charger les données. Nous avons fixé les quatre objectifs suivants pour accomplir la tâche de chargement des bases de données.

- Le premier objectif est de constituer des BD de volume réaliste. Les volumes des BD dépendent de la quantification des données prédéfinie dans la section 3.4 du texte du mémoire.

- Le deuxième objectif est de donner une distribution réaliste des données dans les BD. Nous déterminerons dans ce but les tailles des populations des types d'entités ainsi que les distributions du nombre d'associations par entité (voir plui loin).

- Le troisième objectif est de charger les mêmes données dans les bases de données DBMS et RDB. A partir des mêmes quantifications et répartitions des données, les programmes de chargement qui sont traduits d'un même algorithme vont garantir la réalisation de cet objectif.

- Le quatrième objectif est de charger les BD avec un coût minimum. Il est impossible de créer une BD assez volumineuse manuellement. Nous avons donc à des programmes de chargement qui génèrent automatiquement les données.

Dans le paragraphe suivant, nous décrirons les principes de la génération automatique des données.

2) Principes de la génération des données

- Valeurs d'item

Nous envisagons deux types de valeurs d'item dans notre expérience: le type numérique et le type de chaîne de caractères.

Il est facile de générer les données numériques. Nous les générons normalement avec un compteur ou une liste de valeurs numériques. Par exemple, pour l'item ANNEE d'OUVRAGE, nous n'avons défini que 40 années différentes. Nous avons donc defini une liste LISAN dans le "WORKING-STORAGE SECTION", qui contient ces 40 valeurs. Chaque fois que le programme charge un record d'OUVRAGE, il suffit de tirer une année au hasard dans cette liste.

Afin de générer une chaîne de caractères, le programme dispose d'une procédure qui a pour fonction de fournir 5 caractères choisi parmi les 26. Chaque fois que le programme appelle cette procédure, il obtient une chaine de caractères de longueur 5. Par exemple, pour générer un nom d'AUTEUR, le programme ajoute un caractère "N" au début de la chaîne obtenue.

- les identifiants

Nous remarquons que dans le système BIBLIOTHEQUE, les valeurs des identifiants sont des types numériques. Il convient donc de les générer avec un compteur. Avant de charger un type d'article, celui-ci est remis à zéro. Chaque fois que le programme charge un record dans ce type d'article, ce compteur est augmenté d'une unité, le contenu du compteur est ensuite assigné à l'identifiant de l'article. Il est évident qu'il n'y a pas de valeurs dupliquées pour cet identifiant.

- Les associations

La création des associations s'effectue selon la procedure suivante:

(1) Etant donné un record OUVRAGE par exemple, on détermine le nombre d'associations que l'on va créer (le nombre d'AUTEUR qu'on va lui associer par exemple). On utilise leur tableau de répartition tel que celui de la figure-A.1. On y indique par exemple que 30% des OUVRAGE qui sont ecrits par deux AUTEUR.

aut/ouv	% d'ouv	nbr/ouv
0	5	50
1	40	400
2	30	300
3	10	100
4	10	100
5	5	50

Figure-A.1 Répartition d'OUVRAGE-AUTEUR

(2) On sélectionne aléatoirement le nombre d'AUTEUR ainsi désigné, par tirage aléatoire de leur valeur d'identifiant. On vérifie que ces auteurs sont distincts.

(3) On crée les associations.

- Ordre de chargement

Lors du chargement des données, chaque type d'article peut être considéré comme une unité de chargement.

Lorsque l'on prédéfinit les contraintes d'existence sur plusieurs articles, on doit charger ces articles en considérant l'ordre de chargement. Par exemple, dans le schéma-3 du texte, on a la contrainte: NUMEM (: EMPRUN-ARCH) IN NUMEM (: EMPRUNTEUR). Il est clair que l'on doit charger les records d'EMPRUNTEUR avant de charger les records d'EMPRUNT-ARCH.

En effet, dans les schémas des BD, quelques types d'article sont à l'origine des types d'association du schéma conceptuel. Nous les appelons les articles de type "association". Chaque article de ce type est originellement lié à certains articles de type "entité". Selon la nature d'existence d'une association, la règle du chargement sera:

de charger d'abord les articles de type "entité" et ensuite les articles de type "association".

Par exemple, OUVAUT dans le schéma-3 et schéma-4 ne peut être chargé avant que OUVRAGE et AUTEUR soient chargés.

B) LE PROGRAMME DE CHARGEMENT

Le programme de chargement est une réalisation des objectifs décrits dans le paragraphe précédent.

Nous avons deux programmes de chargement, l'un pour la BD/DBMS et l'autre pour la BD/RDB. En ce qui concerne l'architecture et l'algorithme, ces programmes sont les mêmes.

1) Générateur de nombre aléatoire

Nous avons choisi le générateur de nombre aléatoire du système en utilisant l'instruction COBOL: CALL "MTH$RANDOM" USING ....

Ce générateur donne les nombres aléatoires répartis uniformément entre 0 et 1.

2) Architecture des programmes

Ce programme consiste en cinq modules. Nous en donnons le diagramme (cfr. FIG B-1) ci-après. Chaque rectangle correspond à un module du programme.

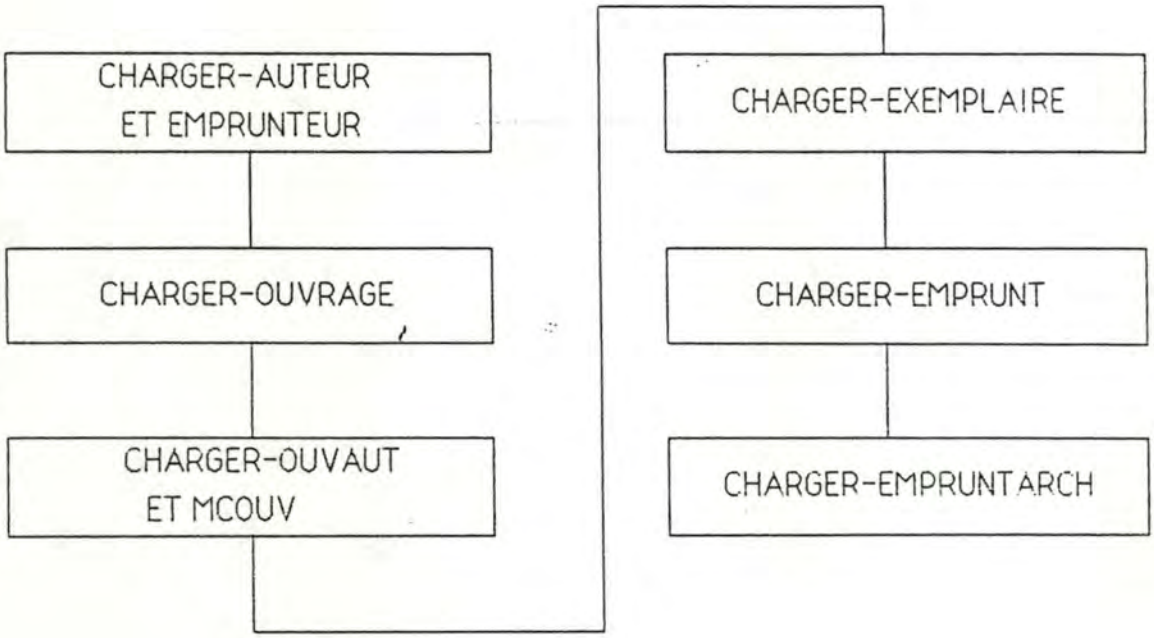


FIG B-1 Les modules du programme de chargement

3) Decription et algorithme des modules

3.1) CHARGER-AUTEUR ET EMPRUNTEUR

Ce module a pour but de charger 2 types d'article : AUTEUR et EMPRUNTEUR. On les charge d'abord dans ce module parce qu'ils sont des articles de type "entité".

ALGORITHME :

```
For compt := 0 to 500 do     (* 500 auteurs prédéfinis *)
    call procedure permutation( car-perm) ;
    if compt < 100 then       (* 100 emprunteurs prédéfinis *)
        create E := EMPRUNTEUR ( : NUMEM = compt ,
                                   NOMEM = "E" + car-perm ,
                                   ADRESSE = "A" + car-perm );
        endif ;
    create A := AUTEUR (: NUMAU = compt,
                          NOMAU = "N" + car-perm ,
                          ORIGINE = "O" +car-perm );
endFor ;
```

3.2) CHARGER-OUVRAGE :

Ce module a pour but de charger l'article OUVRAGE. On a une liste qui contient 40 années.

ALGORITHME:

```
For compt := 0 to 1000 do  (* 1000 ouvrages prédéfinis *)
    generate at random a number ANNEE1 from "liste d' annees";
    car-perm := a name of AUTEUR generated at random;
    create O := OUVRAGE (: NUMOU = compt ,
                           TITRE = "T" + car-perm,
                           ANNEE = ANNEE1,
                           EDITEUR = "E" + car-perm );
endFor ;
```

3.3) CHARGER-OUVAUT ET MCOUV:

Ce module établit l'association "many-to-many" entre AUTEUR et OUVRAGE. On sait que, pour la base de données DBMS, il faut créer un type de record supplémentaire. Ici il s'appelle OUVAUT.

Il est un peu compliqué de charger OUVAUT en DBMS. Il faut tenir compte des concepts de l'occurence d'un set et des courants d'un type de record. Selon la répartition du nombre d'AUTEUR par OUVRAGE, chaque fois qu'on charge un record d'OUVRAGE, on doit mettre au point des courants d'OUVRAGE et d'AUTEUR (cfr. le programme de chargement à l'annexe-4).

Pour la base de données relationnelle, grâce à l'uniformité des représentations logiques, la façon de charger est toujours la même.

Pour faciliter la tâche, on suppose que la répartition entre MOTCLE et OUVRAGE soit la même que celle entre AUTEUR et OUVRAGE. C'est-à-dire qu'il y a aussi de 0 à 5 mots-clés par ouvrage.

ALGORITHME :

```
numero-de-base := 50  (* suppose les ouvrages de numéro 0 à 50
                         sans auteur *)
For i := 1 to 5 do
   j = rep-ou-au ( i )  (* rep-ou-au est une table qui
                           correspond à la TABLE 6-1 *)
     for k := 1 à j do
         numéro-de-base := numéro-de-base + 1
         for h :=1 to i do
            generate at random a numbre NUMAU1 ;
            ganerate at random a string CAR-PERM from AUTEUR ;
            create OU := OUVAUT (( OO : OUVRAGE (: NUMOU =
                                    numéro-de-base )) and
                                    ( AO : AUTEUR (: NUMAU =
                                           NUMAU1 ))) ;
            create M := MCOUV (( OMC : OUVRAGE ( : NUMOU =
                                 numéro-de-base )) and
                                 ( : MOTCLE ( = "M" + CAR-PERM ))) ;
         endfor ;
   endfor ;
endFor .
```

3.4) CHARGER-EXEMPLAIRE:

On charge les données à EXEMPLAIRE dans ce module.

On dispose d'une liste de localisation de l'EXEMPLAIRE. On dispose également d'un tableau de répartition Per-ou-ex entre OUVRAGE et EXEMPLAIRE.

```
ALGORITHME:
compt := 0 ;
for i :=1 to 10 do
    j = Rep-ou-ex ( i );
    for k :=1 to j do
        generate at random a number NUMOU1 ;
        generate at random a number LOCA from liste of local ;
        for h :=1 to i do
            create EX := EXEMPLAIRE ((( OE : OUVRAGE ( : NUMOU
                                        = NUMOU1)) and
                                        ( : NUMEX = compt) and
                                        (: LOCALISATION= LOCA ));
            compt = compt + 1;
        endfor;
endfor;
```

3.5) CHARGER-EMPRUNT :

On charge 80 records pour EMPRUNT dans ce module. On dispose d'une liste de dates. Ici il faut surtout considérer qu'un exemplaire ne peut être prêté qu'une seule fois avant qu'il soit restitué. Lors du chargement, on tire au hasard des numéros d'exemplaire en vérifiant que chaque numéro n'est tiré qu'une seule fois.

ALGORITHME :

```
for i := 0 à 80 do
    generate at random two numbers NUMEX1, NUMEM1 ;
    generate at random a date DATE1 from table of dates ;
    for EX := EXEMPLAIRE ( : NUMEX = NUMEX ) do
        if NUMEX not in NUMEX(:EMPRUNT) then
            create EM := EMPRUNT(( : DATE-DEBUT = DATE1 ) and
                                (EXE: EX) and (EMP : EMPRUNTEUR
                                ( : NUMEM = NUMEM1 )) ;
        endif ;
    endfor ;
endfor ;
```

3.6) CHARGER-EMPRUNTARCH :

On charge 280 records dans EMPRUNTARCH. Pour faciliter la tâche, on suppose que DATE-FIN est toujours égale à DATE-DEBUT + 14. On dispose d'une liste de dates LISDATE.

```
ALGORITHME:
 for compt := 1 to 280 do
     ganerate at random a number NUMEX1 ;
     if NUMEX1 not in NUMEX(: EMPRUNT) then
        generate at random a number NUMEM1 ;
        generate at random a date DATE1 from LISDATE ;
        create EA := EMPRUNTARCH ( (: DATE-DEBUT = DATE1) and
                                   (: DATE-FIN = DATE + 14) and
                    ( EXEA : EXEMPLAIRE (:NUMEX = NUMEX1)) and
                    ( EMPA : EMPRUNTEUR ( :NUMEM = NUMEM ))) ;
     endif ;
 endfor ;
```

C) DETERMINATION DES PARAMETRES PHYSIQUES DES BD

___ Exemple pour la BD de taille réduite

Cette phase a pour but de créer une base de données adéquate. En général, il vaut mieux qu'un taux de remplissage de pages de stockage soit assez élevé, qu'il ait le moins de fragmentation possible, et que la taille de pool des tampons ne soit ni trop grande, ni trop petite.

Pour déterminer les meilleurs valeurs de ces paramètres, on doit calculer la taille de records que l'on va stocker dans la base de données.

On se réfère aux tableaux suivants :

DATA TYPE	BYTES
signed word	2
signed longword	4
signed quadword	8
f_floating	4
g_floating	8
date	8
text	n
varying text	

TAB C-1 SIZE OF RDB/VMS DATA TYPE

DATA TYPE	BYTES
signed byte	1
word	2
long word	4
quadword	8
octaword	16
f_floating	4
d_floating	8
g_floating	8
character	n
unsigned numeric	n

TABL C-2 SIZE OF DBMS/VAX DATA TYPE

Ici nous faisons attention au plus grand type de record dans la base de données. On peut facilement conclure que le type de record OUVRAGE est le plus grand dans la BD/DBMS ainsi que dans la BD/RDB.

1) EVALUATION DU VOLUMME DE LA BD/DBMS

Nous prenons la valeur de défaut pour la longueur de page : 2 blocs par page = 1024 bytes.

La longueur de données gestion est différent pour les articles de type différents (cfr. 1.1).

Nous donnons la liste suivante (unité des chiffres est en bytes).

EREA	ARTICLE	LLA	LPA	NOCA	NAPP	NP	NPAR	NPAR/t
FAUTEUR	AUTEUR	62	81	500	12	42	100	120
	OUVAUT	0	23	1950	44	45		
FOUV	OUVRAGE	66	97	1000	10	100	316	380
	MCOUV	30	44	1950	23	85		
FEXMP	EXEMPLAIRE	8	33	2100	31	68	88	105
FEMPR	EMPRUNTEUR	62	87	100	11	10	31	38
	EMPRUNT	4	29	80	35	3		
	EMPRUNT-ARCH	8	33	280	31	10		
					TOTAL		535	643

LISTE C-1 Volume de la BD/DBMS BIBLIO

D' où :

LLA : longueur logique d'un article ;

LPA : longueur physique d' un article (LLA + données gestions);

NOCA : nombre prédéfini d' occurrences d'un article ;

NAPP : nombre maximum d' articles par page ;

NP : nombre de pages pour stocker tous les records d' un article ;

NPAR : nombre de pages d'une AREA (cfr. fin du chapitre) ;

NPAR/t : nombre de pages d'une AREA à taux de remplissage 80% .

1.1) Exemple de calcul du volume de la BD/DBMS

- CALCUL DE LA LONGUEUR DES DONNEES DE GESTION

Avant de calculer, le lecteur doit se référer au chapitre 8 "Internal Representation of the Database" de [DBMS 84].

Pour calculer la longueur de données de gestion il faut tenir compte de la structure de records de stockage (voir figure-C.1).

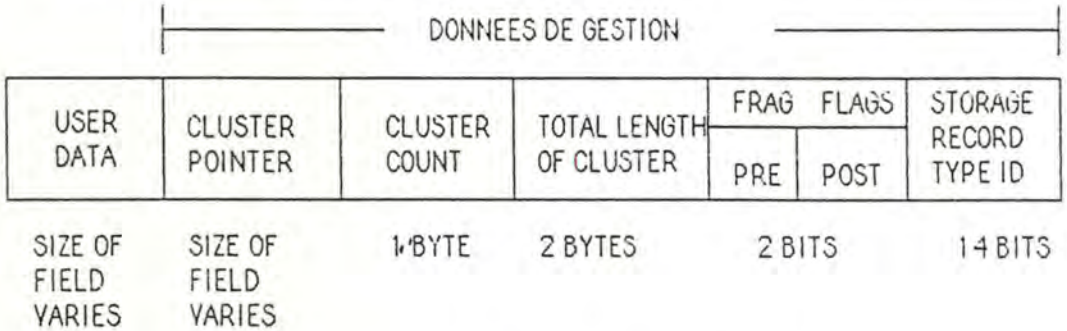


FIGURE-C.1 STORAGE RECORD STRUCTURE

Le format général des pointeurs (CLUSTER POINTER de la figure-C.1) est présenté dans la figure-C.2.

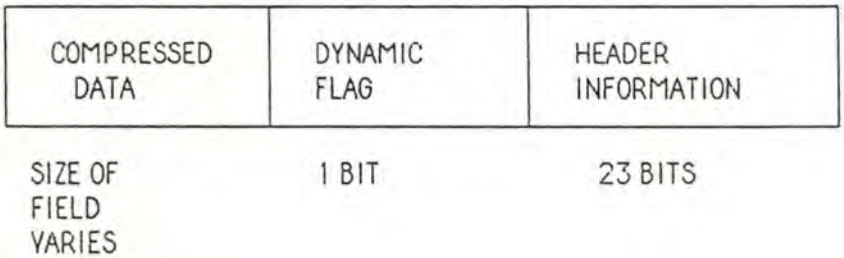


FIGURE-C.2 GENERAL FORMAT OF CLUSTER

Au vu de la figure-C.1, les informations utilisées par le SGBD sont en longueur de 5 bytes fixés. Au vu de la figure-C.2, les informations sur les caractèristiques des pointeurs sont en longueur de 3 bytes. En général, le SGBD a besoin de 2 bytes pour représenter un pointeur. Nous notons qu'il y a toujours 3 pointeurs NEXT, OWNER et PRIOR pour un MEMBER record du set. Si le record est dans un set de mode CALC, il faut compter 2 bytes de plus pour le pointeur calculé . En ce qui concerne l'OWNER d'un set, il garde toujours le pointeur FIRST.

On calcule, par exemple la longueur des données de gestion pour l'AUTEUR. On voit dans le schéma 2, que l'AUTEUR est un membre record du SYSTEM SET et un OWNER du AO SET. Or, la longueur de données de gestion

5 (header) + 3 (carac. de pointeur) + (2 * 3 (pointeurs)) +
+ 3 (carac.de pointeur) + 2 (pointeur FIRST) =
= 19 bytes .

Remarque :

Le calcul ci-dessus se fait dans des conditions où les records ne sont pas fragmentés

- CALCUL DE NOMBRES DE PAGES D'UNE AREA (NPAR):

Soit l'AREA FAUTEUR qui ne contient qu'AUTEUR.

Pour calculer les pages totales d'AREA, il faut tenir compte de la description des pages de la base de données (voir FIGURE-C.3).

PAGE HEADER
LINE INDEX
LOCKED FREE SPACE
FREE SPACE
STORAGE SEGMENTS

FIGURE-C.3 La description des pages de la base de données

PAGE HEADER: l'information pour le SGBD (22 bytes par page). Or, on a besoin de 924 bytes (1 page) pour les 42 pages de données AUTEUR.

LINE INDEX : c' est un directory pour les records de stockage dans la page. Le format est présenté dans la figure-C.4. Il dépend de NAPP (cfr. LISTE C-1) . On reprend AUTEUR, par exemple. Une page peut contenir 12 records d'AUTEUR, alors la longueur de LINE INDEX est: 4 bytes * 12 + 2 bytes = 50 bytes.

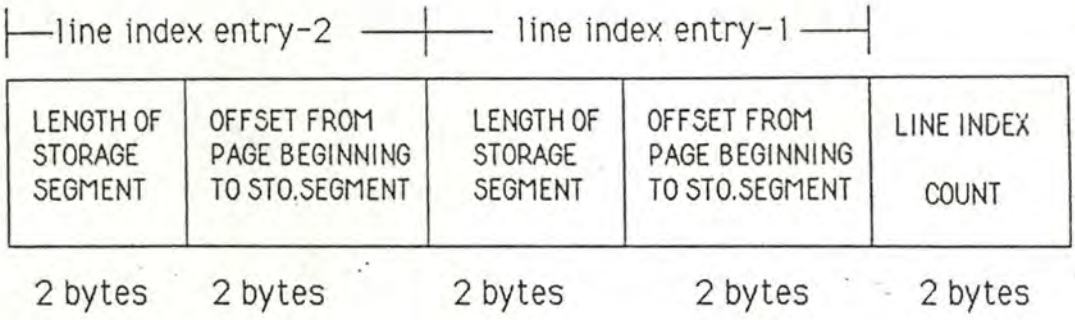


FIGURE-C.4 LINE INDEX FORMAT

Lorsqu' il faut 42 pages pour contenir tous les records d'AUTEUR, on a besoin de 42 * 50 = 2100 bytes (≈2 pages) supplémentaires pour l'AREA.

Lorsqu'AUTEUR se situe dans le SYSTAUTEUR (set) de mode INDEX, il faut aussi compter les places qui sont occupées par les INDEX NODE. On sait :

- NOCA = 500. On definit la taille de noeud d'index avec la formule qui est donnée dans [DBMS 84]: 16 + 10 * √ 500 = 240 bytes. Ensuite, on définit SIZE OF NODE dans le STORAGE_SCHEMA avec cette valeur;

- la clé NUMAU = 2 BYTES . Or, chaque noeud peut avoir le nombre d'entrées : (240 - 16) / 2 = 112;

- 500 / 112 = 4. C'est le nombre de noeuds dont on a besoin pour le set SYSTAUTEUR.

- Enfin les places totales pour les index sont 4 * 240 = 960 bytes. Une page est donc suffissante.

SYSTEM RECORD : D'après la représentation interne de la page, il y a un système record pour chaque page. La longueur d'un système record est de 5 bytes. Selon les calculs ci-dessus, on sait qu'on a besoin de 42 pages pour les données AUTEUR , de 1 page pour les PAGE HEADER, de 2 pages pour LINE INDEX et de1 page pour index, ce qui revient à 46 pages. Or, 46 * 5 = 230 bytes (≈1 page) supplémentaires sont ajoutés pour l'AREA FAUTEUR.

Enfin, 47 pages sont suffisantes pour l'AREA FAUTEUR.

2) PARAMETRES DE LA CREATION DE LA BD/DBMS:

- Taille de page de stockage :

La valeur de défaut = 2 blocks par page.

Nous acceptons cette valeur, parce qu'il y a environ 819 bytes par page susceptibles de stocker les données en considérant le taux de remplissage de 80%. Dans notre cas, le record le plus long est l'OUVRAGE (97 bytes).

Alors, une page peut contenir 8 records au moins .
- Allocation de page pour les AREAS:

La valeur de défaut est = 100 pages par AREA.

D'après liste C-1, la plupart des AREAs ont besoin d'un nombre de pages oscillant entre 100 et 200. OUVRAGE en a besoin de plus. On peut lui donner une EXTENSION en définissant:

ALLOCATION =200/EXTENSION =100 .

- Taille des tampons :

Le principe de calcul de la taille est de conserver en mémoire les pages qui risquent d' être réutilisées dans un proche avenir.

Pour chaque RUN UNIT, le DBCS (Database Control system) gère un pool indivuduel de tampons. Ces tampons sont utilisés pour les transferts de données entre le RUN UNIT et les fichiers. Lors d'un COMMIT, DBMS "nettoie" les tampons du pool, écrit les changements dans le journal.

Il est possible de paramètrer:

- LENGTH_BUFFER (par défaut 5 * la taille de la plus grande page);

- BUFFERS (le nombre des tampons, par défaut 10). Cela permet de copier beaucoup de pages avant d'être forcé de devoir nettoyer le pool en raison d'un nouvel arrivage.

La taille du pool de tampons est importante, parce que:

- si elle est trop grande, le système de gestion de la mémoire virtuelle entre en action et deux E/S sont nécessaires pour lire une page de la BD dans la mémoire centrale.

- si elle est trop petite, le nombre d'opérations E/S croît.

Si l' on accepte les valeurs de défaut, alors le pool de tampons peut contenir
10 * 5 * 2 = 100 BLOCS = 50 pages.

D'après la liste c-1 pour déterminer la taille du pool, nous allons faire certains compromis. Nous avons défini :

BUFFERS=18/LENGTH_BUFFER=28

Cela donne 252 pages pour le pool.

3) EVALUATION DU VOLUME DE LA BD/RDB

On prend la valeur de défaut pour la page de stockage :
2 block/page = 1024 bytes/page.

Soit, la longueur de données gestions par article = 10 bytes . On donne la liste suivante :

ARTICLE	LLA	LPA	NOCA	NAPP	NP	NP/t
AUTEUR	62	72	500	14	36	44
OUVAUT	4	14	1950	73	27	33
OUVRAGE	66	76	1000	13	72	87
MCOUV	32	42	1950	25	78	94
EXEMPLAIRE	10	20	2100	51	42	51
EMPRUNTEUR	62	72	100	14	8	10
EMPRUNT	8	18	80	56	2	3
EMPRUNT-ARCH	12	22	280	47	5	6
				TOTAL	270	325

LISTE C-2 Volume de la BD/RDB BIBLIO

REMARQUE :

La liste C-2 n'est pas comparable avec la liste C-1 du point de vue des véritables volumes de stockage, puisque l'on n'a pas évalué les volumes d'index et les données de gestion en RDB. Nous n'avons pas d'informations suffisantes pour calculer la longueur de données de gestion. Nous donnons la liste sous cette forme afin de fixer une idee de détermination des paramètres de création de la BD.

3.1) Les paramètres:

Les valeurs de défaut :

- number-page/database : 400 pages (extension automatique);
- page_blocs : 2 (1024 bytes) ;
- number_buffers : 20 ;
- buffer_blocks : 3 * page_blocks .

Pour des raisons comparables à celles de DBMS, nous avons

NUMBER OF BUFFERS IS 18 ;
SIZE OF BUFFER IS 28.
ALLOCATION : la valeur de défaut .

ANNEXE-3

ALGORITHMES CONFORMES AUX SGBD

ANNEXE-3

ALGORITHMES CONFORMES AUX SGBD

Nous donnons ici, tous les algorithmes qui sont conformes aux SGBD. Le lecteur doit faire référence au point 3.3.4 du texte du mémoire afin de savoir la manière de transformer l'algorithme prédicatif en algorithmes conformes.

A) CONSULTATION DE LA BASE DE DONNEES

1) APPLICATION1

Elle n'est pas nécessaire d' être transformée, Car c'est une requête simple avec la condition évaluable .

2) APPLICATION2

Pas nécessaire.

3) APPLICATION3

3.1 Conforme à DBMS :

```
input ( 1970 ) ;
for O := OUVRAGE do
    if ANNEE ( : O ) = 1970 then
       print O ;
    endif ;
endfor ;
```

3.2 Conforme à RDB :

On reprend l'algorithme prédicatif puisqu'en RDB il offre la primitive d'accéder à la base de données à condition filtrée .

4) APPLICATION4

4.1 Conforme à DBMS:

```
for O := 1 OUVRAGE do
    if OE not empty then
       print O ;
    endif ;
endfor ;
```

4.2 Conforme à RDB (optimisation implicite) :

```
for O := 1 OUVRAGE ( : NUMOU = NUMOU ( : EXEMPLAIRE ) ) do
    print O;
endfor ;
```

4.2.1 Conforme à RDB (optimisation explicite) :

```
for O := OUVRAGE do
    for EX := EXEMPLAIRE ( : NUMOU = NUMOU ( : O ) ) do
        print O ;
        exit EX ;
    endfor;
endfor ;
```

5) APPLICATION5

5.1 Conforme à DBMS :

```
for O := 1 OUVRAGE do
    if OO not empty then
        print O ;
    endif ;
endfor ;
```

5.2 Conforme à RDB (optimisation implicite):

```
for O :=  1 OUVRAGE ( : NUMOU = NUMOU ( : OUVAUT ) ) do
    print O ;
endfor ;
```

5.2.1 Conforme à RDB (optimisation explicite):

```
for O :=OUVRAGE do
    for OU := OUVAUT (:NUMOU = NUMOU (:O) do
        print O;
        exit OU;
    endfor;
endfor;
```

6) APPLICATION6

6.1 Conforme à DBMS:

```
input ( XNUMAU ) ;
for A := AUTEUR ( : NUMAU = XNUMAU ) do
    for OU := OUVAUT ( OA : A ) do
        for O := OUVRAGE ( OO : OU ) do
            print O ;
        endfor ;
    endfor ;
endfor ;
```

6.2 Conforme à RDB (optimisation implicite) :

```
input ( XNUMAU ) ;
for O := OUVRAGE ( ; NUMOU = NUMOU ( : OUVAUT ( : NUMAU =
                                        NUMAU(: XNUMAU )))) do
    print O ;
endfor ;
```

6.21 Conforme à RDB (optimisation explicite) :

```
input ( XNUMAU ) ;
for OU := OUVAUT ( : NUMAU = XNUMAU ( : A ) ) do
   for O := OUVRAGE ( : NUMOU = NUMOU ( : OU ) ) do
      print O ;
   endfor ;
endfor ;
```

7) APPLICATION7

Idem que l'application6 sauf l'entrée : XNOMAU au lieu de XNUMAU .

8) APPLICATION8

8.1 Conforme à DBMS :

```
input ( XORIGINE ) ;
for A := AUTEUR do
    if ORIGINE( : A ) = XORIGINE then
       for OU := OUVAUT ( AO : A ) do
           for O := OUVRAGE ( OO : OU ) do
               print O ;
           endfor ;
       endfor ;
    endif ;
endfor ;
```

8.2 Confome à RDB (optimisation implicite) :

```
for O := OUVRAGE ( : NUMOU = NUMOU ( : OUVAUT ( : NUMAU
                    = NUMAU ( : AUTEUR ( : ORIGINE =
                                XORIGINE ) ) ) ) ) do
    print O ;
endfor ;
```

8.2.1 Conforme à RDB (optimisation explicite) :

```
input ( XORIGINE ) ;
for A := AUTEUR (:ORIGINE = XORIGINE ) do
    for OU := OUVAUT ( NUMAU = NUMAU ( : A ) ) do
        for O := OUVRAGE ( NUMOU = NUMOU ( : OU ) ) do
            print O ;
        endfor ;
    endfor ;
endfor;
```

9) APPLICATION9

9.1 Conforme à DBMS :

```
input ( XTITRE , XAN , XNOMAU ) ;
trouve := false ;
for A := AUTEUR ( : NOMAU = XNOMAU ) do
    for O := OUVRAGE ( OUVAUT : A ) do
        if ( TITRE( : O ) = XTITRE) and (ANNEE( : O ) = XAN )) then
          for EX := EXEMPLAIRE ( OE : O ) do
              EX-OK := true ;
              for EM := EMPRUNT ( EXE : EX ) do
                  EX-OK := false ;
              endfor ;
              if EX-OK then
                  trouve := true ;
                  print NUMEX ( :EX ) ;
                  exit A ;
                  endif ;
            if trouve then exit EX ; endif ;
            endfor ;
          endif ;
      if trouve then exit O ; endif ;
      endfor ;
  if trouve then exit A ; endif ;
  endfor ;
```

9.2 Conforme à RDB :

```
input ( XTITRE, XAN , XNOMAU ) ;
TROUVE := false ;
for O := OUVRAGE (( : ANNEE = XAN ) and ( : TITRE = XTITRE ) and
                  ( : NUMOU = NUMOU ( : OUVAUT ( :NUMAU
                      = NUMAU ( : AUTEUR ( : NOMAU
                                          = XNOMAU))))) do
    for EX := EXEMPLAIRE ( ( : NUMEX not in NUMEX ( : EMPRUNT ))
                  and ( : NUMOU = NUMOU ( : O ))) do
        TROUVE := true ;
        print NUMEX ( : EX ) ;
        exit EX ;
    endfor ;
    if TROUVE then exit O  endif ;
  endfor;
```

10) APPLICATION10 , 11, 12 , 13

10.1 Conforme à DBMS:

```
input ( XNUMOU );
for O := OUVRAGE( : NUMOU = XNUMOU ) do
    print O ;
    for M := MCOUV ( OMC: O ) do
        print MOTCLE ( : M) ;
    endfor ;
    for OU := OUVAUT ( OO: O ) do
        for A := AUTEUR ( AO: OU ) do
            print A ;
        endfor ;
    endfor ;
    for EX := EXEMPLAIRE ( OE :O ) do
        print EX
        for EM := EMPRUNT (EXE: EX ) do
            print EM ;
            for EP := EMPRUNTEUR( EMP: EM) do
                print EP ;
            endfor ;
        endfor ;
        for EA := EMPRUNTARCH ( EXEA: EX ) do
            print EA ;
        endfor ;
    endfor ;
endfor ;
endFor .
```

10.2 Conforme à RDB

```
input ( XNUMOU );
for O := OUVRAGE( : NUMOU = XNUMOU ) do
    print O ;
    for M := MCOUV ( : NUMOU = NUMOU ( : O ) do
        print MOTCLE ( : O ) ;
    endfor ;
    for OU := OUVAUT ( : NUMOU = NUMOU ( : O ) ) do
        for A := AUTEUR ( : NUMAU = NUMAU (: OU ) ) do
            print A ;
        endfor ;
    endfor ;
    for EX := EXEMPLAIRE ( : NUMOU = NUMOU ( :O ) ) do
        print EX
        for EM := EMPRUNT ( NUMEX = NUMEX ( : EX ) ) do
            print EM ;
            for EP := EMPRUNTEUR( :NUMEM = NUMEM (: EM)) do
                print EP ;
            endfor ;
        endfor ;
        for EA := EMPRUNTARCH ( : NUMEX = NUMEX (: EX ) ) do
            print EA ;
        endfor ;
    endfor ;
endfor ;
endFor .
```

B) MISE A JOUR DE LA BASE DE DONNEES

14) APPLICATION14 , 15

Il n' est pas nécessaire de les transformer .

16) APPLICATION16

16.1 Conforme à DBMS:

```
input (XNUMOU, XMOTCLE);
for O:= OUVRAGE (:NUMOU = XNUMOU) do
    for M := MCOUV(OMC: O) do
        modify M(:MOTCLE = XMOTCLE);
    endfor;
endfor;
```

16.2 Conforme à RDB:

```
input (XNUMOU, XMOTCLE);
for M := MCOUV (:NUMOU = XNUMOU) do
    modify M (:MOTCLE = XMOTCLE);
endfor;
```

17) APPLICATION17

Pas nécessaire.

18) APPLICATION18

18.1 Conforme à DBMS:

```
input ( XNUMOU ) ;
DEL-FIN := false;
for O := OUVRAGE ( : NUMOU = XNUMOU ) do
    for EX := EXEMPLAIRE ( OE : O ) do
        if EX not in EXE then
            delete EX ;
            exit EX ;
            DEL-FIN := true ;
        endif;
    endfor ;
    if DEL-FIN then exit O ; endif ;
endfor ;
endFor .
```

18.2 Conforme à RDB :

```
input ( XNUMOU ) ;
for EX := EXEMPLAIRE ( ( : NUMOU = XNUMOU ) do
    if EX (:NUMEX not in NUMEX ( : EMPRUNT ) ) then
    for EM := EMPRUNT-ARCH ( : NUMEX = NUMEX ( : EX ) ) do
        delete EM ;
    endfor ;
    delete EX ;
    exit EX ;
    endif;
endfor ;
```

19) APPLICATION19

19.1 Conforme à CODASYL :

```
input ( XOUVRAGE ) ;
OU-EXIST := false;
if NUMOU(: XOUVRAGE) in NUMOU then
   OU-EXIST := true
endif;
if not OU-EXIST then
   create O := OUVRAGE ((:NUMOU = NUMOU(:XOUVRAGE)) and
                        (:TITRE = TITRE(:XOUVRAGE)) and
                        (:ANNEE = ANNEE(:XOUVRAGE)) and
                        (:EDITEUR = EDITEUR(:XOUVRAGE)));
   for 1 to 3 do
        input ( LIAUT[i], LIEXE[i] ) ;
        AU-EXIST := false ;
            if NUMAU (:XAUTEUR ) in NUMAU ( :AUTEUR ) then
              AU-EXIST := true ;
            endif ;
         if not AU-EXIST then
            create A := AUTEUR ((:NUMAU = NUMAU(:XAUTEUR)) and
                                (:NOMAU = NOMAU(:XAUTEUR)) and
                                (:ORIGINE = ORIGINE(:XAUTEUR))) ;
            create OU := OUVAUT ( ( OO : O ) and ( AU : A ) ) ;
         endif ;
         EX-EXIST := false;
             if NUMEX (:XEXEMP) = NUMEX(: EXEMPALIRE) then
            EX-EXIST := true;
            endif;
         if not EX-EXIST then
            create EX := EXEMPLAIRE ((:NUMEX = NUMEX(:XEXEMP)) and
                                     (:LOCALISATION =
LOCALISATION(:XEXEMP)) and
                                     ( OE : O ) ) ;
         endif;
      endfor ;
endif;
```

19.2 Conforme à RDB :

```
input ( XOUVRAGE ) ;
OU-EXIST := false;
for O := OUVRAGE(:NUMOU = NUMOU(:XOUVRAGE) do
   OU-EXIST := true ;
   exit O ;
endfor;
if not OU-EXIST then
   create O := OUVRAGE ((:NUMOU = NUMOU(:XOUVRAGE)) and
                        (:TITRE = TITRE(:XOUVRAGE)) and
                        (:ANNEE = ANNEE(:XOUVRAGE)) and
                        (:EDITEUR = EDITEUR(:XOUVRAGE)));
   for 1 to 3 do
       input ( LIAUT[i], LIEXE[i] ) ;
       AU-EXIST := false ;
       for A := AUTEUR(:NUMAU = NUMAU (:XAUTEUR ) do
           AU-EXIST := true ;
           exit A ;
       endfor ;
       if not AU-EXIST then
         create A := AUTEUR ((:NUMAU = NUMAU(:XAUTEUR)) and
                             (:NOMAU = NOMAU(:XAUTEUR)) and
                             (:ORIGINE = ORIGINE(:XAUTEUR))) ;
         create OU := OUVAUT ((: NUMOU = NUMOU(: O)) and
                              (: NUMAU = NUMAU(: A )));
       endif ;
       EX-EXIST := false;
       for EX := EXEMPLAIRE(:NUMEX  = NUMEX(: XEXEMP ) do
           EX-EXIST := true;
           exit EX ;
       endfor;
       if not EX-EXIST then
          create EX := EXEMPLAIRE ((:NUMEX = NUMEX(:XEXEMP)) and
                                   (:LOCALISATION = LOCALISATION(:XEXEMP))
and
                                   ( OE : O ) ) ;
       endif;
   endfor ;
endif;
```

20) APPLICATION20

20.1 Conforme à DBMS :

```
input ( XOUVRAGE ) ;
for O := OUVRAGE (  = XOUVRAGE ) do
    delet-ok := true
    for EX := EXEMPLAIRE ( OE: O ) do
        if EXE not empty then
           delet-ok := false ;
           exit-o ;
        end-if;
    endfor;
```

```
        if delet-ok then
           for EX := EXEMPLAIRE ( OE : O ) do
              delete EX ;
           endfor ;
           for OU := OUVAUT ( OO : O ) ) do
              for A := AUTEUR ( : NUMAU ( AO : OU ) ) do
                 delete A ;
              endfor ;
           endfor ;
           delete O ;
        endif;
     endfor ;
```

20.2 Conforme à RDB :

```
input ( XOUVRAGE ) ;
delet-ok := true;
for EX:= EXEMPLAIRE ((: NUMOU = NUMOU (:XOUVRAGE)) and (:NUMEX in
                                                NUMEX (: EMPRUNT))) do
    delet-ok := false;
    exit-ex ;
endfor;
if delet-ok then
   for O := OUVRAGE ( = XOUVRAGE ) do
      for OU := OUVAUT ( : NUMOU = NUMOU ( : O ) ) do
         for A := AUTEUR ( :NUMAU =NUMAU ( : OU ) ) do
            delete A ;
         endfor ;
      endfor ;
      for EX := EXEMPLAIRE ( : NUMOU = NUMOU ( : O ) ) do
            for EA := EMPRUNTARCH ( : NUMEX = NUMEX (: EX)) do
              delete EA ;
            endfor ;
         delete EX ;
      endfor ;
      delete O ;
   endfor ;
endif;
```

<u>21) APPLICATION21</u>

21.1 Conforme à DBMS :

```
input ( XNUMEX ) ;
for EX := EXEMPLAIRE ( : NUMEX = XNUMEX ) do
    for E := EMPRUNT ( EXE : EX ) do
        for EM := EMPRINTEUR ( EMP : E ) do;
            create EA := EMPRUNTARCH ( (EXEA : EX) and (EMPA : EM )
                    and ( : DATE-DEBUT = DATE-DEBUT( : E ) )
                    and ( : DATE-FIN = DATE-DU-JOUR ) ) ;
        delete E ;
        endfor ;
    endfor ;
endfor .
```

21.2 Conforme à RDB :

```
input ( XNUMEX ) ;
for E := EMPRUNT ( : NUMEX = XNUMEX ) do
    create EA := EMPRUNTARCH( (NUMEX = NUMEX (: E )) and
                              ( NUMEM =NUMEM (: E)) and
                        ( DATE-DEBUT = DATE-DEBUT( : E ) ) and
                        ( : DATE-FIN = DATE-DU-JOUR ) ;
    delete E ;
endfor ;
```

22) APPLICATION22

22.1 Conforme à DBMS :

```
input ( LISAN( 1...N ) ) ;
for i := 1 to N do
    for O := OUVRAGE do
        if ANNEE ( : O ) = LISAN( i ) then
          input ( XEDITEUR , XAUTEUR ) ;
          modify O ( : EDITEUR = XEDITEUR ) ;
           create A := AUTEUR ((:NUMAU = NUMAU(:XAUTEUR)) and
                               (:NOMAU = NOMAU(:XAUTEUR)) and
                               (:ORIGINE = ORIGINE(:XAUTEUR)) and
                               (AO : OU )) ;
           create OU := OUVAUT ((OO: O) and (AO: A));
        endif ;
    endfor ;
endfor ;
```

22.2 Conforme à RDB :

```
input ( LISAN( i ) ) ;
for i := 1 to N do
    for O := OUVRAGE ( : ANNEE = LISAN( i ) ) do
        input ( XEDITEUR , XAUTEUR ) ;
        modify O ( : EDITEUR = XEDITEUR ) ;
        create A := AUTEUR ((:NUMAU = NUMAU(:XAUTEUR)) and
                            (:NOMAU = NOMAU(:XAUTEUR)) and
                            (:ORIGINE = ORIGINE(:XAUTEUR));
        create OU := OUVAUT ((:NUMOU (:O) and (:NUMAU (:A));
    endfor ;
endfor;
```

23) APPLICATION23

23.1 Conforme à DBMS :

```
input ( LISNUM( 1...N) ) ;
for i := 1 to N do
   for EM := EMPRUNTEUR (: NUMEM = LISNUM( i) ) do
        for E := EMPRUNT ( : NUMEM = NUMEM ( : EM ) ) do
            delete E ;
        endfor ;
        delete EM ;
   endfor ;
endfor ;
```

23.2 Conforme à RDB :

```
input ( LISNUM( 1...N) ) ;
for i := 1 to N do
   for EM := EMPRUNTEUR (: NUMEM = LISNUM( i) ) do
        for E := EMPRUNT ( : NUMEM = NUMEM ( : EM ) ) do
            delete E ;
        endfor ;
        for EA := EMPRUNTARCH ( : NUMEM = NUMEM (: EM ) ) do
            delete EA ;
        endfor ;
        delete EM ;
   endfor ;
endfor ;
```

24) APPLICATION24

Pas nécessaire. Cfr. APPLICATION10.

ANNEXE-4

PROGRAMMES DES APPLICATIONS

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*
*   A)  PROGRAMMES/DBMS/COB
*
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

*****************************************
* APPLICATION 1
* lister tous les numeros d'OUVRAGE.
*****************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI1.

ENVIRONMENT DIVISION.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.

WORKING-STORAGE SECTION.
01 FIN-OU PIC 9.
01 I PIC 9999 VALUE 0.

PROCEDURE DIVISION.
100-BEGIN.
         CALL "LIB$INIT_TIMER".
         PERFORM BIBLIO1 UNTIL I > 10.
         CALL "LIB$SHOW_TIMER".
         STOP RUN.

BIBLIO1.
         READY FOUV PROTECTED RETRIEVAL.
         FETCH FIRST OUVRAGE WITHIN SYSTOUV.
*         DISPLAY NUMOU OF OUVRAGE.
         MOVE  0 TO FIN-OU.
         PERFORM BOUCL THRU BOUCL-EXIT UNTIL FIN-OU = 1.
         COMMIT.
         COMPUTE I = I + 1.

BOUCL.
        FETCH NEXT OUVRAGE WITHIN SYSTOUV
              AT END MOVE 1 TO FIN-OU
                     GO TO BOUCL-EXIT.
*        DISPLAY NUMOU OF OUVRAGE.

BOUCL-EXIT.
        EXIT.
```

```
*****************************************************************
* APPLICATION 6 :
* Fournir le(s) ouvrage(s) qui est (sont) ecrit(s)
*  par un auteur donne ( etant donne le numero d'AUTEUR ).
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI6.

ENVIRONMENT DIVISION.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.

WORKING-STORAGE SECTION.
01 FIN-SET PIC 9.
01 I PIC 99 VALUE 0.

PROCEDURE DIVISION.
100-BEGIN.
         CALL "LIB$INIT_TIMER".
         PERFORM BOUCL UNTIL I > 50.
         CALL "LIB$SHOW_TIMER".
         STOP RUN.

BOUCL.
        READY FOUV FAUTEUR RETRIEVAL.
        MOVE 245 TO NUMAU OF AUTEUR.
        FIND FIRST AUTEUR WITHIN SYSTAUTEUR
             USING NUMAU OF AUTEUR.
        FIND FIRST OUVAUT WITHIN AO.
        PERFORM FET-OU.
        MOVE 0 TO FIN-SET.
        PERFORM NEXT-OUVAUT THRU NEXT-EXIT
                UNTIL FIN-SET = 1.
        COMMIT.
        COMPUTE I = I + 1.

NEXT-OUVAUT.
        FETCH NEXT OUVAUT WITHIN AO
           AT END MOVE 1 TO FIN-SET
                  GO TO NEXT-EXIT.
        PERFORM FET-OU.

NEXT-EXIT.
        EXIT.
FET-OU.
       FETCH OWNER WITHIN OO.
*       DISPLAY NUMOU, TITRE, ANNEE, EDITEUR.
```

```
*****************************************************************
* APPLICATION 3
* Fournir le(s) ouvrage(s) qui est(sont) ecrit(s)
* par un auteur donne ( etant donne l' origine d'AUTEUR ).
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI3.

ENVIRONMENT DIVISION.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.

WORKING-STORAGE SECTION.
01 FIN-SET PIC 9.
01 TROUVE  PIC 9.
01 I PIC 99 VALUE 0.

PROCEDURE DIVISION.
100-BEGIN.
         CALL "LIB$INIT_TIMER".
         PERFORM TROUVER-AU UNTIL I > 10.
         CALL "LIB$SHOW_TIMER".
         STOP RUN.

TROUVER-AU.
        READY FOUV FAUTEUR PROTECTED RETRIEVAL.
        FETCH FIRST AUTEUR WITHIN SYSTAUTEUR.
        IF ORIGINE OF AUTEUR = "DABCVY"
           PERFORM FET-OUVAUT.
        MOVE 0 TO TROUVE.
        PERFORM NEXT-AU THRU NEXTAU-EXIT UNTIL TROUVE = 1.
        COMMIT.
        COMPUTE I = I + 1.

NEXT-AU.
       FETCH NEXT AUTEUR WITHIN SYSTAUTEUR.
       IF ORIGINE OF AUTEUR = "DABCVY"
          MOVE 1 TO TROUVE
          PERFORM FET-OUVAUT
          GO TO NEXTAU-EXIT.

NEXTAU-EXIT.
        EXIT.

FET-OUVAUT.
        FIND FIRST OUVAUT WITHIN AO.
        PERFORM FET-OU.
        MOVE 0 TO FIN-SET.
        PERFORM NEXT-OUVAUT THRU NEXT-EXIT UNTIL FIN-SET = 1.

NEXT-OUVAUT.
        FETCH NEXT OUVAUT WITHIN AO
           AT END MOVE 1 TO FIN-SET
                  GO TO NEXT-EXIT.
        PERFORM FET-OU.
```

```
NEXT-EXIT.
        EXIT.
FET-OU.
       FETCH OWNER WITHIN OO.
*       DISPLAY NUMOU, TITRE, ANNEE, EDITEUR.
```

```
*****************************************************
* APPLICATION 9
* Etant donne le titre , l'annee d'edition et
* le nom d'un auteur d'un ouvrage , fournir le numero
* d' un exemplaire disponible.
*****************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI9.

ENVIRONMENT DIVISION.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.

WORKING-STORAGE-SECTION.
01 TROUVE PIC 9.
01 FIN-AU PIC 9.
01 FIN-AO PIC 9.
01 FIN-OE PIC 9.
01 EX-OK  PIC 9.

PROCEDURE DIVISION.
100-BEGIN.
         CALL "LIBSINIT_TIMER".
         READY PROTECTED RETRIEVAL.
         MOVE 0 TO TROUVE.
         PERFORM FIRST-AU THRU AU-EXIT.
         IF TROUVE = 0
            PERFORM NEXT-AU THRU AU-EXIT UNTIL
                    ( TROUVE = 1 ) OR ( FIN-AU = 1 ).
         COMMIT.
         CALL "LIBSSHOW_TIMER".
         STOP RUN.

FIRST-AU.
         FETCH FIRST AUTEUR WITHIN SYSTAUTEUR
         AT END GO TO AU-EXIT.
         IF NOMAU OF AUTEUR = "NABDHW"
            PERFORM CHER-OUAUT THRU OUAUT-EXIT.

NEXT-AU.
         FETCH NEXT AUTEUR WITHIN SYSTAUTEUR
               AT END MOVE 1 TO FIN-AU
                      GO TO AU-EXIT.
         IF NOMAU OF AUTEUR = "NABDHW"
            PERFORM CHER-OUAUT THRU OUAUT-EXIT.

AU-EXIT.
         EXIT.


CHER-OUAUT.
         IF AO EMPTY
            GO TO OUAUT-EXIT.
         FIND FIRST OUVAUT WITHIN AO.
         PERFORM FETCH-OWNER.
         IF TROUVE = 0
            MOVE 0 TO FIN-AO
```

```
            PERFORM NEXT-OUVAUT THRU AO-EXIT UNTIL FIN-AO = 1.

OUAUT-EXIT.
          EXIT.

FETCH-OWNER.
          FETCH OWNER WITHIN OO
          IF ANNEE OF OUVRAGE = "1955" AND
             TITRE OF OUVRAGE = "TASCTU"
             IF OE NOT EMPTY
                FETCH FIRST EXEMPLAIRE WITHIN OE
                PERFORM VERIF-EX-OK
                IF EX-OK = 1
                   MOVE 1 TO TROUVE
                   DISPLAY NUMEX OF EXEMPLAIRE
                ELSE
                   PERFORM NEXT-EXEMP THRU NEXT-EX-EXIT
                      UNTIL ( FIN-OE = 1 ) AND ( EX-OK = 1 ).


NEXT-OUVAUT.
          FIND NEXT OUVAUT WITHIN AO
               AT END MOVE 1 TO FIN-AO
                      GO TO AO-EXIT.
          PERFORM FETCH-OWNER.

AO-EXIT.
          EXIT.

NEXT-EXEMP.
          FETCH NEXT EXEMPLAIRE WITHIN OE
                AT END MOVE 1 TO FIN-OE
                       GO TO NEXT-EX-EXIT.
          PERFORM VERIF-EX-OK.
          IF EX-OK = 1
             MOVE 1 TO TROUVE
             DISPLAY NUMEX OF EXEMPLAIRE.

NEXT-EX-EXIT.
          EXIT.

VERIF-EX-OK.
          IF EXE EMPTY
             MOVE 1 TO EX-OK
          ELSE
             MOVE 0 TO EX-OK.
```

```
*****************************************************************
* APPLICATION 18.
* Supprimer un exemplaire d'un ouvrage (le numero est donnee).
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI18.

ENVIRONMENT DIVISION.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.

WORKING-STORAGE SECTION.
01 SEED PIC 9(5) VALUE 967.
01 A-NUM COMP-1.
01 X1  PIC 9(5).
01 I PIC 9999.
01 DEL-FINI PIC 9.
01 FIN-OE   PIC 9.

PROCEDURE DIVISION.
100-BEGIN.
         CALL "LIB$INIT_TIMER".
         MOVE  0 TO I.
         PERFORM BOUCL UNTIL I > 10.
         CALL "LIB$SHOW_TIMER".
         STOP RUN.

BOUCL.
        READY PROTECTED UPDATE.
        CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
        MULTIPLY A-NUM BY 1000 GIVING X1.
        MOVE X1 TO NUMOU OF OUVRAGE.
        FIND FIRST OUVRAGE WITHIN SYSTOUV USING NUMOU OF OUVRAGE.
        MOVE 0 TO DEL-FINI.
        IF OE NOT EMPTY
           PERFORM DEL-UN-EX THRU DEL-EXIT.
        IF DEL-FINI = 0
           DISPLAY " IL N'Y A PAS D'EXEMPLAIRE
                     A SUPPRIMER POUR OUVRAGE : ", X1.
        COMMIT.
        COMPUTE I = I + 1.

DEL-UN-EX.
           FETCH FIRST EXEMPLAIRE WITHIN OE.
           IF EXE EMPTY
              MOVE 1 TO DEL-FINI
              ERASE ALL
              GO TO DEL-EXIT
           ELSE
              PERFORM NEXT-EXEMPLAIRE THRU NEXT-EX-EXIT
                      UNTIL FIN-OE = 1.

DEL-EXIT.
          EXIT.

NEXT-EXEMPLAIRE.
          FETCH NEXT EXEMPLAIRE WITHIN OE
```

```
                AT END MOVE 1 TO FIN-OE
                       GO TO NEXT-EX-EXIT.
                IF EXE EMPTY
                   ERASE ALL
                   MOVE 1 TO DEL-FINI
                   MOVE 1 TO FIN-OE
                   GO TO NEXT-EX-EXIT.

NEXT-EX-EXIT.
           EXIT.
```

```
*****************************************************************
* APPLICATION 21 :
* Enregistrer la restitution d'un exemplaire emprunte
* de numero donne
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI21.

ENVIRONMENT DIVISION.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.

WORKING-STORAGE SECTION.
01 I PIC 99 VALUE 1.
01 NUMOU1 PIC 9.
01 EX-NOEXIST PIC 9.

PROCEDURE DIVISION.
100-BEGIN.
        CALL "LIB$INIT_TIMER".
        PERFORM BIBLIO THRU BIBLIO-EXIT UNTIL I > 10.
        CALL "LIB$SHOW_TIMER".
        STOP RUN.

BIBLIO.
        READY PROTECTED UPDATE.
        MOVE 655 TO NUMEX OF EXEMPLAIRE.
        MOVE 0 TO EX-NOEXIST.
        FIND FIRST EXEMPLAIRE WITHIN SYSTEXEMP
             USING NUMEX OF EXEMPLAIRE
             AT END MOVE 1 TO EX-NOEXIST.
        IF EX-NOEXIST = 1
           DISPLAY "EXEMPLAIRE 655 N'EXISTE PAS ! "
           GO TO BIBLIO-EXIT
        ELSE
           PERFORM RESTITUTION.
        COMMIT.
        COMPUTE I = I + 1.

BIBLIO-EXIT.
        EXIT.

RESTITUTION.
        IF EXE NOT EMPTY
           FETCH FIRST EMPRUNT WITHIN EXE
           FETCH OWNER WITHIN EMP
           MOVE DATEDEBUT OF EMPRUNT TO DATE-DEBUT OF EMPRUNTARCH
           MOVE 872210 TO DATE-FIN OF EMPRUNTARCH
           STORE EMPRUNTARCH
           FIND FIRST EMPRUNT WITHIN EXE
           ERASE ALL.
```

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* APPLICATION 25
* PROGRAMME DE CHARGEMENT /DBMS
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

IDENTIFICATION DIVISION.
PROGRAM-ID.  CHARCOD.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. VAX-11.
OBJECT-COMPUTER. VAX-11.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB BIBLSUB WITHIN BIBLIO1 FOR BIBLIO1.
LD KEEPLIST-1.

WORKING-STORAGE SECTION.
01 P     PIC 99 VALUE 5.
01 L     PIC 99.
01 Y     PIC 99.
01 B     PIC 99 VALUE 22.
01 N     PIC 9999.
01 COMPT PIC 9(8).
01 K     PIC 9999.
01 I     PIC 99.
01 J     PIC 999.
01 H     PIC 9999.
01 NUMERO-DE-BASE PIC 9999.

********************************************************
* DESCRIPTION DE DONNEE POUR GENERATEUR D' ALEATOIR
********************************************************
01 SEED    PIC 9(5) VALUE 967.
01 A-NUM   COMP-1.
01 X1   PIC 9(5).

********************************************************
* DESCRIPTION DE DONNEES POUR EMPRUNTEUR ET MOTCLE
********************************************************
01 EMPRUNTEUR1.
  03 TETE-EMPRUNT PIC X VALUE "E".
  03 EMPRUNT-NOM PIC X(5).
01 ADRESSE1.
  03 TETE-ADR PIC X VALUE "A".
  03 CORP-ADRESSE PIC X(5).
01 MOTCLE1.
  03 TETE-MOT PIC X VALUE "M".
  03 CORP-MOTCLE PIC X(5).

********************************************************
*  DESCRIPTION DE DONNEES POUR CHARGER L'AUTEUR
********************************************************
01 SS-1.
  03 FILLER PIC 9(10) VALUE ZEROS.
01 REDEFINES SS-1.
  03 S OCCURS 5 TIMES PIC 99.
01 SCON-1.
  03 FILLER PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
```

```
01 REDEFINES SCON-1.
  03 SCON OCCURS 26 TIMES PIC X.
01 STR-1.
  03 FILLER PIC X(5) VALUE SPACES.
01 REDEFINES STR-1.
  03 STR OCCURS 5 TIMES PIC X.
01 AUTEUR1.
  03 NOM-TETE PIC X VALUE "N".
  03 NOM-CORP PIC X(5).
01 ORIGINE1.
  03 TETE     PIC X VALUE "O".
  03 CONT-ORIGINE PIC X(5).
01 REPART-OU-AU.
  03 FILLER   PIC 9(15) VALUE 400300100100050.
01 REDEFINES REPART-OU-AU.
  03 REP-OU-AU OCCURS 5 TIMES PIC 9(3).

*****************************************************
* DESCRIPTION DE DONNEES POUR CHARGER L'OUVRAGE
*****************************************************
01 ANNEE1.
  03 FILLER PIC X(20) VALUE "46474849505152535455".
  03 FILLER PIC X(20) VALUE "56575859606162636465".
  03 FILLER PIC X(20) VALUE "66676869707172737475".
  03 FILLER PIC X(20) VALUE "76777879808182838485".
01 REDEFINES ANNEE1.
  03 ANNEE2 OCCURS 40 TIMES PIC X(2).
01 ANNEE4.
  03 FILLER PIC XX VALUE "19".
  03 ANNEE3 PIC XX.
01 TITRE1.
  03 TETE-TITRE PIC X.
  03 TITRE-CORP PIC X(5).
01 EDITEUR1.
  03 TETE-EDITEUR PIC X.
  03 EDITEUR-CORP PIC X(5).
01 TITRE2 PIC X(30).
01 W-NUM-OU PIC X(3).

*****************************************************
* DESCRIPTION DE DONNEES POUR CHARGER L'EXEMPLAIRE
*****************************************************
01 ETAGE1.
  03 FILLER PIC 9(18) VALUE 010101020202030303.
  03 FILLER PIC 9(18) VALUE 040404050505060606.
01 REDEFINES ETAGE1.
  03 ETAGE2 OCCURS 18 TIMES PIC 99.
01 TRAVEE1.
  03 FILLER PIC 9(18) VALUE 010203040506070809.
  03 FILLER PIC 9(18) VALUE 101112131415161718.
01 REDEFINES TRAVEE1.
  03 TRAVEE2 OCCURS 18 TIMES PIC 99.
01 RAYON1.
  03 FILLER PIC 9(18) VALUE 010203040506070809.
  03 FILLER PIC 9(18) VALUE 101112131415161718.
01 REDEFINES RAYON1.
  03 RAYON2 OCCURS 18 TIMES PIC 99.
01 REPART-EXEM-OU.
  03 FILLER PIC 9(10) VALUE 4020200502.
  03 FILLER PIC 9(10) VALUE 0101010101.
```

```
01 REDEFINES REPART-EXEM-OU.
  03 REP-EXEM-OU OCCURS 10 TIMES PIC 99.

*****************************************************
* DESCRIPTION DE DONNEES POUR CHARGER L'EMPRUNT
*****************************************************
01 DATE1.
  03 FILLER PIC 9(14) VALUE 01020304050607.
  03 FILLER PIC 9(14) VALUE 08091011121314.
  03 FILLER PIC 9(14) VALUE 15161718192021.
  03 FILLER PIC 9(14) VALUE 22232425262728.
01 REDEFINES DATE1.
  03 DATE2 OCCURS 28 TIMES PIC 99.
01 DATEG.
  03 AN PIC 99 VALUE 87.
  03 MOIS PIC 99.
  03 JOUR PIC 99.
01 DATE4 PIC 9(6).
01 VERF-EX-IN-EMP PIC X.

*****************************************************
* DESCRIPTION DE DONNEES POUR CHARGER EMPRUNTARCH
*****************************************************
01 DATE3 PIC 99.
01 DATEG1.
  03 AN1 PIC 99 VALUE 87.
  03 MOIS1 PIC 99.
  03 JOUR1 PIC 99.
PROCEDURE DIVISION.
100-INIT.
          CALL "LIB$INIT_TIMER".
          READY EXCLUSIVE UPDATE.
          MOVE 0 TO L.
          MOVE 0 TO COMPT.
          PERFORM CHARGER-AUTEUR THRU 200-EXIT.
          COMMIT.
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'AUTEUR ET L'EMPRUNTEUR FINIT".
          READY EXCLUSIVE UPDATE.
          PERFORM CHARGER-OUVRAGE.
          COMMIT.
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'OUVRAGE FINIT".
          READY EXCLUSIVE UPDATE.
          PERFORM CHARGER-OUVAUT.
          COMMIT.
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'OUVAUT ET MCOUV FINIT".
          READY EXCLUSIVE UPDATE.
          PERFORM CHARGER-EXEMPLAIRE.
          COMMIT.
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT D'EXEMPLAIRE FINIT".
          READY EXCLUSIVE UPDATE.
          PERFORM CHARGER-EMPRUNT.
          COMMIT.
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT D'EMPRUNT FINIT".
          READY EXCLUSIVE UPDATE.
          PERFORM CHARGER-EMPRUNTARCH.
```

```
          COMMIT.
          CALL "LIBSSHOW_TIMER".
          DISPLAY "CHARGEMENT D' EMPRUNTARCH ET PROG FINIT".
          STOP RUN.

*-----------------------------------------------------------
* MODULE 1 CHARGER-AUTEUR ET EMPRUNTEUR
*-----------------------------------------------------------
CHARGER-AUTEUR.
          IF L = P
             GO TO 100-STORE1
          ELSE
             IF L = 0
                MOVE 1 TO Y
                GO TO GENNOM-2
             ELSE
                COMPUTE Y = S ( L ) + 1
                GO TO GENNOM-2.

100-STORE1.
          PERFORM STORE-AUTEUR.
          IF COMPT < 100
             PERFORM STORE-EMPRUNTEUR.
          COMPUTE COMPT = COMPT + 1.
          IF COMPT > 500
             GO TO 200-EXIT
          ELSE
             IF L > 0
                GO TO GENNOM-1
             ELSE
                GO TO 200-EXIT.

STORE-AUTEUR.
          MOVE STR-1 TO NOM-CORP.
          MOVE STR-1 TO CONT-ORIGINE.
          MOVE COMPT TO NUMAU OF AUTEUR.
          MOVE AUTEUR1 TO NOMAU OF AUTEUR.
          MOVE ORIGINE1 TO ORIGINE OF AUTEUR.
          STORE AUTEUR WITHIN FAUTEUR.

STORE-EMPRUNTEUR.
          MOVE STR-1 TO EMPRUNT-NOM.
          MOVE STR-1 TO CORP-ADRESSE.
          MOVE COMPT TO NUMEM OF EMPRUNTEUR.
          MOVE EMPRUNTEUR1 TO NOMEM OF EMPRUNTEUR.
          MOVE ADRESSE1 TO ADRESSE OF EMPRUNTEUR.
          STORE EMPRUNTEUR WITHIN FEMPR.

GENNOM-1.
          MOVE S ( L ) TO Y
          COMPUTE L = L - 1
          COMPUTE B = B - 1
          COMPUTE Y = Y + 1
          GO TO GENNOM-2.

GENNOM-2.
          IF Y > B
          IF L = 0
             GO TO 200-EXIT
            ELSE
```

```
            GO TO GENNOM-1
         ELSE
            COMPUTE B = B + 1
            COMPUTE L = L + 1
            MOVE Y TO S ( L )
            MOVE SCON ( Y ) TO STR ( L )
            GO TO CHARGER-AUTEUR.

200-EXIT.
         EXIT.

*-----------------------------------------------------
*  MODULE 2 CHARGER-OUVRAGE
*-----------------------------------------------------
CHARGER-OUVRAGE.
         MOVE 0 TO COMPT.
         PERFORM CHARGE-OU1 UNTIL COMPT > 1000.

CHARGE-OU1.
         CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
         MULTIPLY A-NUM BY 40 GIVING X1.
         IF X1 = 0
            MOVE 1 TO X1.
         MOVE ANNEE2 ( X1 ) TO ANNEE3.
         PERFORM CHERCH-AU.
         COMPUTE COMPT = COMPT + 1.

CHERCH-AU.
         CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
         MULTIPLY A-NUM BY 500 GIVING X1.
         MOVE X1 TO NUMAU OF AUTEUR.
         FIND FIRST AUTEUR WITHIN SYSTAUTEUR USING
              NUMAU OF AUTEUR
              AT END GO TO 300-EXIT.
         GET NOMAU OF AUTEUR.
         MOVE NOMAU OF AUTEUR TO TITRE1.
         MOVE NOMAU OF AUTEUR TO EDITEUR1.
         MOVE "T" TO TETE-TITRE.
         MOVE "E" TO TETE-EDITEUR.
         PERFORM STORE-OUVRAGE.

STORE-OUVRAGE.
         MOVE TITRE1 TO TITRE OF OUVRAGE.
         MOVE ANNEE4 TO ANNEE OF OUVRAGE.
         MOVE COMPT TO NUMOU OF OUVRAGE.
         MOVE EDITEUR1 TO EDITEUR OF OUVRAGE.
         STORE OUVRAGE WITHIN FOUV.

300-EXIT.
         ROLLBACK.
         EXIT PROGRAM.

*-------------------------------------------------------------
* MODULE 3 CHARGER-OUVAUT ET MCOUV
*-------------------------------------------------------------
CHARGER-OUVAUT.
         MOVE  50 TO NUMERO-DE-BASE.
         MOVE 1 TO I.
         PERFORM REPART UNTIL I > 5.
```

```
REPART.
         MOVE REP-OU-AU ( I ) TO J.
         MOVE 1 TO K.
         PERFORM REPART1 UNTIL K > J.
         COMPUTE I = I + 1.

REPART1.
         COMPUTE NUMERO-DE-BASE = NUMERO-DE-BASE + 1.
         MOVE NUMERO-DE-BASE TO NUMOU OF OUVRAGE.
         FIND FIRST OUVRAGE WITHIN SYSTOUV
              USING NUMOU OF OUVRAGE
              AT END GO TO 300-EXIT.
         MOVE 1 TO H.
         PERFORM RETAIN-AUT UNTIL H > I.
         FREE ALL FROM KEEPLIST-1.
         COMPUTE K = K + 1.

RETAIN-AUT.
         CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
         MULTIPLY A-NUM BY 500 GIVING X1.
         MOVE X1 TO NUMAU OF AUTEUR.
         FIND FIRST AUTEUR WITHIN SYSTAUTEUR
              USING NUMAU OF AUTEUR
              AT END GO TO 300-EXIT.
         IF CURRENT IS WITHIN KEEPLIST-1
            GO TO RETAIN-AUT.
         KEEP CURRENT USING KEEPLIST-1.
         GET NOMAU OF AUTEUR.
         MOVE NOMAU OF AUTEUR TO MOTCLE OF MCOUV.
         INSPECT MOTCLE OF MCOUV REPLACING
                 ALL "N" BY "M" BEFORE "A".
         STORE MCOUV WITHIN FOUV.
         STORE OUVAUT WITHIN FAUTEUR.
         COMPUTE H = H + 1.

*-------------------------------------------------------
* MODULE 4 CHARGER- EXEMPLAIRE
*-------------------------------------------------------
CHARGER-EXEMPLAIRE.
         MOVE 0 TO COMPT.
         MOVE 1 TO I.
         PERFORM REP-EXEMP UNTIL I > 10.

REP-EXEMP.
         MULTIPLY REP-EXEM-OU ( I ) BY 10 GIVING J.
         MOVE 1 TO K.
         PERFORM REP-EXEMP1 UNTIL K > J.
         COMPUTE I = I + 1.

REP-EXEMP1.
         MOVE 1 TO H.
         CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
         MULTIPLY A-NUM BY 18 GIVING L.
         IF L = 0
            MOVE 1 TO L.
         CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
         MULTIPLY A-NUM BY 1000 GIVING X1.
         MOVE X1 TO NUMOU OF OUVRAGE.
         FIND FIRST OUVRAGE WITHIN SYSTOUV USING NUMOU OF OUVRAGE.
         PERFORM STORE-EXEMP UNTIL H > I.
```

```
          COMPUTE K = K + 1.

STORE-EXEMP.
          MOVE COMPT TO NUMEX OF EXEMPLAIRE.
          MOVE ETAGE2 ( L ) TO ETAGE OF EXEMPLAIRE.
          MOVE TRAVEE2 ( L ) TO TRAVEE OF EXEMPLAIRE.
          MOVE RAYON2 ( L ) TO RAYON OF EXEMPLAIRE.
          STORE EXEMPLAIRE WITHIN FEXMP.
          COMPUTE COMPT = COMPT + 1.
          COMPUTE H = H + 1.

*-----------------------------------------------------------------
* MODULE 5 CHARGER-EMPRUNT
*-----------------------------------------------------------------
CHARGER-EMPRUNT.
          MOVE 0 TO COMPT.
          MOVE 10 TO MOIS OF DATEG.
          PERFORM CHARGE-EM UNTIL COMPT > 80.

CHARGE-EM.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 2000 GIVING X1.
          MOVE X1 TO NUMEX OF EXEMPLAIRE.
          FIND FIRST EXEMPLAIRE WITHIN SYSTEXEMP
               USING NUMEX OF EXEMPLAIRE
               AT END GO TO 300-EXIT.
          IF EXE EMPTY
             PERFORM STORE-EMPRUNT
          ELSE
             GO TO CHARGE-EM.

STORE-EMPRUNT.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 100 GIVING J.
          MOVE J TO NUMEM OF EMPRUNTEUR.
          FIND FIRST EMPRUNTEUR WITHIN SYSTEMPR
                USING NUMEM OF EMPRUNTEUR
               AT END GO TO 300-EXIT.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 28 GIVING I.
          IF I = 0
             MOVE 1 TO I.
          MOVE DATE2 ( I ) TO JOUR OF DATEG.
          MOVE DATEG TO DATE4.
          MOVE DATE4 TO DATEDEBUT OF EMPRUNT.
          STORE EMPRUNT WITHIN FEMPR.
          COMPUTE COMPT = COMPT + 1.

*-----------------------------------------------------------------
* MODULE 6 CHARGER-EMPRUNTARCH
*-----------------------------------------------------------------
CHARGER-EMPRUNTARCH.
          MOVE 1 TO COMPT.
          MOVE 10 TO MOIS OF DATEG.
          PERFORM CHARGE-EMPAR UNTIL COMPT > 280.

CHARGE-EMPAR.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 2100 GIVING X1.
          MOVE X1 TO NUMEX OF EXEMPLAIRE.
```

```
        FIND FIRST EXEMPLAIRE WITHIN SYSTEXEMP
             USING NUMEX OF EXEMPLAIRE.
        IF EXE NOT EMPTY
           GO TO CHARGE-EMPAR.
        PERFORM CONSTRUIRE-DONNEE.
        COMPUTE COMPT = COMPT + 1.

CONSTRUIRE-DONNEE.
        CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
        MULTIPLY A-NUM BY 100 GIVING I.
        MOVE I TO NUMEM OF EMPRUNTEUR.
        FIND FIRST EMPRUNTEUR WITHIN SYSTEMPR
             USING NUMEM OF EMPRUNTEUR.
        CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
        MULTIPLY A-NUM BY 28 GIVING J.
        IF J = 0
           MOVE 1 TO J.
        MOVE DATE2(J) TO JOUR OF DATEG.
        COMPUTE DATE3 = 14 + DATE2(J).
        IF DATE3 > 28
           COMPUTE DATE3 = DATE3 - 28
           MOVE 11 TO MOIS1 OF DATEG1
        ELSE
           MOVE MOIS OF DATEG TO MOIS1 OF DATEG1.
        PERFORM STORE-EMPRUNTARCH.

STORE-EMPRUNTARCH.
        MOVE DATE3 TO JOUR1 OF DATEG1.
        MOVE DATEG TO DATE-DEBUT OF EMPRUNTARCH.
        MOVE DATEG1 TO DATE-FIN OF EMPRUNTARCH.
        STORE EMPRUNTARCH WITHIN FEMPR.
```

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*
*B) PROGRAMMES/RDB/COB
*
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

***************************************
* APPLICATION 1
* lister tous les numeros d'OUVRAGE.
***************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI1.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUM-OU1 PIC 9(5).
01 I PIC 99 VALUE 0.

&RDB&     INVOKE DATABASE PATHNAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
          PERFORM BIBLIO UNTIL I > 10.
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

BIBLIO.
&RDB&     START_TRANSACTION READ_ONLY RESERVING
&RDB&           OUVRAGE FOR PROTECTED READ
          PERFORM FETCH-OUV.
&RDB&     COMMIT
          COMPUTE I = I + 1.

FETCH-OUV.
&RDB&     FOR O IN OUVRAGE GET
&RDB&         NUM-OU1 = O.NUMOU
&RDB&                        END-GET
*          DISPLAY NUM-OU1
&RDB&     END-FOR.
```

```
*****************************************************************
* APPLICATION 6 (Optimisation implicite)
* Fournir le(s) ouvrage(s) qui est (sont) ecrit(s)
* par un auteur donne
* (etant donne le numero d'AUTEUR).
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI6.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 I PIC 99 VALUE 0.
01 NUMOU1 PIC 9(5).
01 TITRE1 PIC X(30).
01 ANNEE1  PIC X(4).
01 EDITEUR1 PIC X(30).

&RDB&     INVOKE DATABASE PATHNAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
          PERFORM BIBLIO  UNTIL I > 50
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

BIBLIO.
&RDB&     START_TRANSACTION READ_ONLY RESERVING
&RDB&     OUVRAGE, OUVAUT FOR PROTECTED READ
          PERFORM BOUCL.
&RDB&     COMMIT
          COMPUTE I = I + 1.

BOUCL.
&RDB&     FOR O IN OUVRAGE CROSS OU IN OUVAUT
&RDB&         WITH
&RDB&         O.NUMOU = OU.NUMOU AND
&RDB&         OU.NUMAU = 245
&RDB&         GET
&RDB&         NUMOU1 = O.NUMOU;
&RDB&         TITRE1 = O.TITRE;
&RDB&         ANNEE1 = O.ANNEE;
&RDB&         EDITEUR1 = O.EDITEUR
&RDB&         END-GET
*          DISPLAY NUMOU1, TITRE1, ANNEE1, EDITEUR1
&RDB&     END-FOR.
```

```
*****************************************************************
* APPLICATION 61 : (Optimisation explicite)
* Fournir le(s) ouvrage(s) qui est(sont) ecrit(s)
* par un auteur (etant donne le numero d'AUTEUR) .
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI61.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION,
01 I PIC 99 VALUE 0.
01 NUMOU1 PIC 9(5).
01 TITRE1 PIC X(30).
01 ANNEE1  PIC X(4).
01 EDITEUR1 PIC X(30).
&RDB&     INVOKE DATABASE FILENAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
          PERFORM BIBLIO UNTIL I > 50.
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

BIBLIO.
&RDB&     START_TRANSACTION READ_ONLY RESERVING
&RDB&     OUVRAGE, OUVAUT FOR PROTECTED READ
          PERFORM BOUCL.
&RDB&     COMMIT
          COMPUTE I = I + 1.

BOUCL.
&RDB&     FOR OU IN OUVAUT WITH
&RDB&         OU.NUMAU = 245
&RDB&         FOR O IN OUVRAGE WITH
&RDB&             O.NUMOU = OU.NUMOU
&RDB&         GET
&RDB&         NUMOU1 = O.NUMOU;
&RDB&         TITRE1 = O.TITRE;
&RDB&         ANNEE1 = O.ANNEE;
&RDB&         EDITEUR1 = O.EDITEUR
&RDB&         END-GET
*              DISPLAY NUMOU1, TITRE1, ANNEE1, EDITEUR1
&RDB&         END-FOR
&RDB&     END-FOR.
```

```
*****************************************************************
* APPLICATION 8  (Optimisation implicite)
* Fournir le(s) ouvrage(s) qui est(sont) ecrit(s)
* par un auteur donne (etant donne l'origine d'AUTEUR) .
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI8.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 I PIC 99 VALUE 1.
01 NUMOU1 PIC 9(5).
01 TITRE1 PIC X(30).
01 ANNEE1 PIC X(4).
01 EDITEUR1 PIC X(30).
&RDB&     INVOKE DATABASE PATHNAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIBSINIT_TIMER".
          PERFORM BIBLIO UNTIL I > 10
&RDB&     FINISH
          CALL "LIBSSHOW_TIMER".
          STOP RUN.

BIBLIO.
&RDB&     START_TRANSACTION READ_ONLY RESERVING
&RDB&     AUTEUR, OUVRAGE, OUVAUT FOR PROTECTED READ
          PERFORM BOUCL.
&RDB&     COMMIT
          COMPUTE I = I + 1.

BOUCL.
&RDB&     FOR A IN AUTEUR WITH
&RDB&         A.ORIGINE = "OABCVY"
&RDB&         FOR O IN OUVRAGE CROSS OU IN OUVAUT WITH
&RDB&             OU.NUMAU = A.NUMAU AND
&RDB&             O.NUMOU = OU.NUMOU
&RDB&             GET
&RDB&               NUMOU1 = O.NUMOU;
&RDB&               TITRE1 = O.TITRE;
&RDB&               ANNEE1 = O.ANNEE;
&RDB&               EDITEUR1 = O.EDITEUR
&RDB&             END-GET
*                  DISPLAY NUMOU1, TITRE1, ANNEE1, EDITEUR1
&RDB&         END-FOR
&RDB&     END-FOR.
```

```
*****************************************************
* APPLICATION 81 : (Acces explicit)
* Fournir le(s) ouvrage(s) qui est(sont) ecrit(s)
*  par un auteur (etant donne l' origine d'AUTEUR).
*****************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI81.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION,
01 I PIC 99 VALUE 1.
01 NUMOU1 PIC 9(5).
01 TITRE1 PIC X(30).
01 ANNEE1  PIC X(4).
01 EDITEUR1 PIC X(30).
&RDB&     INVOKE DATABASE FILENAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
          PERFORM BIBLIO UNTIL I > 10.
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

BIBLIO.
&RDB&     START_TRANSACTION READ_ONLY RESERVING
&RDB&     OUVRAGE,AUTEUR,OUVAUT FOR PROTECTED READ
          PERFORM BOUCL.
&RDB&     COMMIT
          COMPUTE I = I + 1.

BOUCL.
&RDB&     FOR A IN AUTEUR WITH
&RDB&         A.ORIGINE = "DABCVY"
&RDB&         FOR OU IN OUVAUT WITH
&RDB&             OU.NUMAU = A.NUMAU
&RDB&             FOR O IN OUVRAGE WITH
&RDB&                 O.NUMOU = OU.NUMOU
&RDB&                 GET
&RDB&                   NUMOU1 = O.NUMOU;
&RDB&                   TITRE1 = O.TITRE;
&RDB&                   ANNEE1 = O.ANNEE;
&RDB&                   EDITEUR1 = O.EDITEUR
&RDB&                 END-GET
*                      DISPLAY NUMOU1, TITRE1, ANNEE1, EDITEUR1
&RDB&             END-FOR
&RDB&         END-FOR
&RDB&     END-FOR.
```

```
*****************************************************
* APPLICATION 9
* Etant donne le titre , l' annee d' edition et
* le nom d' un auteur d' un ouvrage ,
* fournir le numero d' un exemplaire disponible
*****************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI9.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 I PIC 9999.
01 NUMEX1  PIC 9(5).
01 NUMOU1   PIC 9(5).
01 TROUVE PIC 9.
01 FIN-OU PIC 9.
01 EX-FINI PIC 9.
01 OU-FINI PIC 9.
&RDB&     INVOKE DATABASE PATHNAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
&RDB&     START_TRANSACTION READ_ONLY
          PERFORM STARSTREAM.
          MOVE 0 TO TROUVE.
          MOVE 0 TO FIN-OU.
          PERFORM LOOP1 THRU OU-EXIT UNTIL
                  TROUVE = 1 OR FIN-OU = 1.
          PERFORM SORTI-OU.
&RDB&     COMMIT
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

STARSTREAM.
&RDB&     START_STREAM OUVRAGES USING O IN OUVRAGE
&RDB&         CROSS OU IN OUVAUT
&RDB&         CROSS A IN AUTEUR
&RDB&         WITH O.TITRE = "TABCTU" AND
&RDB&         O.ANNEE = "1955" AND
&RDB&         O.NUMOU = OU.NUMOU AND
&RDB&         OU.NUMAU = A.NUMAU AND
&RDB&         A.NOMAU = "NABOHW".

LOOP1.
&RDB&         FETCH OUVRAGES
&RDB&           AT END
                   MOVE 1 TO FIN-OU
&RDB&         END-FETCH
&RDB&         GET NUMOU1 = O.NUMOU
&RDB&         END-GET
              PERFORM CHERCHE-EX
              MOVE 0 TO EX-FINI
              PERFORM LOOP2 THRU EX-EXIT UNTIL EX-FINI = 1
              PERFORM SORTI-EX
```

```
                IF TROUVE = 1
                   GO TO OU-EXIT.

OU-EXIT.
         EXIT.

CHERCHE-EX.
              MOVE 0 TO EX-FINI.
&RDB&         START_STREAM EXEMPLAIRES
&RDB&         USING EX IN EXEMPLAIRE WITH
&RDB&             NOT ANY EM IN EMPRUNT
&RDB&             WITH EX.NUMEX = EM.NUMEX AND
&RDB&             EX.NUMOU = NUMOU1.

LOOP2.
&RDB&             FETCH EXEMPLAIRES AT END
                     MOVE 1 TO EX-FINI
                       GO TO EX-EXIT
&RDB&             END-FETCH
&RDB&             GET NUMEX1 = EX.NUMEX
&RDB&             END-GET
                  DISPLAY NUMEX1
                  MOVE 1 TO EX-FINI
                  MOVE 1 TO TROUVE
                  GO TO EX-EXIT.

EX-EXIT.
         EXIT.

SORTI-EX.
&RDB&          END-STREAM EXEMPLAIRES.

SORTI-OU.
&RDB&          END-STREAM OUVRAGES.
```

```
*****************************************************************
* APPLICATION 18
* Supprimer un exemplaire d' un ouvrage
* ( le numero est donne ).
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI18.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 SEED PIC  9(5) VALUE '967.
01 A-NUM  COMP-1.
01 X1     PIC 9(5).
01 I PIC 9999 VALUE 0.
01 TROUVE PIC 9.
01 UN-EMPRUNT PIC 9.
01 FIN-BOUCL PIC 9.
01 NUMEX1 PIC 9.

&RDB&     INVOKE DATABASE PATHNAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
          MOVE 0 TO I.
          PERFORM BOUCL UNTIL I > 10.
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

BOUCL.
&RDB&     START_TRANSACTION READ_WRITE
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 1000 GIVING X1.
          PERFORM STREAM-EXEMP.
          MOVE  0 TO TROUVE.
          MOVE 0 TO FIN-BOUCL.
          PERFORM BOUCL-EXEMP THRU BOUCL-EX-EXIT UNTIL
                 ( FIN-BOUCL = 1 ) OR  ( TROUVE = 1).
          PERFORM FIN-STREAM.
          IF TROUVE = 0
             DISPLAY "IL N' Y A PAS D'EXEMPLAIRE
                      A SUPPRIMER POUR L'OUVRAGE",X1.
&RDB&     COMMIT
          COMPUTE I = I + 1.

STREAM-EXEMP.
&RDB&     START_STREAM EXEMPLAIRES USING
&RDB&     EX IN EXEMPLAIRE WITH
&RDB&        EX.NUMOU = X1.

BOUCL-EXEMP.
&RDB&     FETCH EXEMPLAIRES AT END
                MOVE 1 TO FIN-BOUCL
                GO TO BOUCL-EX-EXIT
&RDB&     END-FETCH
&RDB&     GET
```

```
&RDB&        NUMEX1 = EX.NUMEX
&RDB&     END-GET.
          MOVE 0 TO UN-EMPRUNT
          PERFORM EX-EMPRUNT
          IF UN-EMPRUNT = 0
             PERFORM DEL-EMPRUNTARCH
             MOVE 1 TO TROUVE
&RDB&        ERASE EX
             GO TO BOUCL-EX-EXIT.

BOUCL-EX-EXIT.
          EXIT.

EX-EMPRUNT.
&RDB&     FOR E IN EMPRUNT WITH
&RDB&         E.NUMEX = EX.NUMEX
              MOVE 1 TO UN-EMPRUNT
&RDB&     END-FOR.

DEL-EMPRUNTARCH.
&RDB&     FOR EMA IN EMPRUNT-ARCH WITH
&RDB&         EMA.NUMEX = EX.NUMEX
&RDB&         ERASE EMA
&RDB&     END-FOR.

FIN-STREAM.
&RDB&     END_STREAM EXEMPLAIRES.
```

```
*****************************************************************
* APPLICATION 21
* Enregistrer la restitution d'un exemplaire emprunte
* de numero donne
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID. APPLI21.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 I PIC 999 VALUE 1.
01 TITRE1 PIC X(30).
01 EX-EXIST PIC 9.
01 NUMEM1 PIC 9(5).
01 DATE-DEBUT1 PIC 9(6).

&RDB&     INVOKE DATABASE PATHNAME 'BIBLIO'

PROCEDURE DIVISION.
100-BEGIN.
          CALL "LIB$INIT_TIMER".
          MOVE 1 TO I.
          PERFORM BOUCL UNTIL I > 10.
*&RDB&     COMMIT
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          STOP RUN.

BOUCL.
&RDB&     START_TRANSACTION READ-WRITE
          MOVE  0 TO EX-EXIST.
          PERFORM VERIF-EMPRUNT.
          IF EX-EXIST = 0
             DISPLAY "EXEMPLAIRE 655 N'EST PAS EMPRUNTE !".
&RDB&     COMMIT
          COMPUTE I = I + 1.

VERIF-EMPRUNT.
&RDB&     FOR E IN EMPRUNT WITH
&RDB&         E.NUMEX = 655
&RDB&         GET NUMEM1 = E.NUMEM;
&RDB&             DATE-DEBUT1 = E.DATE-DEBUT
&RDB&         END-GET
          MOVE 1 TO EX-EXIST
          PERFORM STORE-EMPRUNTARCH
&RDB&     ERASE E
&RDB&     END-FOR.

STORE-EMPRUNTARCH.
&RDB&     STORE EMA IN EMPRUNT-ARCH USING
&RDB&           EMA.NUMEX = 655;
&RDB&           EMA.NUMEM = NUMEM1;
&RDB&           EMA.DATE-DEBUT = DATE-DEBUT1;
&RDB&           EMA.DATE-FIN = 871022
&RDB&     END-STORE.
```

```
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* APPLICATION 25
* PROGRAMME DE CHARGEMENT /RDB
*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

IDENTIFICATION DIVISION.
PROGRAM-ID.  CHARRDB.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. VAX-11.
OBJECT-COMPUTER. VAX-11.

DATA DIVISION.

WORKING-STORAGE SECTION.
01 P      PIC 99 VALUE 5.
01 L      PIC 99.
01 N      PIC 99.
01 Y      PIC 99.
01 B      PIC 99 VALUE 22.
01 I      PIC 99.
01 J      PIC 9(5).
01 H      PIC 9999.
01 NUMERO-DE-BASE PIC 9999.
01 COMPT PIC 9(8).
01 K      PIC 9999.
01 VERF-EMPRUNTEUR PIC X.

*****************************************************
* DESCRIPTION DE DONNEE POUR GENERATEUR D' ALERTOIR
*****************************************************
01 SEED     PIC 9(5) VALUE 967.
01 A-NUM    COMP-1.
01 X1    PIC 9(5).

****************************************************
*  DESCRIPTION DE DONNEES POUR CHARGER L'AUTEUR
****************************************************
01 SS-1.
  03 FILLER PIC 9(10) VALUE ZEROS.
01 REDEFINES SS-1.
  03 S OCCURS 5 TIMES PIC 99.
01 SCON-1.
  03 FILLER PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
01 REDEFINES SCON-1.
  03 SCON OCCURS 26 TIMES PIC X.
01 STR-1.
  03 FILLER PIC X(5) VALUE SPACES.
01 REDEFINES STR-1.
  03 STR OCCURS 5 TIMES PIC X.
01 AUTEUR1.
  03 NOM-TETE PIC X VALUE "N".
  03 NOM-CORP PIC X(5).
01 ORIGINE1.
  03 TETE       PIC X VALUE "O".
  03 CONT-ORIGINE PIC X(5).
01 REPART-DU-AU.
  03 FILLER    PIC 9(15) VALUE 400300100100050.
```

```
01 REDEFINES REPART-OU-AU.
  03 REP-OU-AU OCCURS 5 TIMES PIC 9(3).

*****************************************************************
* DESCRIPTION DE DONNEES POUR CHARGER L'OUVRAGE
*****************************************************************
01 ANNEE1.
  03 FILLER PIC X(20) VALUE "46474849505152535455".
  03 FILLER PIC X(20) VALUE "56575859606162636465".
  03 FILLER PIC X(20) VALUE "66676869707172737475".
  03 FILLER PIC X(20) VALUE "76777879808182838485".
01 REDEFINES ANNEE1.
  03 ANNEE2 OCCURS 40 TIMES PIC X(2).
01 ANNEE4.
  03 FILLER  PIC XX VALUE "19".
  03 ANNEE3  PIC XX.
01 TITRE1.
  03 TETE-TITRE PIC X.
  03 TITRE-CORP PIC X(5).
01 EDITEUR1.
  03 TETE-EDITEUR PIC X.
  03 EDITEUR-CORP PIC X(5).
01 EMPRUNTEUR1.
  03 TETE-EMPRUNT PIC X VALUE "E".
  03 EM-NOM PIC X(5).
01 ADRESSE1.
  03 TETE-AD PIC X VALUE "A".
  03 ADRESSE-CORP PIC X(5).
01 MOT-CLE1.
  03 TETE-MOT PIC X.
  03 FILLER PIC X(29).
01 TITRE2 PIC X(30).
01 W-NUM-OU PIC X(3).

*****************************************************************
* DESCRIPTION DE DONNEES POUR CHARGER L' EXEMPLAIRE
*****************************************************************
01 ETAGE1.
  03 FILLER PIC 9(18) VALUE 010101020202030303.
  03 FILLER PIC 9(18) VALUE 040404050505060606.
01 REDEFINES ETAGE1.
  03 ETAGE2 OCCURS 18 TIMES PIC 99.
01 TRAVEE1.
  03 FILLER PIC 9(18) VALUE 010203040506070809.
  03 FILLER PIC 9(18) VALUE 101112131415161718.
01 REDEFINES TRAVEE1.
  03 TRAVEE2 OCCURS 18 TIMES PIC 99.
01 RAYON1.
  03 FILLER PIC 9(18) VALUE 010203040506070809.
  03 FILLER PIC 9(18) VALUE 101112131415161718.
01 REDEFINES RAYON1.
  03 RAYON2 OCCURS 18 TIMES PIC 99.
01 REPART-EXEM-OU.
  03 FILLER  PIC 9(10) VALUE 4020200502.
  03 FILLER  PIC 9(10) VALUE 0101010101.
01 REDEFINES REPART-EXEM-OU.
  03 REP-EXEM-OU OCCURS 10 TIMES PIC 99.

*****************************************************************
* DESCRIPTION DE DONNEES POUR CHARGER L'EMPRUNT
```

```
*****************************************************************
01 DATE1.
  03 FILLER  PIC 9(14) VALUE 01020304050607.
  03 FILLER  PIC 9(14) VALUE 08091011121314.
  03 FILLER  PIC 9(14) VALUE 15161718192021.
  03 FILLER  PIC 9(14) VALUE 22232425262728.
01 REDEFINES DATE1.
  03 DATE2 OCCURS 28 TIMES PIC 99.
01 DATEG.
  03 AN PIC 99 VALUE 87.
  03 MOIS PIC 99.
  03 JOUR PIC 99.
01 DATE4 PIC 9(6).

*****************************************************************
* DESCRIPTION DE DNNEES POUR CHARGER EMPRUNT-ARCH
*****************************************************************
01 DATE3 PIC 99.
01 DATEG1.
  03 AN1 PIC 99 VALUE 87.
  03 MOIS1 PIC 99.
  03 JOUR1 PIC 99.
01 DATE5 PIC 9(6).
01 VER-NUMEX PIC X.
&RDB&   INVOKE DATABASE FILENAME 'BIBLIO'

PROCEDURE DIVISION.
100-INIT.
          CALL "LIB$INIT_TIMER".
&RDB&     START_TRANSACTION READ_WRITE RESERVING
&RDB&           AUTEUR FOR EXCLUSIVE WRITE,
&RDB&           EMPRUNTEUR FOR EXCLUSIVE WRITE
          MOVE 0 TO L.
          MOVE 0 TO COMPT.
          PERFORM CHARGER-AUTEUR THRU 200-EXIT.
&RDB&     COMMIT
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'AUTEUR ET L'EMRUNTEUR FINIT".
&RDB&     START_TRANSACTION READ_WRITE RESERVING
&RDB&           AUTEUR FOR PROTECTED READ,
&RDB&           OUVRAGE FOR EXCLUSIVE WRITE
          PERFORM CHARGER-OUVRAGE.
&RDB&     COMMIT
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'OUVRAGE FINIT".
&RDB&     START_TRANSACTION READ_WRITE RESERVING
&RDB&           AUTEUR FOR PROTECTED READ,
&RDB&           OUVRAGE FOR PROTECTED READ,
&RDB&           MCOUV FOR EXCLUSIVE WRITE,
&RDB&           OUVAUT FOR EXCLUSIVE WRITE
          PERFORM CHARGER-OUVAUT.
&RDB&     COMMIT
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'OUVAUT ET MCOUV FINIT".
&RDB&     START_TRANSACTION READ_WRITE RESERVING
&RDB&           OUVRAGE FOR PROTECTED READ,
&RDB&           EXEMPLAIRE FOR EXCLUSIVE WRITE
&RDB&           EVALUATING NUM-OU-EX-EXIST AT COMMIT_TIME
          PERFORM CHARGER-EXEMPLAIRE.
&RDB&     COMMIT
```

```
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'EXEMPLAIRE FINI".
&RDB&     START_TRANSACTION READ_WRITE RESERVING
&RDB&           EXEMPLAIRE FOR PROTECTED READ,
&RDB&           EMPRUNTEUR FOR PROTECTED READ,
&RDB&           EMPRUNT FOR EXCLUSIVE WRITE
&RDB&           EVALUATING NUM-EMP-EXIST AT COMMIT_TIME
          PERFORM CHARGER-EMPRUNT.
&RDB&     COMMIT
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT DE L'EMPRUNT ET PROG FINI".
&RDB&     START_TRANSACTION READ_WRITE RESERVING
&RDB&           EMPRUNT FOR PROTECTED READ,
&RDB&           EMPRUNTEUR FOR PROTECTED READ,
&RDB&           EXEMPLAIRE FOR PROTECTED READ,
&RDB&           EMPRUNT-ARCH FOR EXCLUSIVE WRITE
          PERFORM CHARGER-EMPRUNTARCH
&RDB&     COMMIT
&RDB&     FINISH
          CALL "LIB$SHOW_TIMER".
          DISPLAY "CHARGEMENT D' EMPRUNTARCH ET PROG FINIT".
          STOP RUN.

*----------------------------------------
*MODULE 1 CHARGER-AUTEUR ET EMPRUNTEUR
*----------------------------------------
CHARGER-AUTEUR.
          IF L = P
             GO TO 100-STORE1
          ELSE
             IF L = 0
                MOVE 1 TO Y
                GO TO GENNOM-2
             ELSE
                COMPUTE Y = S ( L ) + 1
                GO TO GENNOM-2.

100-STORE1.
          PERFORM STORE-AUTEUR.
          IF COMPT < 100
            PERFORM STORE-EMPRUNTEUR.
          COMPUTE COMPT = COMPT + 1.
          IF COMPT > 500
             GO TO 200-EXIT
          ELSE
             IF L > 0
                GO TO GENNOM-1
             ELSE
                GO TO 200-EXIT.

STORE-AUTEUR.
          MOVE STR-1 TO NOM-CORP.
          MOVE STR-1 TO CONT-ORIGINE.
&RDB&     STORE A IN AUTEUR USING
&RDB&           A.NUMAU = COMPT ;
&RDB&           A.NOMAU = AUTEUR1 ;
&RDB&           A.ORIGINE = ORIGINE1
&RDB&     END-STORE
          MOVE K TO K.
```

```
STORE-EMPRUNTEUR.
          MOVE STR-1 TO EM-NOM.
          MOVE STR-1 TO ADRESSE-CORP.
&RDB&     STORE E IN EMPRUNTEUR USING
&RDB&           E.NUMEM = COMPT ;
&RDB&           E.NOMEM = EMPRUNTEUR1 ;
&RDB&           E.ADRESSE = ADRESSE1
&RDB&     END-STORE
          MOVE K TO K.

GENNOM-1.
          MOVE S ( L ) TO Y
          COMPUTE L = L - 1
          COMPUTE B = B '- 1
          COMPUTE Y = Y + 1
          GO TO GENNOM-2.

GENNOM-2.
          IF Y > B
          IF L = 0
             GO TO 200-EXIT
            ELSE
             GO TO GENNOM-1
          ELSE
             COMPUTE B = B + 1
             COMPUTE L = L + 1
             MOVE Y TO S ( L )
             MOVE SCON ( Y ) TO STR ( L )
             GO TO CHARGER-AUTEUR.

200-EXIT.
          EXIT.

*-----------------------------------------
* MODULE 2 CHARGER-OUVRAGE
*-----------------------------------------
CHARGER-OUVRAGE.
          MOVE 0 TO COMPT.
          PERFORM CHARGE-OU1 UNTIL COMPT > 1000.

CHARGE-OU1.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 40 GIVING X1.
          IF X1 = 0
             MOVE 1 TO X1.
          MOVE ANNEE2 ( X1 ) TO ANNEE3.
          PERFORM CHERCH-AU.
          COMPUTE COMPT = COMPT + 1.

CHERCH-AU.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 500 GIVING X1.
&RDB&     FOR A IN AUTEUR WITH
&RDB&     A.NUMAU = X1
&RDB&      GET W-NUM-OU = A.NUMAU;
&RDB&          TITRE1 = A.NOMAU;
&RDB&          EDITEUR1 = A.ORIGINE
&RDB&      END-GET
&RDB&     END-FOR
          MOVE "T" TO TETE-TITRE.
```

```
          MOVE "E" TO TETE-EDITEUR.
          PERFORM STORE-OUVRAGE.

STORE-OUVRAGE.
&RDB&     STORE O IN OUVRAGE USING
&RDB&          O.NUMOU = COMPT ;
&RDB&          O.TITRE = TITRE1 ;
&RDB&          O.ANNEE = ANNEE4 ;
&RDB&          O.EDITEUR = EDITEUR1
&RDB&     END-STORE.

*-----------------------------------------------
* MODULE 3 CHARGER-OUVAUT ET MCOUV
*-----------------------------------------------
CHARGER-OUVAUT.
          MOVE 50 TO NUMERO-DE-BASE.
          MOVE 1 TO I.
          PERFORM REPART UNTIL I > 5.

REPART.
          MOVE REP-OU-AU ( I ) TO J.
          MOVE 1 TO K.
          PERFORM REPART1 UNTIL K > J.
          COMPUTE I = I + 1.

REPART1.
          COMPUTE NUMERO-DE-BASE = NUMERO-DE-BASE + 1.
          MOVE 1 TO H.
          PERFORM CHERCHE-AUT UNTIL H > I.
          COMPUTE K = K + 1.

CHERCHE-AUT.
          CALL "MTHsRANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 500 GIVING X1.
&RDB&     FOR A IN AUTEUR WITH
&RDB&          A.NUMAU = X1
&RDB&        GET
&RDB&          MOT-CLE1 = A.NOMAU
&RDB&        END-GET
&RDB&     END-FOR
          MOVE "M" TO TETE-MOT.
          PERFORM STORE-OUV-MC.
          COMPUTE H = H + 1.

STORE-OUV-MC.
&RDB&     STORE OA IN OUVAUT USING ON ERROR
               GO TO CHERCHE-AUT
&RDB&                                END-ERROR
&RDB&          OA.NUMOU = NUMERO-DE-BASE;
&RDB&          OA.NUMAU = X1
&RDB&     END-STORE
&RDB&     STORE M IN MCOUV USING ON ERROR
               GO TO CHERCHE-AUT
&RDB&                              END_ERROR
&RDB&          M.NUMOU = NUMERO-DE-BASE:
&RDB&          M.MOTCLE = MOT-CLE1
&RDB&     END-STORE.

*-----------------------------------------
* MODULE 4 CHARGER-EXEMPLAIRE
```

```
*----------------------------------------
CHARGER-EXEMPLAIRE.
          MOVE 0 TO COMPT.
          MOVE 1 TO I.
          PERFORM REP-EXEMP UNTIL I > 10.

REP-EXEMP.
          MULTIPLY REP-EXEM-OU ( I ) BY 10 GIVING J.
          MOVE 1 TO K.
          PERFORM REP-EXEMP1 UNTIL K > J.
          COMPUTE I = I + 1.

REP-EXEMP1.
          MOVE 1 TO H.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 18 GIVING L.
          IF L = 0
             MOVE 1 TO L.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 1000 GIVING X1.
          PERFORM CHAR-EXEMPLAIRE UNTIL H > I.
          COMPUTE K = K + 1.

CHAR-EXEMPLAIRE.
&RDB&     STORE EX IN EXEMPLAIRE USING
&RDB&           EX.NUMOU = X1;
&RDB&           EX.NUMEX = COMPT;
&RDB&           EX.ETAGE = ETAGE2 ( L );
&RDB&           EX.TRAVEE = TRAVEE2 ( L );
&RDB&           EX.RAYON = RAYON2 ( L )
&RDB&     END-STORE
          COMPUTE COMPT = COMPT + 1.
          COMPUTE H = H + 1.

*----------------------------------------
* MODULE 5 CHARGER-EMPRUNT
*----------------------------------------
CHARGER-EMPRUNT.
          MOVE 0 TO COMPT.
          MOVE 10 TO MOIS OF DATEG.
          PERFORM CHARGE-EM UNTIL COMPT > 80.

CHARGE-EM.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 2000 GIVING X1.
          MOVE "N" TO VER-NUMEX.
          PERFORM VERIF-NUMEX THRU VERIF-EX-EXIT.
          IF VER-NUMEX = "N"
             PERFORM STORE-EMPRUNT
          ELSE
             GO TO CHARGE-EM.
          COMPUTE COMPT = COMPT + 1.

STORE-EMPRUNT.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 100 GIVING J.
          CALL "MTH$RANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 28 GIVING I.
          IF I = 0
             MOVE 1 TO I.
```

```
          MOVE DATE2( I ) TO JOUR OF DATEG.
          MOVE DATEG TO DATE4.
&RDB&     STORE EMP IN EMPRUNT USING
&RDB&           EMP.NUMEX = X1;
&RDB&           EMP.NUMEM = J;
&RDB&           EMP.DATE-DEBUT = DATE4
&RDB&     END-STORE.

*-----------------------------------------------------------
* MODULE 6 CHARGER-EMPRUNTARCH
*-----------------------------------------------------------
CHARGER-EMPRUNTARCH.
          MOVE 1 TO COMPT.
          MOVE 10 TO MOIS OF DATEG.
          PERFORM CHARGE-EMPAR UNTIL COMPT > 280.

CHARGE-EMPAR.
          CALL "MTHSRANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 2100 GIVING X1.
          MOVE "N" TO VER-NUMEX.
          PERFORM VERIF-NUMEX THRU VERIF-EX-EXIT.
          IF VER-NUMEX = "Y"
             GO TO CHARGE-EMPAR.
          PERFORM CONSTRUIRE-DONNEE.
          PERFORM STORE-EMPRUNTARCH.
          COMPUTE COMPT = COMPT + 1.

CONSTRUIRE-DONNEE.
          CALL "MTHSRANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 100 GIVING I.
          CALL "MTHSRANDOM" USING SEED GIVING A-NUM.
          MULTIPLY A-NUM BY 28 GIVING J.
          IF J = 0
             MOVE 1 TO J.
          MOVE DATE2( J ) TO JOUR OF DATEG.
          COMPUTE DATE3 = 14 + DATE2( J ).
          IF DATE3 > 28
             COMPUTE DATE3 = DATE3 - 28
             MOVE 11 TO MOIS1 OF DATEG1
          ELSE
             MOVE MOIS OF DATEG TO MOIS1 OF DATEG1.

STORE-EMPRUNTARCH.
          MOVE DATE3 TO JOUR1 OF DATEG1.
          MOVE DATEG TO DATE4.
          MOVE DATEG1 TO DATE5.
&RDB&     STORE EMA IN EMPRUNT-ARCH USING
&RDB&           EMA.NUMEX = X1;
&RDB&           EMA.NUMEM = I;
&RDB&           EMA.DATE-DEBUT = DATE4;
&RDB&           EMA.DATE-FIN = DATE5
&RDB&     END-STORE.

VERIF-NUMEX.
&RDB&     FOR E IN EMPRUNT WITH
&RDB&         E.NUMEX = X1
          MOVE "Y" TO VER-NUMEX
          GO TO VERIF-EX-EXIT
&RDB&     END-FOR.
```

```
VERIF-EX-EXIT.
          EXIT.
```