UNIVERSITE DE GENEVE
FACULTE DE SES – DEPARTEMENT COMIN

MEMOIRE DE LICENCE EN INFORMATIQUE DE GESTION

Réalisation d'une nouvelle organisation informatique au centre de calcul du CERN pour la gestion des utilisateurs, des budgets de groupe, et des statistiques.

Professeur: M. M. LEONARD
Assistants: M. R. BURSENS et M. I. PRINCE

Auteurs: Silvio BOLOGNA et Catherine Jean DIXON

Octobre 1987

TABLE DES MATIÈRES

0.1 Avant-propos

0.1.1 Mise en garde

Les propositions et les conclusions de ce travail n'engagent que la responsabilité de leurs auteurs et en aucun cas celle de l'université ou de ses organes.

0.1.2 Remerciements

Nous remercions le professeur Léonard et ses assistants Messieurs Bursens et Prince pour leurs utiles conseils et encouragements.

Nous sommes particulièrement reconnaissant à l'équipe de Gestion de Base de Données du CERN, en particulier à Mr. S. Santiago qui nous a proposé l'argument de ce travail, et qui a surveillé attentivement son acheminement. Nous sommes également reconnaissant à Mme. G. Ferran et Mr. G. Moorhead pour leurs aides et suggestions lors des stades préliminaires de l'implantation, ainsi qu'à Mr. A. Koppanyi et Mr. T. Osborne pour les discussions utiles que nous avons eu avec eux.

Finalement, un grand merci à la direction du CERN pour nous avoir donné la possibilité d'utiliser ses ressources.

CHAPITRE 1
INTRODUCTION

1.1 Présentation générale du CERN

1.1.1 Ses buts

L'organe appelé Conseil Européen pour la Recherche Nucléaire (CERN) a été institué en février 1952 et, au cours de la même année, Genève fut choisie comme siège de la future organisation. La Convention établissant l'Organisation européenne pour la recherche nucléaire a été signée en 1953, et le 29 septembre 1954 cette convention entrait officiellement en vigueur, après avoir été ratifiée par un nombre suffisant d'Etats membres.

L'Organisation, selon les buts qui lui sont assignés dans le texte original de la Convention, "assure la collaboration entre Etats européens pour les recherches nucléaires de caractère purement scientifique et fondamental, ainsi que pour d'autres recherches en rapport essentiel avec celles-ci".

1.1.2 Ses structures

Le CERN comprend actuellement 14 Etats membres: Allemagne, Autriche, Belgique, Danemark, Espagne, France, Grèce, Italie, Norvège, Pays-Bas, Portugal, Royaume-Uni, Suède, Suisse. D'autres Etats ont un statut d'observateur.

Le Conseil est l'organe souverain qui régit les activités du CERN. Il est composé de deux délégués de chaque Etat membre, lesquels peuvent être accompagnés par des conseillers. Les membres du Conseil et leurs conseillers, ainsi que les membres des Comités spécialisés, assurent la liaison avec les autorités des Etats membres.

1.2 La Division Données et Documents (DD).

Cette Division est le Département Informatique du CERN.

1.2.1 Son Rôle

Ce département emploie environ 300 personnes (pour 4000 au CERN) et se décompose en sept groupes, eux même décomposés en sections.

Il a pour principale vocation, le soutien informatique des chercheurs physiciens du CERN. Sa présence est donc essentiellement technique et très peu de recherches en informatique pure y sont entreprises. Sa politique penche vers la maintenance de produits informatiques achetés à l'extérieur et non vers leur conception (tout du moins pour les gros logiciels) au sein du CERN. Cette tendance s'explique par le simple fait que le CERN est avant tout un centre de recherche en physique.

En fait, beaucoup de physiciens programment eux-mêmes et n'emploient le département qu'à titre de conseil quand ils ont des problèmes ou qu'ils ressentent la nécessité d'acquérir de nouveaux produits informatiques. Avant de venir s'adresser au 'DD', ils bénéficient du soutien d'informaticiens détachés dans leur propre division.

1.2.2 Ses Outils.

Le CERN dispose d'une grande puissance de calcul répartie sur tous les départements. Plusieurs centaines de mini−ordinateurs de puissance comparable à celle du Vax 8300 assistent les expériences de physique et sont gérés par les départements les possédant.

La DD gère pour sa part deux grosses machines, une Siemens 7890/S et un IBM 3090−200 auxquelles viendront bientôt s'ajouter une Cray X−MP/48 remplaçant une machine CDC (Cyber 875 et 835) jugée obsolète. Ces machines sont accessibles via le système VM/CMS en interactif, et via le système MVS Wylbur pour les travaux batch. Plusieurs Vax (8650, 8650, 8300, Microvax, vax-station avec des systèmes VMS ou UNIX suivant les machines) sont aussi à la disposition du département. Toutes ces machines sont accessibles par le biais d'environ 2000 terminaux, dont une bonne partie en libre service, à l'aide de systèmes de commutation Gandalf appelés Index.

Plusieurs réseaux relient ces machines, notamment CERNET et DECNET reliant la plupart des Vax. Pour les communications avec l'extérieur, le système Extase permet accès au réseau suisse de commutation de paquets utilisant la norme X25. On peut ensuite, via ce réseau, accéder à des réseaux internationaux comme Transpac (France), Datex.P (Allemagne), PSS (Royaume-Uni), Tymnet et Telenet (USA) etc ...

Des lignes directes existent avec divers instituts européens comme le Laboratoire de physique des particules d'Annecy (LAPP), Saclay, l'INFN Rome, les Universités de Genève, le laboratoire d'Appleton Rutherford, etc...

1.3 La section Gestion de Bases de Données.

1.3.1 Son Rôle.

La section GBD assure la maintenance du SGBD Oracle. Ce travail se divise en trois grandes parties :

– Réceptionner les nouveaux produits (α – tests, β – tests, releases, updates, nouvelles versions) Oracle et les implanter (fixer les paramètres de fonctionnement du système) suivant les Systèmes d'exploitation et les machines utilisées (VMS sur Vax, VM/CMS sur IBM, DOS sur IBM PC).

– Assurer la fonction d'administrateur des Bases de Données pour les gros systèmes. (Création de nouvelles bases, définition d'espace mémoire, conseils techniques etc ...)

– Aide à la conception pour les nouvelles applications créées par les utilisateurs des machines du CERN.

Cette section gère, pour le compte d'un autre département du CERN, un Vax 8650 (VMS) pour le développement d'applications et le test des nouvelles versions d'Oracle et pour la planification des travaux d'installation du LEP. Cette machine est totalement consacrée à Oracle.
La section doit parallèlement assurer la maintenance d'Oracle sur des Vax 8800 et 8650 destinés aux physiciens et surtout sur l'IBM 3090 VM/CMS (sur lequel nous travaillons).

1.3.2 Ses Rapports avec Oracle Corporation.

Oracle est un produit en constante évolution. La Société Oracle Corp. ne fournit que le code objet de ses logiciels. Il arrive que des "bugs" soient découverts. Il s'agit alors de reproduire ces "bugs" sur une base standard (connue d'Oracle Corp.) avec une application la plus simple possible et de leur envoyer afin qu'ils corrigent leur logiciel. Les modifications reviennent alors soit sous la forme d'updates (petits bouts de logiciel modifiés), soit sous la forme de releases ("petites nouvelles versions") si les modifications sont relativement importantes.

De nouveaux produits qui n'ont pas encore étés totalement testés, ou pas testés du tout sont envoyés par Oracle Corporation. Le groupe GBD ne paie pas ces logiciels mais aide à leur mise au point.

CHAPITRE 2
ORACLE : UN SGBD RELATIONNEL

2.1 Présentation Générale.

2.1.1 Généralités.

Oracle est un des Système de Gestion de Bases de Données Relationnel le plus répandu. Il est disponible sur un grand nombre d'ordinateurs (...minis, micros) de marque différente, ce qui permet aux entreprises d'avoir un matériel hétérogène avec un environnement software standard pour le développement des applications. Son utilisation se fait au travers de divers utilitaires programmés sur une couche supérieure au système proprement dit. Cependant, cette structuration est totalement transparente pour l'utilisateur et l'administrateur de Base de Données. C'est ce qui en fait son intérêt et son succès.

En revanche, cette transparence oblige parfois le concepteur de Base de Données à procéder par tâtonnements pour trouver des solutions convenables en temps de réponse ainsi qu'en espace mémoire. Dans la pratique, beaucoup de manipulations tiennent plus du "bricolage" que d'une méthode rigoureuse.

Les données se manipulent par le biais de SQL, devenu le langage standard international par excelence. C'est un langage qui est proche de la langue naturelle, ce qui est agréable pour l'utilisateur qui fait l'effort de l'apprendre.

Ce système propose par rapport aux langages de programmation classiques l'immense intérêt de pouvoir déplacer facilement la "frontière" rapidité d'exécution / espace mémoire, par les simples commandes "create et drop index" de l'utilitaire SQLPLUS, sans pour autant modifier les structures internes des données et des applications.

Une chose très importante est à remarquer quand on suit l'évolution des utilitaires Oracle. Le système tend de plus en plus à se passer des langages de programmation, même pour des applications compliquées. Il devient presque autonome et peut être assimilé à un langage de programmation spécialisé dans le traitement interactif de grands ensembles de données. Cette évolution est surtout due à l'utilitaire IAF, se décomposant en IAG (Interactive Application Generator) et IAP (Interactive Application Processor). IAG a été évolué en un autre utilitaire s'appelant FASTFORM, ayant l'avantage de ne pas devoir répondre interactivement à toutes les questions, pour des applications simples. L'etape suivante a été la naissance de SQLFORMS qui reprend les caracteristiques de IAF en les améliorant avec une nouvelle "philosophie" de travail. En effet, ce nouveau utilitaire permet de créer des écrans d'application appelés "formes" (sans devoir spécifier lignes, colonnes, etc...), grâce à un système de fenêtres et ménus mettant à disposition toutes les options nécessaires, en accès direct.

On peut définir ORACLE comme suit :

C'est un système relationnel (tout est défini à l'aide de relations, même le dictionnaire des données) qui :

1) fournit une interface avec un langage assurant dynamiquement la définition, l'interrogation, la mise à jour et le contrôle d'un ensemble de relations.

2) assure l'indépendance des données par rapport aux structures physiques de stockage et des chemins d'accès à l'information.

3) assure l'intégrité des données en contrôlant les accès, la concurrence entre usagers et les pannes éventuelles, afin de garder la base dans un état cohérent.

Le système est flexible et on peut par conséquent ajouter des relations, des colonnes aux relations existantes ou modifier ces dernières, sans que ceci entraîne une réorganisation de la base, ce qui n'est pas le cas pour tous les SGBD, mêmes relationnels.

2.1.2 Qu'est-ce qu'une Base de Données pour Oracle.

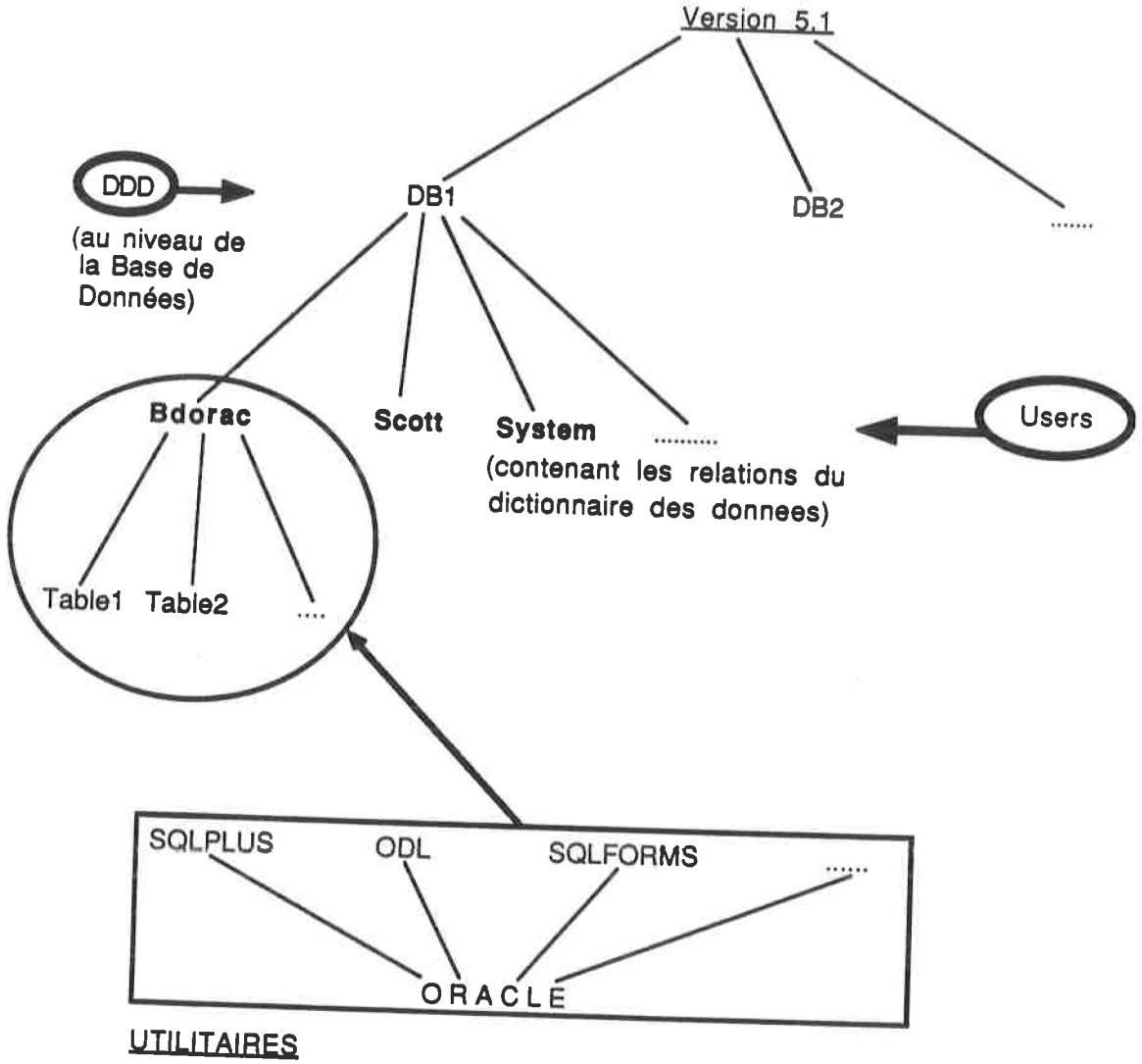

Fig. 1. Configuration du SGBD Oracle.

Il existe toute une hiérarchie dans la structuration d'Oracle. Cette hiérarchie peut être représentée par un arbre dont la racine serait la version d'Oracle sous laquelle "tournent" une ou plusieurs Bases de Données. La structuration des données commence à ce niveau, où l'on regroupe plusieurs "users".

(Exemple : DB1 regroupe BDORAC, SCOTT, ...) Un "user" est un ensemble de relations, accessible par les utilitaires Oracle à condition d'en connaître le nom et le mot de passe (d'où le nom de "user" par analogie avec sa signification pour un système d'exploitation). Grâce à ce mécanisme de "user", chaque utilisateur travaille avec ses données dans la base, selon le format qu'il désire, indépendamment des autres. Un "user" d'une Base de Données peut accéder aux relations d'un autre "user" appartenant à une autre Base de Données grâce à des modules de communications.
Les relations du dictionnaire des données sont définies dans l'un de ces "user", appelé SYSTEM. Chaque autre "user" peut interroger le dictionnaire des données et obtient des informations selon la configuration de son application s'il interroge la liste des relations, des indexes, etc...

Oracle permet à plusieurs utilisateurs d'effectuer des traitements en même temps, sur le même "user". Si deux ou plusieurs utilisateurs travaillent sur la même relation, ils peuvent lire les données en même temps, mais s'ils veulent faire une mise à jour de la même rangée, ils ne peuvent le faire que l'un après l'autre et la relation est verrouillée tant que le premier changement n'est pas effectif dans la base (transaction "commited") c'est après cette transaction que relation est déverouillé et qu'un autre utilisateur peut effectuer son traitement. Le verrouillage d'une relation ou d'une rangée est fait soit automatiquement par Oracle, soit par ordre SQL, spécifié par l'utilisateur dans l'application. Oracle gère lui-même les situations de blocage en interrompant la transaction de l'un des deux utilisateurs en conflit et en envoyant un message lui indiquant qu'il doit relancer son traitement (tout en continuant la transaction de l'autre utilisateur).

On peut distinguer trois cas de protection des données :
- Lorsqu'on parle d'intégrité, cela veut dire que les valeurs entrées dans la Base sont vérifiées par certaines règles pour éviter les erreurs.
- Les données sont considérées en sécurité quand il faut une autorisation pour y accéder (mot de passe).
- En cas de panne, Oracle redémarre la Base sans perte de données.

Le temps pour effectuer un traitement dans Oracle est fonction du nombre d'utilisateurs travaillant sur la machine et de la taille de l'application traitée. (Nous en avons souffert avec notre application sur SQLFORMS aux heures de pointe, alors que tout le monde travaillait sur l'IBM VM/CMS...)
A noter que c'est au niveau d'une base, que l'on choisit la version d'Oracle utilisée.

Pour notre compte, nous travaillons sur une base de test avec la version 5.1 qui est la plus évoluée et la plus riche d'utilitaires (elle contient SQLFORMS). Elle est certes plus agréable à utiliser que la version précédente mais contient encore quelques "bug" que nous découvrons peu à peu, Oracle corporation ne l'ayant pas testée à fond.

2.2 Dictionnaire de données Oracle (DDD)

2.2.1 graphe du DDD

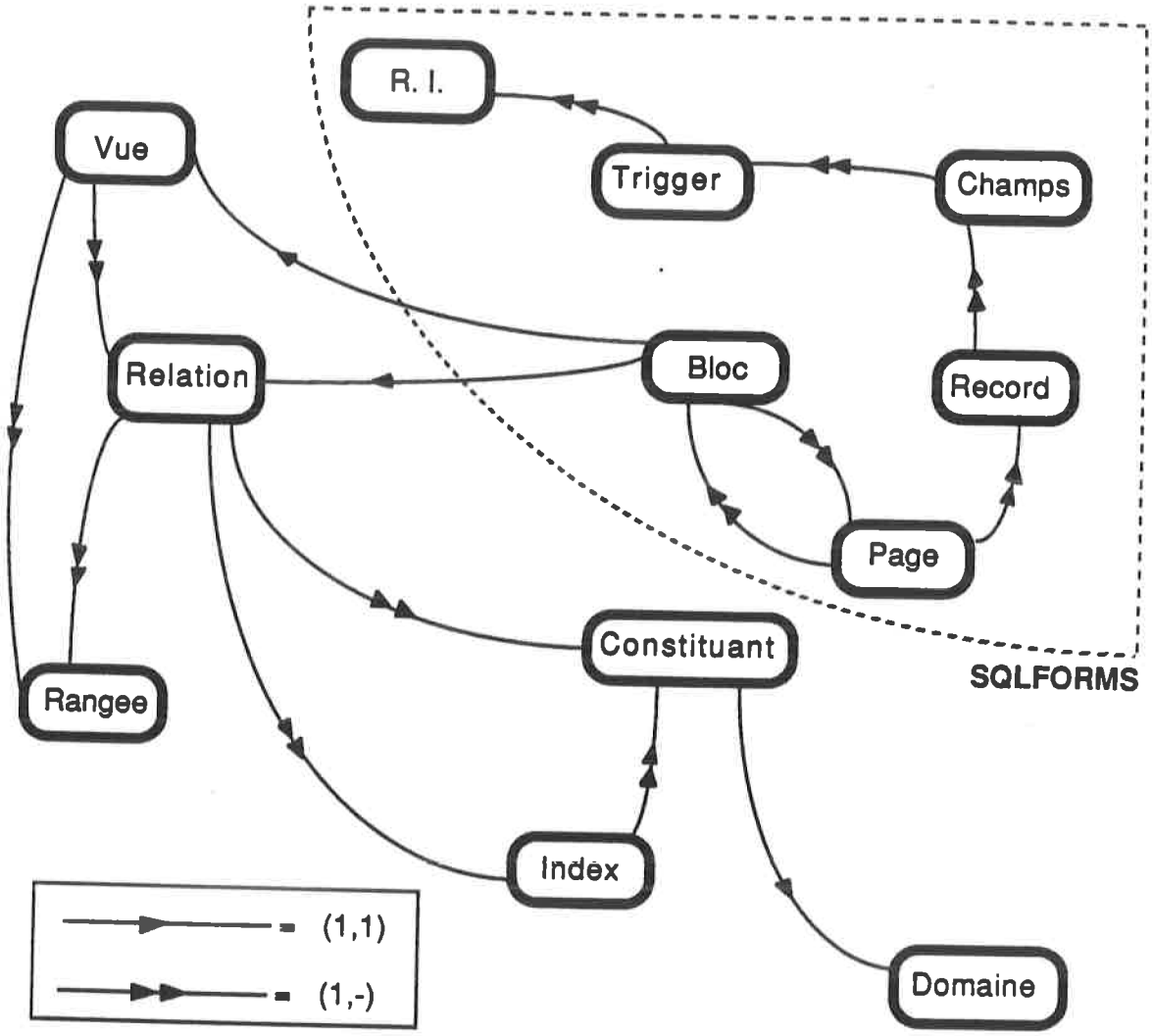

Fig. 2. Représentation des entités du DDD

Dans le DDD sont mentionnées les caractéristiques d'Oracle. Ce SGBD ne consent pas d'associer les règles d'intégrité compliquées (autres que celles sur les domaines) directement aux relations (comme la pluspart des SGBD). On est obligé de les intégrer au système à un niveau supérieur, celui de l'utilitaire qui est soit un programme procédural (du type Cobol, Pascal, C), soit un logiciel integré au système Oracle, non procédural (du type IAF, ou du plus récent SQLFORMS, permettant d'etablir l'interface page-écran à travers laquelle l'utilisateur fait des traitements sur la Base de Données dont la consistance est justement préservée par les règles y étant incorporées).

Les liens entre bloc et relation (ou vue) et ceux apparaissant dans l'encadré sont décrits plus loin dans la section SQLFORMS (cf. Les utilitaires d'Oracle).

Une relation peut contenir plusieurs rangées, mais une rangée donnée ne peut se trouver que dans une seule relation, autrement on aurait une relation double dans la base.

Une relation est formée par plusieurs constituants. Un constituant peut se trouver dans plusieurs relations, avec le même identificateur donné par le concepteur mais avec un signification différente, car avec Oracle, on définit les constituants dans le contexte de définition d'une relation.

A une relation on peut associer plusieurs indexes.
A noter ici qu'un indexe défini unique est l'équivalent d'une clé (au sens du cours). Ceci est un outil puissant pour éviter les rangées doubles dans une relation. Un indexe simple (non unique) n'est rien d'autre qu'un chemin d'accès qui permet de consulter plus rapidement les données (il améliore le temps de réponse). Il faut quand même être judicieux avec cet outil, car si l'on multipliait les indexes dans une relation, le temps de réponse pourrait paradoxalement être pénalisé! (Cas où l'on ait des mises à jour nombreuses).

Un indexe peut être établit sur plusieurs constituants et un constituant peut être contenu dans plusieurs indexes de la même relation. Il est important de dire que les constituants sur lesquels l'indexe est définit peuvent être mis à jour, ce qui n'est pas les cas pour d'autres Bases de Données connues. Néanmoins la mise à jour d'une clé d'une relation peut créer des inconsistances, s'il existe des données dépendant de la même clé dans une autre relation.

Un constituant a un et un seul domaine. On peut modifier ce domaine à condition qu'il n'existe pas de données dans la relation contenant ce constituant.

Le création et la destruction de relations et d'indexes n'entraîne pas une réorganisation de la Base de Donnée, ni la création de constituants supplémentaires, par contre cela serait le cas si on devait supprimer un constituant de la relation. On peut changer le nom d'une relation mais pas le nom d'un constituant.

On a donc vu ce qu'il est possible de faire sur la structure de la Base, mais il est intéressant de savoir comment Oracle montre lui-même son DDD.

2.2.2 Disponibilité du DDD : sous l'utilitaire SQLPLUS

```
TNAME          REMARKS
-------------- ------------------------------------------------------------
Reference Date ORACLE catalog as of 10-Oct-85, installed on 30-JUN-87 17:35:49.
AUDIT_ACCESS   Audit entries for accesses to user's tables/views (DBA sees all)
AUDIT_ACTIONS  Maps auditing action numbers to action names
AUDIT_CONNECT  Audit trail entries for user logon/logoff (DBA sees all users)
AUDIT_DBA      Audit trail entries for DBA activities -- for DBA use only
AUDIT_EXISTS   Audit trail entries for objects which do NOT EXIST -- DBA's only
AUDIT_TRAIL    Audit trail entries relevant to the user (DBA sees all)
CATALOG        Tables and views accessible to user (excluding data dictionary)
CLUSTERS       Clusters and their tables (either must be accessible to user)
CLUSTERCOLUMNS Maps cluster columns to clustered table columns
COL            Specifications of columns in tables created by the user
COLUMNS        Columns in tables accessible to user (excluding data dictionary)
DEFAULT_AUDIT  Default table auditing options
DTAB           Description of tables and views in Oracle Data Dictionary
EXTENTS        Data structure of extents within tables
INDEXES        Indexes created by user and indexes on tables created by user
PARTITIONS     File structure of files within partitions -- for DBA use only
PRIVATESYN     Private synonyms created by the user

                                                        MORE...   CERNVM
PUBLICSYN      Public synonyms
SESSIONS       Audit trail entries for the user's sessions (DBA sees all)
SPACES         Selection of space definitions for creating tables and clusters

TNAME          REMARKS
-------------- ------------------------------------------------------------
STORAGE        Data and Index storage allocation for user's own tables
SYNONYMS       Synonyms, private and public
SYSAUDIT_TRAIL Synonym for sys.audit_trail -- for DBA use only
SYSCATALOG     Profile of tables and views accessible to the user
SYSCOLAUTH     Directory of column update access granted by or to the user
SYSCOLUMNS     Specifications of columns in accessible tables and views
SYSEXTENTS     Data structure of tables throughout system -- for DBA use only
SYSINDEXES     List of indexes, underlying columns, creator, and options
SYSPROGS       List of programs precompiled by user
SYSSTORAGE     Summary of all database storage -- for DBA use only
SYSTABALLOC    Data and index space allocations for all tables -- for DBA's
SYSTABAUTH     Directory of access authorization granted by or to the user
SYSTEM_AUDIT   System auditing options -- for DBA use only
SYSUSERAUTH    Master list of Oracle users -- for DBA use only
SYSUSERLIST    List of Oracle users
SYSVIEWS       List of accessible views

                                                        MORE...   CERNVM
TAB            List of tables, views, clusters and synonyms created by the user
TABALLOC       Data and index space allocations for all user's tables
TABQUOTAS      Table allocation (space) parameters for tables created by user
TABLE_AUDIT    Auditing options of user's tables and views (DBA sees all)
VIEWS          Defining SQL statements for views created by the user

TNAME          REMARKS
-------------- ------------------------------------------------------------
DBLINKS        Public and private links to external databases
SYSDBLINKS     All links to external databases -- for DBA use only

44 records selected.
```

Fig.3 . Relations du DDD sous SQLPLUS

Au niveau d'Oracle, le DDD est un groupe de relations qui s'instaure automatiquement lors de la création de la Base de Données, où sont décrits relations colonnes, indexes, etc... Cela permet à l'utilisateur de vérifier ce qu'il a à disposition.
Pour intérroger le DDD globalement, il suffit d'entrer la commande SELECT * FROM DTAB, qui produit une table contenant à son tour, une liste de tables et de vues (Figure 1). Chaque table (ou vue) est munie d'une explication qui indique son contenu. On peut remarquer que DTAB est lui-même contenu dans la liste.

Interrogeons par exemple INDEXES qui doit donner la liste de toutes les indexes uniques (clés) et les indexes non uniques crées par le user (dans ce cas nous-même). On rentre donc la commande SELECT * FROM INDEXES, ce qui donne le résultat suivant :

INAME	ICREATOR					
TNAME	CREATOR					
COLNAMES	INDEXTYPE	IORD	COMPRESSIO		SEQ	CONCATID
XBJ_STAT	BDORAC					
BJ_STAT	BDORAC					
J_CLASS_DEF	UNIQUE	ASC	COMPRESS		1	1
XBJ_STAT	BDORAC					
BJ_STAT	BDORAC					
SYSTEM	UNIQUE	ASC	COMPRESS		2	1
XBJ_STAT	BDORAC					
BJ_STAT	BDORAC					
WEEK	UNIQUE	ASC	COMPRESS		3	1
XBJ_TURNSTAT	BDORAC					
				MORE...	CERNUM	
BJ_TURNSTAT	BDORAC					
J_CLASS_DEF	UNIQUE	ASC	COMPRESS		1	1
XBJ_TURNSTAT	BDORAC					
INAME	ICREATOR					
TNAME	CREATOR					
COLNAMES	INDEXTYPE	IORD	COMPRESSIO		SEQ	CONCATID
BJ_TURNSTAT	BDORAC					
TURN_TIME	UNIQUE	ASC	COMPRESS		2	1
XBJ_TURNSTAT	BDORAC					
BJ_TURNSTAT	BDORAC					
SYSTEM	UNIQUE	ASC	COMPRESS		3	1
XBJ_TURNSTAT	BDORAC					
BJ_TURNSTAT	BDORAC					
WEEK	UNIQUE	ASC	COMPRESS		4	1
				MORE...	CERNUM	

Fig.4 : Relations des indexes sous SQLPLUS

Ceci est un exemple d'information pouvant être consulté dans le DDD qui regroupe aussi toute l'information sur les privilèges donnés par un "user" (au sens Oracle) à un autre "user", ainsi les informations concernant le découpage des relations en mémoire physique.

La protection de ces données est assurée par des vues, (Une vue est une relation virtuelle, sans adresse physique) qui sont établies sur les relations-système, inatteignables par un user "normal" n'ayant pas les privilèges du système. Le DDD pourra seulement être consulté.

CHAPITRE 3
DESCRIPTION GÉNÉRALE DU PROJET

3.1 Situation avant notre arrivée

La situation était la suivante :
il existait (et elles continuent à exister) des innombrables petites Bases de Données conçues individuellement, la plupart par des programmeurs pour des buts bien précis, (stockage de matériel, données produites par les expériences de physique, etc...) mais indépendantes entre elles. Or, en examinant l'ensemble, on se rend compte de existence d'une forte redondance (pas bénéfique du tout, car il y a plusieurs versions à tenir à jour). Une solution aurait consisté à créer un effet d'inertie exercé par l'information d'une Base sur l'information d'autres Bases, mais vu le nombre de Bases existantes, cela serait trop laborieux.

En outre, le groupe UCO ("User Consultancy Office) qui est le plus grand detenteur d'informations sur les systèmes, les groupes et les users avait les problèmes suivants:

- Rapports de système.

Chaque système était individuel (VMS – VM – CRAY) ce qui pose des problèmes aux administrateurs de groupe car ils ne disposaient d'aucune flexibilité au niveau des systèmes.
Un des buts qui est recherché avec une nouvelle base de donnée est justement de permettre une centralisation de l'information de fa$con à voir ce qui se passe au niveau d'autres systèmes.

- Enregistrement.

UCO est l'organe qui doit effectuer, entre autres, l'enregistrement des nouveaux utilisateurs. Pour ne pas créer des enregistrements incompatibles avec ceux qui existent déjà, UCO dispose d'information sur un disque, dans des fichiers REXX – EXEC (le REXX est l'interprète de VM. Ce langage de contrôle permet d'exécuter aisément un ensemble de commandes appartenant à CP, CMS, ou XEDIT. Il correspond au langage Wylbur Exec, CCL (Cyber Control Language), C – Shell scripts (UNIX) et DCL (DEC Control Language). programmes) sur VM, mais qui n'étaient pas très performants ou jugés pas assez satisfaisants. En effet les programmes "exec" lisaient des données, faisaient quelques validations, mais au niveau de la sélection de certaines informations (ex. sélectionner toutes les personnes qui n'ont pas utilisés leur "account" depuis 6 mois) Oracle est un outil beaucoup plus puissant pour le faire. Si l'on voulait mettre en place toute une procédure de vérification des données cela serait trop compliqué avec ces fichiers.

Il était donc nécessaire de rendre l'information consistante en la structurant mieux de façon à l'optimiser et à lui donner une concaténation logique.

3.2 Le projet

Le but du projet est de produire une Base de Données "centrale" Oracle, afin de rationaliser toute l'information concernant les machines, leurs performances, les utilisateurs et les budgets.
Elle doit servir de la façon suivante :

– aux enregistrements des utilisateurs.

Les membres de UCO collectent les données et effectuent les enregistrements nécessaires. Pour cela, ils doivent vérifier les données existantes sur la personne qui est enregistrée afin de valider l'information dans la Base. Après l'enregistrement (correcte) Oracle doit retourner un message au système en question pour que l'enregistrement puisse être pris en compte par le système

– à l'élaboration de statistiques.

Chaque semaine, les systèmes produisent des fichiers de données statistiques (sur la performance des machines, leur utilisation, etc...) qui doivent être stockés dans les relations de la Base (au moyen d'utilitaires ou programmes Fortran), afin de pouvoir être interrogés plus facilement et fournir sélectivement des données pour l'élaboration de graphiques.

– Au stockage de données concernant les groupes et les budgets.

On ne va pas structurer l'information au niveau du de l'utilisateur individuel, mais au niveau du groupe dont on veut garder un historique sur toutes les requêtes et les allocations.

Initialement, toutes les données sont rentrées dans la Base par le biais de fichiers formatés. Par la suite des procédures seront mises en place pour effectuer des insertions et des mises à jour.

On peut résumer comme suit les input et output de la Base après sa création :

Input :

* Données concernant l'utilisateur, saisies par UCO.
* Feedback nocturne sur l'enregistrement des utilisateurs de chaque système.
* Priorités par groupe sur chaque système.
* Changements sur les users et l'annuaire téléphonique.
* Performances et utilisation hebdomadaire de chaque système.
* Statistiques hebdomadaires sur les groupes et les users.
* mises à jours hebdomadaire de l'annuaire téléphonique fournies par l'unité ADP ("Administrator Data Processing").

Output :

* Budgets devant être envoyés aux fichiers du groupe pour chaque système concerné.
* Messages adressés à chaque système pour l'enregistrement de l'utilisateur.
* Rapports de routine.

En règle générale, L'information de la Base sera accessible sans trop de restrictions, aux managers de la division DD ayant besoin de renseignements concernant les ordinateurs.

Autrement, l'accès à la Base est très strictement réglementé, il existe quatre classes d'utilisateurs de la Base :

- Les utilisateurs courants.
- les administrateurs de groupe.
- les administrateurs de système.
- les managers de ressource.

Leurs privilèges s'accroissent au fur et à mesure qu'on passe de la première à la dernière de ces classes.

A part le travail de création de la Base elle-même, il faut créer une procédure de gestion des utilisateurs de toute pièce, permettant d'accéder à la Base avec certaines restrictions (dépendantes de la classe de privilèges possédé par l'utilisateur) où l'on fera surtout des vérifications entre les informations de l'utilisateur (source : UCO) et les informations propres à la personne, contenues dans l'annuaire téléphonique (source : ADP). Il faut absolument que tous les utilisateurs d'ordinateur apparaissent dans cet annuaire sauf exceptions (ex. personne qui vient pour effectuer un travail extraordinaire tout en restant inconnu par l'administration).

La procédure de mise à jour de l'annuaire téléphonique va rester inchangée, même si elle va à l'encontre des principes d'Oracle. En effet elle manipule le fichier de l'annuaire et donc on est obligé de l'exporter de la Base dans un fichier, de faire tourner le programme pour le mettre à jour et ensuite l'importer à nouveau dans la Base...

Il faut enfin tenir compte des toutes les règles qui se présentent lorsqu'on travaille avec un SGBD comme Oracle, pour assurer le plus possible la validité et la cohérence de l'information (ex. lors d'une mise à jour).

3.3 Notre rôle dans le projet

Notre rôle est de mener une certaine analyse (décrite dans la prochaine section) afin de pouvoir créer la structure de la Base (les constituants avec leur domaine, les relations et les clés).
Autre tâche importante est la gestion de cette structure (il faut la mettre à jour car elle n'est pas statique tout au long du projet).
En outre il faut remplir la base à l'aide de fichiers de données (nous ne pouvons pas tester la validité d'une Base sans données) et corriger les éventuelles erreurs présentes dans le fichier d'entrée.
Pour assurer l'intégrité d'au moins une partie de la Base, nous devons également construire le prototype de la procédure de gestion des utilisateurs.
En faites on distingue trois grandes parties dans la Base, décomposable en trois applications distinctes :

- Procédure de gestion des utilisateurs.
- Procédure de gestion des budgets.
- Procédure de gestion des statistiques.

Mais faute de temps, on ne s'occupe que de la première qui est la plus importante et urgente pour le CERN, et aussi la plus difficile car pour des raisons de sécurité de la Base, les règles qui la protègent sont très compliquées.
Notre dernier travail est de fournir un rapport pour que d'autres personnes puissent comprendre la Base et continuer à implémenter les applications de ce projet paraissant simple dans la description, mais qui est complexe dés qu'on en examine les détails.

CHAPITRE 4
ANALYSE

4.1 Description générale

L'analyse globale peut être rapportée à une représentation de la structure du CERN par un arbre hiérarchique.

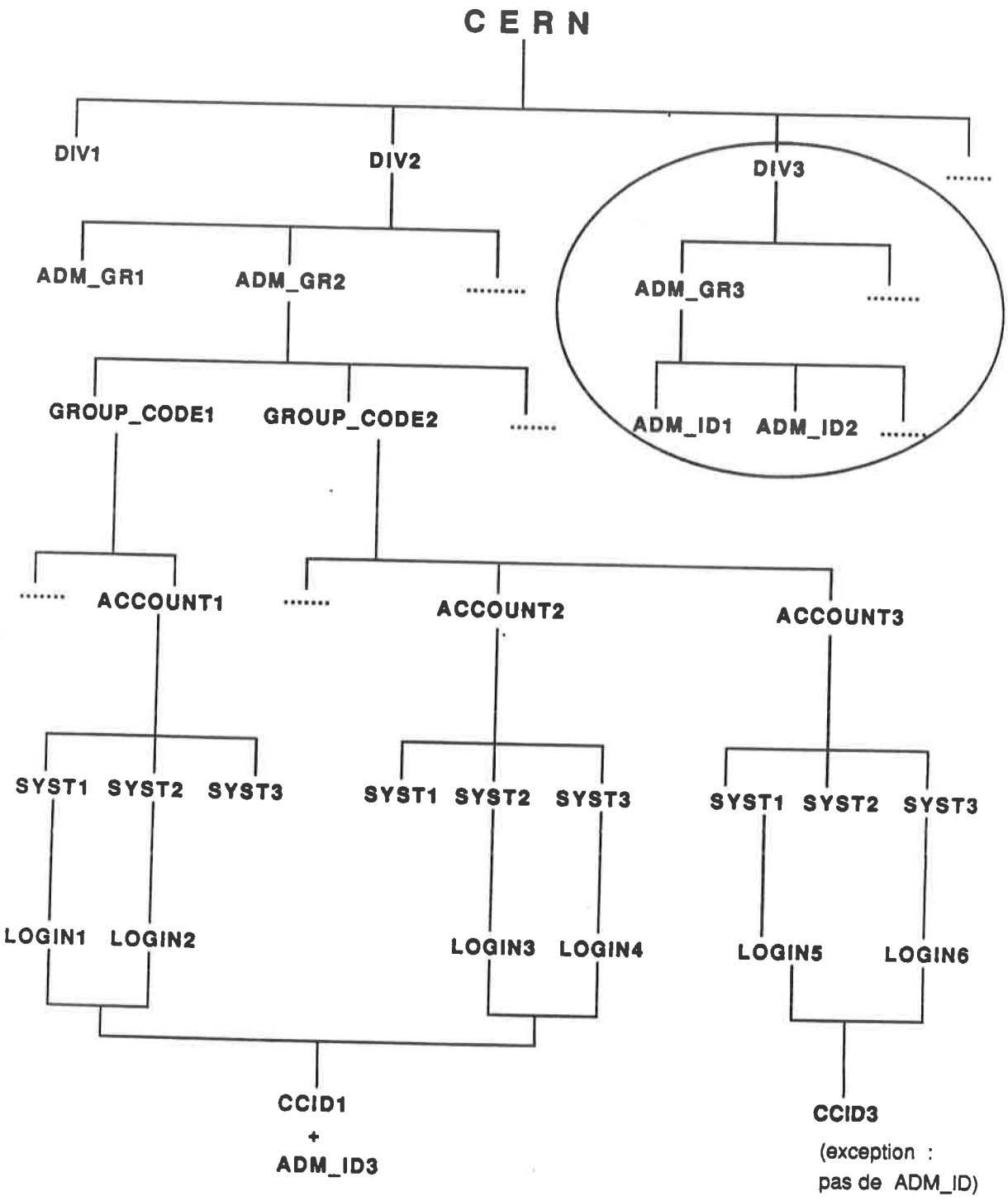

Fig. 5. Organigramme du CERN

Légende :

DIV = division du CERN
ADM_GR = groupe du point de vue administratif
ADM_ID = identificateur administratif
d'une personne travaillant au CERN
GROUP_CODE = groupe d'attribution des ressorces d'un
système d'exploitation à un utilisateur.
ACCOUNT = ressource attribué par l'administrateur d'un système
aux utilisateurs, pour qu'ils puissent
travailler sur les ordinateurs
SYST = système d'exploitation d'un ordinateur
LOGIN = clé d'accès pour un système d'un ordinateur
CCID = identificateur d'une personne travaillant sur
un ordinateur

Pour les niveaux supérieurs, la représentation est assez évidente : le CERN est constitué de plusieurs divisions, décomposées en plusieurs groupes dans lesquels se trouvent plusieurs personnes (une personne appartient à un et un seul groupe administratif).

Ce qui apparaît dans le cercle est la hiérarchie pour une personne travaillant au CERN, n'utilisant pas les ordinateurs avec les systèmes représentés plus bas (il peut toujours se servir d'un PC, mais ce fait restera inconnu par le centre de calcul).

C'est un environnement qui devient complexe lorsque nous descendons dans l'arbre, au niveau des utilisateurs d'ordinateurs.
Sous un groupe administratif (adm_gr) on trouve plusieurs sous groupes d'attribution des ressources (group_code) ayant ou non l'autorisation d'utiliser un système donné.
Un "account" a le format UUU−GG où "U" représente un caractére de l'identificateur d'un utilisateur d'un système d'exploitation (user_code) et "G" représente un caractère d'un sous groupe d'allocation des ressources (group_code).
Un utilisateur peut avoir plusieurs "user_code" dans le cas où il veut obtenir plusieurs "accounts" sous le même "group_code". Mais il pourrait aussi obtenir plusieurs "accounts" en ayant plusieurs "group_code" et en étant identifié par le même "user_code". L'"account" permet d'obtenir des "login" sur différents systèmes (un login avec le mot de passe associé, est la clé d'accès pour un seul système). Pour le système MVS la partie "user_code" de l'"account" est confondue avec le "login", ce qui n'est pas le cas pour VMS et VM, où l'on peut différencier le nom du "login". Un utilisateur peut donc obtenir un nombre d'"account" égal à la multiplication du nombre de "groupe_code" et le nombre de "user_code" dont il dispose. Ainsi il peut obtenir un nombre de "login" égal au résultat de la multiplication du nombre d'"account" qu'il possède et du nombre de systèmes sur lequel il peut travailler.

Les règles énoncés ici ne sont qu'une partie de celle qui existent en réalité, mais on en verra plus en détail dans les prédicats des relations définies plus tard.

4.2 Procédé d'analyse

La manière dont on a mené l'analyse peut être exposée par l'algorithme qui suit :

Faire 1) et 2) tant que le "client" n'est pas satisfait.

1)

a) Liste des informations de bases fournies par UCO.

b) Structurer l'information.

c) Nommer les relations et les constituants mnémoniquement.

d) Chercher les clés.

e) Restructurer l'information.
(ajouter, enlever un constituant d'une relation,
ajouter, enlever une relation)

f) décider des nouvelles clés.

2)

g) Se mettre en contact avec "le "client"" pour voir s'il approuve la structure.

h) – si le "client" est satisfait, alors fin de l'algorithme
– sinon passer à l'étape suivante.

i) Le "client" modifie et/ou complète la structure selon ses critères (qui ne sont pas ceux de l'optimisation de la Base !)

l) Retour dans 1) et effectuer les mêmes opérations, sauf c)

Or, ce qu'il aurait fallu faire pour minimiser les règles d'intégrités à écrire dans l'application, c'est le procédé suivant :

* Chercher les dépendances fonctionnels.
* Faire un réseau de noeuds et d'étoiles.
* Voir quelles sont les dépendances fonctionnelles élémentaires, puis obligatoires pour trouver une Base irrédondante.
* Faire une décomposition à partir de la Base irrédondante et construire le graphe à partir de cette décomposition.

La raison pour laquelle nous n'avons pas pu appliquer ce dernier procédé, est que l'on a du construire une Base qui devait être adapté aux systèmes existants car il n'était pas consentis de construire une Base et d'en adapter ensuite l'environnement. En effet, la contrainte était posée par les fichiers préformatés contenant les données qui devaient remplir la Base. L'exigence des relations statistiques et de budget était que celle-ci fournissent des resultats clairs pour les utilisateurs et "non encombrants" pour la Base.
En admettant que cette contrainte n'existe pas, on aurait pu construire une Base de Données avec moins de relations et des vues sur les informations les plus intéressantes. Par exemple, on aurait pu regrouper presque toutes les statistiques dans une seule table et faire ensuite des vues "système", "groupe" et "utilisateur"...

4.3 Définition des constituants

Tout d'abord, il vaut mieux définir quelques termes à propos de la définition des constituants qui sont classés selon l'ordre d'apparition dans les relations définies après.

– Les types Oracle déclarés sont CHAR, NUMBER, DATE mais ici on utilisera des termes plus précis mot, texte, entier, réel, date.

– La longueur correspond à celle du champs, un réel de longueur 7,1 signifie qu'il est composé de sept chiffres, dont un décimal.

– "Null" signifie que la valeur du constituant peut être absente (champs vide). Remarque : avec Oracle on déclare le contraire, c'est-à-dire, NOT NULL (valeur à insérer obligatoirement, lorsqu'on remplit une table).

– Le domaine correspond à une table à une seule colonne, pour la validation des valeurs qui entrent dans les autres tables de la base.

– Le terme "personne" est à entendre au sens du droit, c'est-à-dire qu'elle représente la personne "physique" (l'individu) ou la personne "morale" (l'Entreprise).

ALLOC_CPU1 : allocation de temps CPU du premier trimestre.

Type : réel longueur : 7,1

ALLOC_CPU2 : allocation de temps CPU du deuxième trimestre.

Type : réel longueur : 7,1

ALLOC_CPU3 : allocation de temps CPU du troisième trimestre.

Type : réel longueur : 7,1

ALLOC_CPU4 : allocation de temps CPU du quatrième trimestre.

Type : réel longueur : 7,1

ALLOC_LIM_DATE : date limite d'allocation.

Type : date

AUTHORIZE : nom de la personne autorisant la requête ou le changement.

Type : texte longueur : 20

BAT_SCPU : temps CPU en batch utilisé par semaine.

Type : réel longueur : 7,2

BEEP_NUM : numéro (entouré de "<" et de ">") de l'appareil de recherche du personnel.

Type : texte longueur : 6 Null

BIRTHDATE : date de naissance (l'année a été supprimée car c'est une information très confidentielle!

Impossible de le déclarer date).

Type : texte longueur : 5

BUILD : numéro de bâtiment (précédé par un "B") dans lequel se situe le bureau de la personne.

Type : mot longueur : 4 Null
Domaine : (B001,B010,B100,...)

CASS : nombre de cassettes montées par semaine.

Type : entier longueur : 5

CCID : numéro d'identification pour un utilisateur d'ordinateurs.

Type : entier longueur : 6

CCID_CP : numéro d'identification pour le chef responsable des ordinateurs.

Type : entier longueur : 6 Null

CCID_RESP : numéro d'identification de la personne qui est responsable d'une ou plusieurs personnes appartenant à la catégorie NH.

Type : entier longueur : 6 Null

CERN_GROUP : identificateur d'un groupe administratif.

Type : mot longueur : 3
Domaine : (AM,TIS,VT,...)

CERN_ID : identificateur pour une personne travaillant au CERN.

Type : entier longueur : 5

CHANGER_CCID : numéro d'identification de la personne qui opère le changement sur la relation.

Type : entier longueur : 6

CHGE_DATE : date du dernier changement sur les données de la relation.

Type : date

CNL_COPIES : nombre de copies du bulletin interne à envoyer à la personne.

Type : entier longueur : 3

CONT_PERS : chef (ou entreprise) responsable des ordinateurs du groupe.

Type : texte longueur : 30

CRAY_USE : autorisation d'utiliser le système de CRAY.

Type : mot longueur : 1
Domaine : (Y,N)

CUR_ALLOC_CPU : allocation actuelle totale du temps CPU.

Type : réel longueur : 7,1

CUR_FS_BUD : allocation budgétaire actuelle totale en francs suisses.

Type : réel longueur : 10,2

DIVISION : identificateur d'une division du CERN.

Type : mot longueur : 3
Domaine : (DD,EP,LEP,...)

FIRST_NAME : prénom d'une personne.

Type : texte longueur : 15

FLR_OFFICE : étage, aile et/ou numéro de bureau dans un bâtiment.

Type : mot longueur : 4 Null
Domaine : (R002,RC28,1010,S009,2C21,...)

GROUP_CODE : code d'identification d'un groupe d'allocation des ressources.

Type : mot longueur : 2
Domaine : (V1,VP,VS,SI,KA,...)

HOME_INST : description de la provenance de la personne (ex : université de Genève).

Type : texte longueur : 80 Null

INT_SCPU : temps CPU en interactif utilisé par semaine.

Type : réel longueur : 7,2

J_CLASS_DEF : définition de la classe d'un "job" (programme lancé sur un système).

Type : texte longueur : 15

LAST_DET_WEEK : numéro de semaine de l'année correspondant à la période où le dernier accès avec le login a été détecté.

Type : entier longueur : 6

LIM_DATE : date d'échéance du contrat avec le CERN.

Type : date Null

LOGIN_ID : "user_name" (login) servant à interagir avec un système.

Type : texte longueur : 20

MATCH_DATE : date de dernière vérification de la relation USER_DEF avec la relation PHONE_BOOK.

Type : date Null

MATCH_FLAG : identificateur permettant de connaître le résultat de la comparaison de la relation USER_DEF avec la relation PHONE_BOOK.

Type : entier longueur : 2

Domaine :
(−1 = les relations correspondent, mais la personne dont il s'agit a quitté définitivement le CERN,
0 = les données des deux relations sur la personne ne correspondent pas parfaitement,
1 = les relations correspondent, la personne est encore au CERN)

MINT_PRIME : moyenne hebdomadaire du temps interactif par rapport à la période du "prime shift" (de 9h00 à 17h00) où il y a la plus forte présence d'utilisateurs sur un système.

Type : réel longueur : 7,2 Null

MLOGIN_BUSY : moyenne hebdomadaire du nombre de différents logins fonctionnant sur un système, en période de forte occupation des ressources.

Type : entier longueur : 5

MLOGIN_QUIET : moyenne hebdomadaire du nombre de différents logins actifs sur un système, en période de faible occupation des ressources.

Type : entier longueur : 5

MRESP_BUSY : moyenne hebdomadaire exprimée en milli-secondes, sur le temps de réponse d'un système, en période de forte occupation des ressources.

Type : réel longueur : 5,1 Null

MRESP_QUIET : moyenne hebdomadaire exprimée en milli-secondes, sur le temps de réponse d'un système, en période de faible occupation des ressources.

Type : réel longueur : 5,1 Null

MVS_USE : autorisation d'utiliser le système MVS.

Type : mot longueur : 1
Domaine : (Y,N)

NATIONALITY : nationalité de la personne concernée.

Type : mot longueur : 2 Null

Domaine : (CH,FR,IT,GB,...)

NUM_J_PRT : nombre de "jobs" imprimés par semaine sur une imprimante d'une catégorie donnée.

Type : entier longueur : 5

NUM_J_RUN : nombre de "jobs" exécutés en une semaine sur un système.

Type : entier longueur : 5

NUM_LOGIN : nombre de logins différents utilisés par semaine, sur un système.

Type : entier longueur : 5

NUM_L_PRT : nombre de lignes imprimés par semaine par une imprimante sur une catégorie donnée.

Type : entier longueur : 7

OLD_CCID : numéro d'identification de l'utilisateur qui possédait antérieurement l'"account" qui est actuellement en possession de la personne correspondant à la colonne CCID.

Type : entier longueur : 6

PERC_J_TT : pourcentage de "jobs" exécutés en "batch" (traitement par lots).

Type : entier longueur : 3

PHONE_FLAG : indicateur de l'état d'une rangée de la relation PHONE_BOOK.

Type : mot longueur : 1 Null
Domaine : (C = l'information dans la rangée a été changée depuis la dernière mise à jour,
D = l'information de cette rangée concernant une
personne ayant quitté le CERN est conservée
dans la base malgré qu'elle soit effacée
de l'annuaire téléphonique sur support papier,
A = la personne qui avait quitté le CERN est revenue
et doit réapparaître dans l'annuaire téléphonique

sur support papier,
N = la rangée contenant l'information d'une personne arrivée au CERN n'a jamais existé dans la relation PHONE_BOOK)

PHONE_NUM1 : premier numéro de téléphone associé à la personne.

Type : entier longueur : 4 Null

PHONE_NUM2 : deuxième numéro de téléphone associé à la personne.

Type : entier longueur : 4 Null

PHONE_NUM3 : troisième numéro de téléphone associé à la personne.

Type : entier longueur : 4 Null

PREF_NAME : prénom "préféré" par une personne.

Type : texte longueur : 15

PRIV_CODE : privilège d'accès aux informations contenues dans la base.

Type : mot longueur : 1
Domaine : (0 - - > NU = utilisateur normal
1 - - > GA = administrateur d'un groupe,
2 - - > SA = administrateur d'un système,
3 - - > RM = manager des ressources d'un ordinateur)

PROJ_HEAD : chef de groupe et responsable de projet.

Type : texte longueur : 30

PROJECT : description du projet.

Type : texte longueur : 80

PRT_CLASS : identification de la catégorie de l'imprimante.

Type : mot longueur : 1
Domaine : (F,A,W)

PRT_LOC : identification du lieu où se trouve l'imprimante.

Type : texte longueur : 20

PRT_NAME : identification de l'imprimante par son nom.

Type : texte longueur : 20

PRT_TYPE : identification du type de l'imprimante.

Type : texte longueur : 20

REASON_CHGE : description de la raison du changement d'allocation.

Type : texte longueur : 80

REG_DATE : date d'arrivée et d'enregistrement d'une personne au CERN.

Type : date

REQ_CHGE_DATE : date de nouvelle requête ou de changement de requête.

Type : date

REQUEST : requête de ressources effectuée ou non.

Type : mot longueur : 1
Domaine : (Y,N)

REQ_CPU1 : requête de temps CPU du premier trimestre.

Type : réel longueur : 7,1 Null

REQ_CPU2 : requête de temps CPU du deuxième trimestre.

Type : réel longueur : 7,1 Null

REQ_CPU3 : requête de temps CPU du troisième trimestre.

Type : réel longueur : 7,1 Null

REQ_CPU4 : requête de temps CPU du quatrième trimestre.

Type : réel longueur : 7,1 Null

SIGN : réponse à la question : la personne a-t-elle signé les documents légaux ?

Type : mot longueur : 1
Domaine : (Y,N)

STATUS : fonction remplie au CERN par la personne.

Type : mot longueur : 4 Null
Domaine : (STAFF,USSA,UPAS,...)

STATUS_LOG : état actuel du login d'une personne.

Type : mot longueur : 15
Domaine : (ACTIVE,DEACTIVATED,NOLOG,RETIRED,...)

SURNAME : nom d'une personne connue par l'administration du CERN.

Type : texte longueur : 24

SYSTEM : système d'exploitation d'un ordinateur.

Type : mot longueur : 6
Domaine : (VMCMS,MVS,VAXVMS...)

TAPES : nombre de bandes montées par semaine.

Type : entier longueur : 5

TOT_EOC : total du temps CPU estimé, utilisé à l'extérieur du CERN (indépendamment des systèmes du CERN).

Type : réel longueur : 7,1

TOT_WAIT : temps d'attente total pour un système pour une semaine donnée.

Type : réel longueur : 7,2

TURN_TIME : limite de temps.

Type : entier longueur : 4

USED_DISK : espace disque utilisé actuellement par un groupe sur un système donné.

Type : réel longueur : 8,1 Null

USED_SF : part du budget en francs suisses dépensé jusqu'à aujourd'hui par un groupe sur un système donné.

Type : réel longueur : 10,2

USER_CLASS : catégorie dans laquelle une personne est reconnue.

Type : mot longueur : 2
Domaine : (GU = personnel interne au CERN,
OU = personnel externe au CERN,
SS = étudiant d'été,
NH = Entreprise travaillant en sous traitance pour le CERN).

USER_CODE : identificateur d'une personne travaillant avec les ordinateurs.

Type : texte longueur : 3

VAXVMS_USE : autorisation d'utiliser le système VMS.

Type : mot longueur : 1
Domaine : (Y,N)

VERSION : numéro de version (1 = original) pour les allocations de budget.

Type : entier longueur : 2

VMCMS_USE : autorisation d'utiliser le système VMCMS.

Type : mot longueur : 1
Domaine : (Y,N)

WEEK : numéro de semaine.
(codé de la manière suivante :
les deux premiers chiffres représentent l'année,
les deux derniers chiffres représentent le numéro de semaine)

Type : entier longueur : 4

YEAR_NUM : année (exemple: 1987).

Type : entier longueur : 4

Il est à noter que l'on n'a pas toujours donné deux noms différents à des constituants qui ont une signification légèrement différente d'après leur contexte (relations avec des clés différentes), car lors des requêtes SQL, il n'est pas aisé de se souvenir de noms différenciés. On peut de toute façon marquer la différence en prefixant le constituant avec le nom de la relation.

4.4 Définition des relations

Chaque fois qu'on rencontre dans les predicats le mot "groupe", celui-ci désigne le groupe d'allocation des ressources (group_code).

GROUP_DEF : définition d'un groupe.

(GROUP_CODE, PROJ_HEAD, CONT_PERS, CCID, PROJECT, MVS_USE,
VMSCMS_USE, VAXVMS_USE, CRAY_USE)

Prédicat : A chaque groupe, identifié par un code, est associé un chef, une personne responsable des ordinateurs et son CCID qui l'identifie. On lui associe également une description du(des) projet(s) sur lequel il travaille et l'autorisation d'utiliser les divers systèmes d'exploitation MVS, VMCMS, VMS, CRAY qui est un booléen de valeur "Y" si le groupe à l'autorisation d'utiliser le système et "N" s'il ne l'a pas.

clé : GROUP_CODE

BUD_ALLOC : budgets alloués aux groupes pendant l'année courante.

(GROUP_CODE, SYSTEM, YEAR_NUM, REQUEST, REQ_CPU1, REQ_CPU2,
REQ_CPU3, REQ_CPU4, ALLOC_CPU1, ALLOC_CPU2, ALLOC_CPU3, ALLOC_CPU4,
CUR_ALLOC_CPU, CUR_FS_BUD, ALLOC_LIM_DATE)

Prédicat : A chaque groupe, pour un système et une année donnés on associe un booléen de valeur "Y" si le groupe a l'autorisation d'utiliser le système et "N" s'il ne l'a pas. En outre on mentionne les requêtes et les allocations de temps CPU pour chaque trimestre, ainsi que d'autres informations telles que les allocations actuelles en temps CPU et en francs suisses et la date limite d'allocation.

clé : GROUP_CODE SYSTEM YEAR_NUM

BUD_HIST : historique des budgets de groupe.

(GROUP_CODE, SYSTEM, VERSION, YEAR_NUM, REQUEST, REASON_CHGE,
REQ_CPU1, REQ_CPU2, REQ_CPU3, REQ_CPU4, ALLOC_CPU1, ALLOC_CPU2,
ALLOC_CPU3, ALLOC_CPU4, REQ_CHGE_DATE, AUTHORIZE)

Prédicat : A chaque groupe, pour un système, une version et une année donnés, on associe un booléen de valeur "Y" si le groupe a l'autorisation d'utiliser le système et "N" s'il ne l'a pas, et la (les) raison(s) du changement. En outre on mentionne les requêtes et les allocations de temps CPU pour chaque trimestre, ainsi que d'autres informations telles que la date de nouvelle (ou changement de) requête, la personne autorisant le changement, ainsi que la consommation du budget en temps CPU et francs suisses.

clé : GROUP_CODE SYSTEM VERSION YEAR_NUM

GROUP_DATA : information budgétaire du groupe, indépendamment du système.

(GROUP_CODE, VERSION, YEAR_NUM, TOT_EOC)

Prédicat : pour un groupe, une version et une année données, on associe le temps CPU total estimé utilisé à l'extérieur du CERN.

clé : GROUP_CODE VERSION YEAR_NUM

PHONE_BOOK : annuaire téléphonique du CERN.

(CERN_ID, SURNAME, FIRST_NAME, BIRTHDATE, NATIONALITY,
_ _ _ _ _
DIVISION, CERN_GROUP, STATUS, PHONE_NUM1, PHONE_NUM2, PHONE_NUM3,

BEEP_NUM, BUILD, FLR_OFFICE, CCID, PHONE_FLAG, CHGE_DATE, MATCH_DATE)

Prédicat : à chaque personne connue par l'administration, identifié par un CERN_ID, on associe son nom, prénom, date de naissance, nationalité, division et son code de groupe administratif. De plus, d'autres informations utiles situent la personne telles que sa fonction, ses trois éventuels numéros de téléphone, son numéro d'appareil de recherche, le bâtiment dans lequel se trouve son bureau, à un certain étage. Des informations concernant la mise à jour et validation de la relation sont aussi nécessaires : l'éventuel CCID (si la personne utilise un ordinateur), un indicateur permettant de connaître l'état d'une rangée (changée, à supprimer, nouvelle), la date de dernier changement, et la date de vérification avec les données de la relation USER_DEF.

clé : CERN_ID

USER_DEF : définition d'un utilisateur

(CCID, SURNAME, FIRST_NAME, PREF_NAME, BIRTHDATE, DIVISION,
_ _ _
HOME_INST, SIGN, REG_DATE, LIM_DATE, CNL_COPIES, USER_CLASS,

CCID_RESP, MATCH_FLAG, CHGE_DATE, CHANGER_CCID)

Prédicat : à chaque utilisateur d'ordinateur identifié par un CCID unique, on associe son nom, prénom, prénom préféré, date de naissance, la division. En plus, on a d'autres informations administratives telles que son institut de provenance, la signature des contrats (booléen 'Y','N'), la date d'enregistrement et la date de fin de contrat s'il en existe une, le nombre de copies du bulletin interne et la catégorie à l'intérieur du CERN. Si la catégorie est différente de GU (Général User), on doit lui associer le numéro d'identification de la personne qui est responsable. Des informations concernant la validation et la mise à jour des données de la relation sont nécessaires : on a un indicateur permettant de

savoir si les données de cette relation sont consistantes, la date de dernière mise à jour et le numéro d'identification de la personne qui a fait le changement.

clé : CCID

LOGID_DEF : définition d'un login.

(LOGIN_ID, USER_CODE, GROUP_CODE, SYSTEM, REG_DATE, PRIV_CODE,
STATUS_LOG, CCID, OLD_CCID, LAST_DET_WEEK, CHANGER_CCID, CHGE_DATE)

Prédicat : pour chaque utilisateur appartenant à un groupe ayant un login d'accès et un identificateur dans un système donné, on associe une seule date d'enregistrement, un code de privilège sur la Base (user BDORAC), l'information sur l'état du login. En outre, on associe un et un seul CCID, ainsi que le CCID original (information nécessaire pour les logins qui peuvent être transférés, à défaut on copiera le CCID actuel) et la date où le dernier accès a été détecté. S'il y a mise à jour de la relation on a le numéro d'identification de la personne et la date qui s'y rapporte.

clé : LOGIN_ID USER_CODE GROUP_CODE SYSTEM

GROUP_STAT : statistique hebdomadaire sur le groupe.

(GROUP_CODE, SYSTEM, WEEK, INT_SCPU, BAT_SCPU, TAPES, CASS,
USED_DISK, USED_SF)

Prédicat : pour un groupe, un système et une semaine données, on associe des statistiques concernant les temps CPU interactif et batch, les bandes, les cassettes, l'espace disque et le budget utilisé.

clé : GROUP_CODE SYSTEM WEEK

LOGID_STAT : statistique hebdomadaire sur le login.

(LOGIN_ID, USER_CODE, GROUP_CODE, SYSTEM, WEEK, INT_SCPU, BAT_SCPU,

LOGID_STAT : statistique hebdomadaire sur le login.

(LOGIN_ID, USER_CODE, GROUP_CODE, SYSTEM, WEEK, INT_SCPU, BAT_SCPU,
TAPES, CASS)

Prédicat : pour chaque utilisateur appartenant à un groupe ayant un login d'accès et un identificateur dans un système, pour une semaine donnée on associe les statistiques concernant les temps CPU en interactif et en batch, les bandes et les cassettes.

clé : LOGIN_ID USER_CODE GROUP_CODE SYSTEM WEEK

SYST_STAT : statistique hebdomadaire des systèmes

(SYSTEM, WEEK, BAT_SCPU, INT_SCPU, TOT_WAIT, NUM_LOGIN, TAPES, CASS,
MRESP_BUSY, MRESP_QUIET, MLOGIN_BUSY, MLOGIN_QUIET, MINT_PRIME)

Prédicat : pour chaque système à une semaine donnée, on associe les statistiques sur les temps CPU en batch et en interactif, le temps total d'attente, le nombre de login ayant été connectés, les bandes et les cassettes. On tient compte également sur certaines périodes (chargés et non chargées), des moyennes concernant les temps de réponse et le nombre de login, en plus de la moyenne sur le temps en interactif.

clé : SYSTEM WEEK

BJ_STAT : statistiques sur les "batch jobs"

(J_CLASS_DEF, SYSTEM, WEEK, NUM_J_RUN)

Prédicat : à une classe de "job" d'un système pendant une semaine donnée, on associe un nombre déterminé de "job" exécutés.

clé : J_CLASS_DEF SYSTEM WEEK

BJ_TURNSTAT : statistique sur les "batch jobs" exécutés dans une certaine limite de temps

(J_CLASS_DEF, TURN_TIME, SYSTEM, WEEK, PERC_J_TT)

Prédicat : à une classe de "job" exécuté dans une limite de temps sur un système, pendant une semaine donnée, on associe un nombre déterminé de "job" exécutés.

clé : J_CLASS_DEF TURN_TIME SYSTEM WEEK

PRINT_STAT : statistique sur les imprimantes

(PRT_NAME, PRT_CLASS, WEEK, PRT_TYPE, PRT_LOC, NUM_J_PRT, NUM_L_PRT)

Prédicat : pour une imprimante d'une certaine catégorie à une semaine donnée, on associe un type d'imprimante, une localisation, un nombre de job et de lignes imprimés.

clé : PRT_NAME PRT_CLASS WEEK

En général, dans les relations concernant les statistiques et la relation historique les rangées ne sont jamais modifiées ou supprimées.
Chaque année ces relations vont être vidées pour éviter une trop grande cumulation d'information peu utile. Ces dernières sont quand même gardées sur un support magnétique à part d'où l'on pourra toujours les recharger si besoin est.

Les relations concernant les budgets et les utilisateurs sont permanentes et les mises à jour y sont possibles. On verra en détail, dans la procédure de gestion des utilisateurs quelles sont les possibilités offertes et les restriction sur l'accès à ces informations.

4.5 Graphe de relation

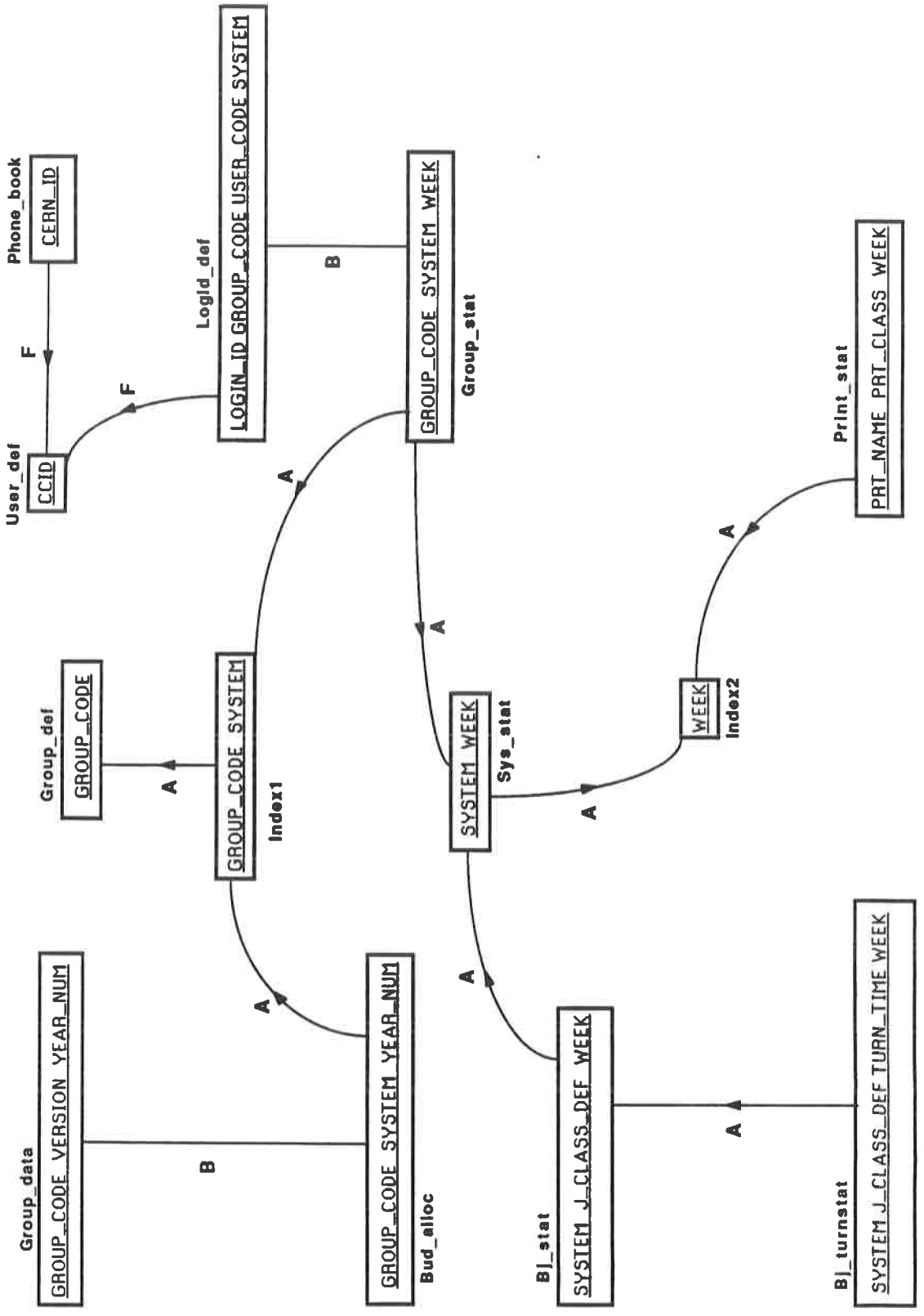

Fig. 6. Graphe de relation

Le graphe de relation (cf page précédente) est obtenu à partir de la décomposition sur les relations provenant de l'analyse en se servant de l'algorithme de modélisation n-aire du cours.

On peut faire les commentaires suivants.

– Ce graphe ne contient pas de cycles, on ne rencontre pas de problèmes d'etanchéité.

– Les deux B-arètes "cachent" respectivement la relation BUD_HIST (historique des budgets de groupe) et la relation LOGID_STAT (statistique sur les users d'un système).

– Les charnières n'ont pas été totalement éliminées, il en reste deux (INDEX1 et INDEX2). La première est est une vue sur les systèmes utilisés par chaque groupe, on aurait pu l'éliminer en ajoutant le constituant SYSTEM à la relation GROUP_DEF, mais les fichiers d'insertion des données ne le prévoyant pas, il est impossible de le faire.

La deuxième est une vue de la semaine de l'année et elle est nécessaire pour que la relation des statistiques sur les imprimantes (PRINT_STAT) ne reste pas isolée (afin d'avoir un graphe connexe). On aurait pu avoir une autre solution avec le constituant SYSTEM dans PRINT_STAT, mais on ne peut pas avoir de données indiquant quelle imprimante est en relation avec tel système.

– Certaines redondances existent dans notre base :
au niveau de la relation BUD_HIST qui n'est rien d'autre qu'un historique de la relation BUD_ALLOC (pour les requêtes et les allocations en temps CPU pour chaque trimestre de l'année).
Les dispositions dans la même rangée des resquêtes avec les allocations d'une part, et des quattres trimestres en même temps d'autre part, ne sont pas très "normales", car il faudrait normalement avoir des rangées trimestrielles, d'un groupe sur un système pour une année donnés pour accéder plus facilement à l'information. Cela sert à donner une vision globale et résumée des budgets. Néanmoins, elles sont nécessaires par les caractéristiques mêmes du projet, les buts visé étant de présenter des résultats compactés, afin d'éviter l'accumulation de données inutiles dans les tables (économie de la place memoire...)
Par conséquent, il faut accepter de valider des règles d'intégrités supplémentaires qui n'existeraient pas sans la redondance.

4.6 Règles d'intégrités

Ces règles d'integrités sont validées seulement lors des accès des administrateurs de système (SA) et des manages de ressource (RM). Les autres règles validées au niveau de la procédure se trouvent dans la section 6.3.3. : les triggers. Leur expression n'est plus

1) **Libellé :** La date d'enregistrement au CERN est antérieure ou égale à la date limite.

Contexte : USER_DEF

Expression :

Pour toute entité ud de USER_DEF
faire
ud.REG_DATE < = ud.LIM_DATE
refaire

Portée :

relation	création	mise à jour	suppression
USER_DEF	oui	ud.REG_DATE ud.LIM_DATE	non

2) **Libellé :** Si au moins une requête de la part d'un groupe sur un système donné a eut lieu, c'est à dire qu'au moins un des champs : REQ_CPU1, REQ_CPU2, REQ_CPU3, REQ_CPU4 n'est pas vide, alors la valeur du constituant "REQUEST" sera égale a "Y".

Contexte : BUD_ALLOC**Expression :**

```
Pour toute entité ba de BUD_ALLOC
  faire
    si ba.REQ_CPU1 < > nil ou
       ba.REQ_CPU2 < > nil ou
       ba.REQ_CPU3 < > nil ou
       ba.REQ_CPU4 < > nil alors
       ba.REQUEST = "Y"
    finsi
  refaire
```

Portée :

relation	création	mise à jour	suppression
BUD_ALLOC	oui	ba.REQ_CPU1 ba.REQ_CPU2 ba.REQ_CPU3 ba.REQ_CPU4	non

3) **Libellé :** La sommation des allocations en temps CPU des quatre trimestres pour un groupe sur un système donné donne la valeur du constituant CUR_ALLOC_CPU.

Contexte : BUD_ALLOC

Expression :

Pour toute entité ba de BUD_ALLOC faire
CUR_ALLOC_CPU := ALLOC_CPU1 + ALLOC_CPU2
ALLOC_CPU3 + ALLOC_CPU4
refaire

Portée :

relation	création	mise à jour	suppression
BUD_ALLOC	oui	ba.ALLOC_CPU1 ba.ALLOC_CPU2 ba.ALLOC_CPU3 ba.ALLOC_CPU4	non

4) **Libellé :** Le budget de consommation totale de francs suisses (CUR_ALLOC_CPU) est égale à l'allocation totale de temps CPU multiplié par une taxe fixe de 250.– francs.

Contexte : BUD_ALLOC

Expression :

```
Pour toute entité ba de BUD_ALLOC
  faire
    FS_BUD_TOT := CUR_ALLOC_CPU * 250
  refaire
```

Portée :

relation	création	mise à jour	suppression
BUD_ALLOC	oui	ba.CUR_ALLOC_CPU ba.FS_BUD_TOT	non

5) **Libellé :** Quand le numéro de version des budgets historiques est égale à "1" c'est alors une allocation initiale

Contexte : BUD_HIST

Expression :

```
Pour toute entité bh de BUD_HIST
faire
  si bh.VERSION = 1 alors
    bh.REASON_CHANGE = "INITIAL ALLOCATION"
refaire
```

Portée :

relation	création	mise à jour	suppression
BUD_HIST	oui	ba.REASON_CHANGE	non

6) **Libellé :** Un utilisateur au CERN appartenant à une catégorie "NH" ou "SS" (dans USER_DEF) doit obligatoirement avoir une personne qui en est responsable. Le groupe et le système de cet utilisateur doivent correspondre à ceux du responsable.

Contexte : USER_DEF * LOGID_DEF

Expression :

```
Pour toute entité ud de la relation USER_DEF
faire
  si ud.USER_CLASS = 'GU' alors ud.CCID_RESP = Nil
  sinon si ud.USER_CLASS appartient à {NH,SS} alors
   il existe ld1, ld2 dans LOGID_DEF tel que
     ud.CCID = ld1.CCID et
     ud.CCID_RESP = ld2.CCID et
     ld1.SYSTEM = ld2.SYSTEM et
     ld1.GROUP_CODE = ld2.GROUP_CODE
  finsi
refaire
```

Portée :

relation	création	mise à jour	suppression
USER_DEF	oui	ud.USER_CLASS ud.CCID_RESP	non
LOGID_DEF	non	ld1.CCID ld2.CCID	oui

7) **Libellé** : Pour qu'un utilisateur puisse avoir un login sur un système donné, il faut que son groupe dispose de l'autorisation sur ce système.
Nous prendrons l'exemple du système MVS.

Contexte : LOGID_DEF * GROUP_DEF

Expression :

Pour toute entité ld de la relation LOGID_DEF
faire
si ld.SYSTEM = "MVS" alors
il existe une entité gd de GROUP_DEF tel que
ld.GROUP_CODE = gd.GROUP_CODE et
vérifier que gd.MVS_USE = "Y"
fsi
refaire

Portée :

relation	création	mise à jour	suppression
LOGID_DEF	oui	non	
GROUP_DEF	non	gd.MVS_USE	oui

Même contexte, schéma de programme et portée pour les constituants gd.VMCMS_USE, gd.VAXVMS_USE, gd.CRAY_USE

8)**Libellé :** Le login et le système sont suffisants pour désigner un et un seul CCID.

Contexte : LOGID_DEF

Expression :

Pour tout entité ld et ld' de LOGID_DEF faire
si ld.CCID = ld'.CCID alors
ld.SYSTEM < > ld'.SYSTEM et
ld.LOGIN_ID < > ld'.LOGIN_ID
Refaire

Portée :

relation	création	mise à jour	suppression
LOGID_DEF	oui	ld.CCID	non

9) **Libellé :** Pour chaque nouvelle requête et allocation effectuée dans BUD_ALLOC, il faudra créer une VERSION dans la table BUD_HIST de façon à garder un historique.
La création des entités de BUD_ALLOC et de BUD_HIST doit avoir lieu successivement et le numéro de version dans BUD_HIST sera égale à 1.
La mise à jour des requêtes dans BUD_ALLOC impliquera la création d'une entité dans BUD_HIST avec un nouveau numéro de version supplémentaire de façon à garder aussi un historique des changements.

Contexte : BUD_HIST * BUD_ALLOC

Expression :

Pour toute entité ba créée dans la relation BUD_ALLOC
faire
  créer une entité bh de BUD_HIST tel que
    bh.GROUP_CODE = ba.GROUP_CODE et
    bh.SYSTEM = ba.SYSTEM et
    bh.YEAR_NUM = ba.YEAR_NUM et
    bh.VERSION = 1 et
    bh.REQ_CPU(1..4) = ba.REQ_CPU(1..4) et
    bh.ALLOC_CPU(1..4) = ba.ALLOC_CPU(1..4) et
    bh.REQUEST = ba.REQUEST
refaire

Pour toute entité ba modifiée dans la relation BUD_ALLOC
faire
  créer une entité bh' de BUD_HIST tel que
    bh'.GROUP_CODE = bh.GROUP_CODE et
    bh'.SYSTEM = bh.SYSTEM et
    bh'.YEAR_NUM = bh.YEAR_NUM et
    bh'.VERSION = bh.VERSION + 1 et
    bh'.REQ_CPU(1..4) = ba.REQ_CPU(1..4) et
    bh'.ALLOC_CPU(1..4) = ba.ALLOC_CPU(1..4) et
    bh'.REQUEST = ba.REQUEST
refaire

Portée :

relation	création	mise à jour	suppression
BUD_HIST	non	non	
BUD_ALLOC	oui	ba.REQ_CPU(1..4) ba.ALLOC_CPU(1..4)	non

10)**Libellé** : Chaque fois que la table GROUP_DATA est révisée, il faut vérifier que le groupe d'une année donnée dont on veut mettre à jour le temps CPU extérieur existe ou non dans la table. S'il n'existe pas, on crée une nouvelle rangée avec une version égale à 1. S'il existe, on crée une nouvelle rangée avec le numéro de version incrémenté de 1.

Contexte : GROUP_DATA

Expression :

```
Soit gd' l'entité à insérer dans la table GROUP_DATA
Soit val_bol = faux (qui signifie entité gd non inséré)
Chercher l'entité gd de GROUP_DATA tel que
    gd'.GROUP_CODE = gd.GROUP_CODE
Tant que gd'.GROUP_CODE = gd.GROUP_CODE
    faire
    lire l'entité gd suivante
    refaire
insérer gd' après la dernière entité gd tel que
gd'.GROUP_CODE = gd.GROUP_CODE et
gd'.YEAR_NUM = "année"
gd'.VERSION = gd.VERSION + 1
gd'.TOT_EOC = "valeur"
val_bol = vrai
refaire

si val_bol = false alors insérer l'entité gd' après la dernière entité gd tel que
        gd'.GROUP_CODE = "nouveau groupe"
        gd'.YEAR_NUM = "année"
        gd'.VERSION = 1
        gd'.TOT_EOC = "valeur"
fsi
```

Portée :

relation	création	mise à jour	suppression
GROUP_DATA	oui	gd.TOT_EOC	non

11)**Libellé :** L'addition des temps CPU consommées en interactif et en batch de toutes les semaines de la même année jusqu'à aujourd'hui doit être inférieure à l'allocation courante totale de l'année du temps CPU.

Contexte : GROUP_STAT * BUD_ALLOC

Expression :

Soit TOT_BAT, TOT_INT, TOT trois variables locales
Pour toute entité ba de BUD_ALLOC
faire
TOT_BAT = 0; TOT_INT = 0; TOT = 0;
Pour toute entité gs de GROUP_STAT tel que
gs.GROUP_CODE = ba.GROUP_CODE et
gs.SYSTEM = ba.SYSTEM et
gs.WEEK inclue dans ba.YEAR_NUM
faire
TOT_BAT = TOT_BAT + gs.BAT_SCPU et
TOT_INT = TOT_INT + gs.INT_SCPU
refaire
TOT = TOT_BAT + TOT_INT
TOT < = ba.CUR_ALLOC_CPU
refaire

Portée :

relation	création	mise à jour	suppression
BUD_ALLOC	oui	ba.CUR_ALLOC_CPU	non
GROUP_STAT	oui	non	non

12)**Libellé :** L'addition des francs suisses dépensés par groupe pour toutes les semaines de la même année jusqu'à aujourd'hui doit être inférieure à l'allocation courante totale de l'année du budget en francs suisses.

Contexte : GROUP_STAT * BUD_ALLOC

Expression :

```
Soit TOT_FS une variable locale pour faire des additions
Pour toute entité ba de BUD_ALLOC
faire
TOT_FS = 0
    Pour toute entité gs de GROUP_STAT tel que
    gs.GROUP_CODE = ba.GROUP_CODE et
    gs.SYSTEM = ba.SYSTEM et
    gs.WEEK inclue dans ba.YEAR_NUM
    faire
        TOT_FS = TOT_FS + gs.USED_SF
    refaire
    TOT_FS < = ba.CUR_FS_BUD
refaire
```

Portée :

relation	création	mise à jour	suppression
BUD_ALLOC	oui	ba.CUR_FS_BUD	non
GROUP_STAT	oui	–	non

13)**Libellé :** La relation GROUP_STAT est obtenue en réalisant la somme dans LOGID_STAT des statistiques des utilisateurs par groupe, système et semaine.

Contexte : GROUP_STAT * LOGID_STAT

Expression :

Soient 4 variables locales
TOT_BAT = 0
TOT_INT = 0
TOT_TAPES = 0
TOT_CASS = 0
Pour toute entité gs de GROUP_STAT
faire
Pour toute entité ls de LOGID_STAT tel que
gs.GROUP_CODE = ls.GROUP_CODE et
gs.SYSTEM = ls.SYSTEM et
gs.WEEK = ls.WEEK
faire
TOT_BAT = TOT_BAT + ls.BAT_SCPU et
TOT_INT = TOT_INT + ls.INT_SCPU et
TOT_TAPES = TOT_TAPES + ls.TAPES et
TOT_CASS = TOT_CASS + ls.CASS
refaire
créer l'entité gs tel que
gs.INT_SCPU = TOT_INT et
gs.BAT_SCPU = TOT_BAT et
gs.TAPES = TOT_TAPES et
gs.CASS = TOT_CASS
refaire

Portée :

relation	création	mise à jour	suppression
LOGID_STAT	oui	–	non
GROUP_STAT	non	–	non

14)**Libellé :** Règles sur le CCID de la relation LOGID_DEF :
a) Lorsqu'un utilisateur est enregistré pour la première fois, le constituant OLD_CCID prend la même valeur que le CCID assigné.
b) Si le CCID assigné est temporaire, alors lorsqu'on veut attribuer le numéro d'identification définitif, les constituants CCID et OLD_CCID sont mis à jour avec la même valeur.
Sinon, lorsqu'un "account" change de propriétaire, la valeur du CCID est prise par OLD_CCID et CCID prend la valeur du nouveau propriétaire.

Contexte : LOGID_DEF

Expression :

a)

```
Pour toute entité ld crée de LOGID_DEF
faire
    ld.OLD_CCID = ld.CCID
refaire
```

b)

```
Pour toute entité ld mise à jour de LOGID_DEF
faire
    si ld.CCID est compris entre {200001..299999} alors
    ld.CCID = "CCID définitif" et
    ld.OLD_CCID = ld.CCID
    sinon
    ld.OLD_CCID = ld.CCID et
    ld.CCID = "nouveau propriétaire CCID"
    fsi
refaire
```

Portée :

relation	création	mise à jour	suppression
LOGID_DEF	oui a)	ld.CCID b)	non

15)**Libellé :** Il est interdit d'attribuer un CCID différent (lorsqu'on insère une nouvelle rangée, ou lorsqu'on le met à jour dans la relation LOGID_DEF) à un utilisateur qui reprend le même login sur le même système mais avec un "account" différent.

Contexte : LOGID_DEF

Expression :

Soit ld une entité de LOGID_DEF
Pour toute entité ld' de LOGID_DEF tel que
ld'.LOGIN_ID = ld.LOGIN_ID et
ld'.SYSTEM = ld.SYSTEM et
ld'.GROUP_CODE < > ld.GROUP_CODE et
ld'.USER_CODE < > ld.USER_CODE
faire
ld'.CCID = ld.CCID
refaire

Portée :

relation	création	mise à jour	suppression
LOGID_DEF	oui	ld.CCID	non

CHAPITRE 5
LES UTILITAIRES D'ORACLE.

Avant toutes choses, il est préférable de définir quelques mots clés du vocabulaire Oracle qui diffèrent de ceux utilisés en cours :

- une table = une relation, c'est-à-dire un ensemble de données structuré de manière logique, chaque donnée ayant un lien avec les autres données de la relation.
- une rangée = un n−uplet, c'est-à-dire un ensemble de valeurs liées par la relation.
- un champ = une des valeurs d'une rangée.

5.1 SQLPLUS.

Cet utilitaire constitue la base d'Oracle. Ce sont en fait des programmes qui lisent de manière interactive les requêtes SQL effectuées par l'utilisateur et interprètent ces requêtes. Si la syntaxe de la requête est correcte, le résultat s'affiche à l'écran avec une présentation agréable. Cette présentation peut être encore améliorée par l'utilisateur à l'aide de commandes dont la portée est la durée de la session SQLPLUS. Il est invoqué avec la commande :

SQLPLUS nom_user_oracle/mot_de_passe

Les requêtes possibles sur les tables peuvent être classées en quatre catégories :

- Celles qui travaillent sur les données rangées dans les tables :

  * select : consultation de données.
  * insert : insertion de nouvelles rangées dans la table.
  * delete : suppression de rangées dans la table.
  * update : modification de certains champs de certaines rangées.

- Celles qui travaillent sur la structure des tables :

  * create table : création de table.
  * drop table : suppression de table.
  * alter table : modification de la structure d'une table (ajout d'une nouvelle colonne, agrandissement de la taille d'une colonne) ou changement du nom d'une table ou d'une colonne.

Remarque : on ne peut pas "diminuer" la structure d'une table...

– Celles qui accélèrent le temps d'exécution des requêtes :

* create (drop) {unique} index : création (resp. suppression) d'une table d'indexes sur une ou plusieurs colonnes de la table. L'option "unique" permet de contrôler lors de la création de l'indexe, puis par la suite lors d'insertion de nouvelles rangées, l'unicité des valeurs de cette ou ces colonnes.

* create cluster : permet de ranger dans des zones contigues de la mémoire physique les rangées de tables différentes ayant des valeurs identiques dans certaines de leurs colonnes (paramètres de la requête) afin d'améliorer le temps de réponse des requêtes comportant un "join" (un produit entre plusieurs relations).

– Divers :

* create (drop) view : création (resp. suppression) de vues dynamiques sur une table ou un produit entre plusieurs tables. Elles peuvent servir à regrouper certaines informations utiles provenant de plusieurs tables.

* grant (revoke) : dons (resp. suppression) de privilèges d'un user à un autre user sur ses tables.

Nous n'avons fait figurer ici, que les requêtes les plus employées.

On dispose aussi de facilités pour sauver une requête compliquée sur fichier (afin de ne pas tout perdre si elle n'est pas syntaxiquement correcte) et pour écrire sur fichier une partie de session sqlplus (pour pouvoir imprimer des résultats)

5.2 ODL, IMPORT – EXPORT

ODL est une facilité qui permet de charger des données contenues dans les fichiers du système d'exploitation dans les tables de la base. Ces dernières doivent exister quand l'utilitaire est invoqué, car celui-ci ne crée pas de tables lui-même.

Il consent en outre :

– de charger plusieurs fichiers dans une table
– de charger des données partielles d'un fichier, dans une table
– de charger les données dans seulement quelques colonnes d'une table
(à condition que les colonnes déclarées NOT NULL y soient incluses, c'est-à-dire, les colonnes qu'on est obligé de remplir)

- de charger un nombre précis de records
- de sauter un nombre spécifié de records avant de commencer à charger
- d'arrêter ou de continuer après un certain nombre d'erreurs

Il est invoqué par la commande du style :

ODL control_file log_file nom_user_oracle/mot_de_passe −options

où :

control_file = fichier contenant la description du record du fichier contenant les données qui doivent être chargés dans la base, et une commande qui établit la correspondance entre les colonnes du record du fichier et les colonnes de la rangée de la base et effectuant le chargement.

log_file = fichier de sortie donnant les statistiques et les résultats du chargements, il donne aussi des indications sur la nature des éventuelles erreurs.

options = ce sont des numéros précédés par une lettre qui informent l'interpréteur qu'il doit charger n records, et/ou qu'il doit sauter n records avant de commencer à charger, etc...

IMPORT−EXPORT est une facilité qui fonctionne de la manière suivante.
En invoquant EXPORT, Oracle pose une petite série de questions pour savoir si l'on veut exporter tout ou une partie d'un "user" (au sense d'Oracle). Ceci est stocké dans un fichier du système d'exploitation. Ensuite, en invoquant IMPORT, ce qui est contenu dans le fichier est instauré dans une nouvelle Base de Donnée (à laquelle on accède avec le même "Oracle user" et mot de passe).
Cet outil est très utile car c'est le seul moyen de passer un user d'une machine sur une autre ou d'une base sur une autre.

Il existe d'autres utilitaires moins importants mais ne les ayant pas utilisés, nous préférons ne pas en parler.

5.3 SQLFORMS.

5.3.1 Présentation générale

SQLFORMS a été conçu pour que les utilisateurs gestionnaires, étant peu à l'aise avec le langage relationnel, puissent quand-même interagir avec les Bases de Données. En effet, au moyen de quelques clés-fonction qui s'apprennent facilement, les utilisateurs peuvent opérer sur la Base de Données en consultant les informations affichés à l'écran. Ces informations, en rapport direct avec la Base, sont gérées par les règles contenues dans les "triggers" (cf.5.3.6).

D'après les termes du cours, nous pouvons définir SQLFORMS comme étant "un serveur d'écran à l'aspect ergonomique, ayant un haut contenu informationnel".

Mais examinons d'abord cet utilitaire depuis notre point de vue, celui du concepteur (nous nous mettrons à la place de l'utilisateur lors de la définitions des clés-fonction – cf clés SQLFORMS).
Son dictionnaire est le suivant :

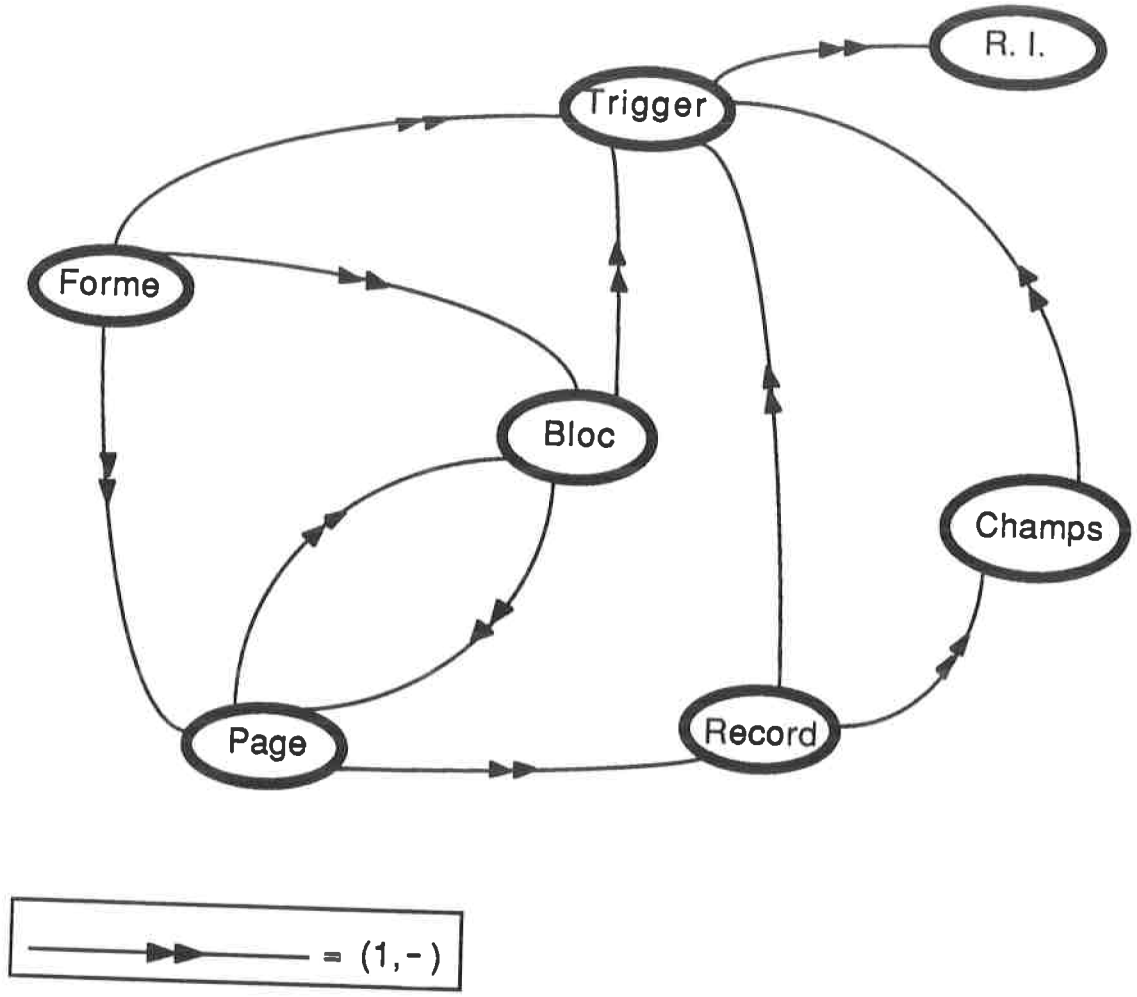

Fig. 7. Représentation des entités du DDD (SQLFORMS)

Une application, (dite forme) se compose de une ou plusieurs pages-écran représentant chacune un ou plusieurs blocs. Un bloc est l'entité destinée à recueillir les informations provenant d'une table. Un bloc est basé sur une et une seule relation (ou éventuellement sur une vue qui elle peut être construite sur plusieurs relations !), mais il peut être constitué sur rien du tout, si l'on veut stocker de l'information indépendente de la relation. Dans chaque page on ne stocke généralement pas plus qu'un bloc pour l'élégance de l'application, mais il est consenti d'en stocker plusieurs par page. Le concepteur a la possibilité d'afficher à l'écran une ou plusieurs enregistrements (records), concernant le bloc.

Un enregistrement (record) est constitué d'un ou plusieurs champs qui correspondent ou ne correspondent pas aux constituants d'une relation.
les triggers peuvent se situer au niveau du champ, du record, du bloc et de la forme (avec des restrictions selon le type, cf. 5.3.4)

Plusieurs règles d'intégrités peuvent être contenues dans un trigger, à l'interieur duquel elles vont être éxécutées sequentiellement.

Le concepteur dispose de facilités qui lui permettent de décorer ses pages, donner des prompts différents des noms des colonnes, choisir l'emplacement des champs à l'écran, rendre ces champs invisibles etc... Ensuite il doit veiller qu' il y ait un contrôle sur les données insérées, modifiées ou supprimées par l'utilisateur. Il doit décider pour chaque champ, s'il est obligatoire (même si la colonne correspondante dans la table ne l'est pas), si l'utilisateur peut entrer le curseur servant à se déplacer sur l'écran, dans l'espace écran alloué au champ, si l'unicité de la valeur fournie par l'utilisateur doit être contrôlée etc ...

Nous avons constaté un abus de langage lors des concertations entre concepteurs d'applications. C'est sur le même mot "rangée" (row en anglais) qui sert à désigner un n-uplet dans un bloc et un n-uplet dans une table. Pour le bloc le terme exact est "enregistrement" (record en anglais).
Or, il faut bien réaliser que ce qui est affiché à l'écran (le bloc) ne représente pas forcément ce qui se trouve dans la base (la table). D'une part parce que lors de l'insertion de données dans les tables, il y a toujours un instant où les données sont présentes à l'écran et pas encore dans la base (concept de transaction : tant que un "commit" n'est pas lancé, les changements ne sont pas effectifs dans la base). Cet melange de notions peut créer des problèmes, lorsque l'on travaille sur des programmes déclenchés juste avant ou juste après l'insertion des données dans la base (il faut différencier un champs d'un bloc d'une colonne de la table lorsqu'on écrit un triggers), et d'autre part parce que le concepteur a la possibilité de n'utiliser que certaines colonnes de la table dans son application, ou d'en cacher d'autres (accessibles uniquement par le concepteur par le biais des triggers).

5.3.2 Session SQLFORMS

Comme indiqué précédemment dans l'introduction SQLFORMS est basé sur IAF qui se décompose en deux parties : IAG (Interactive Application Generator) et IAP (Interactive Application Processor).

IAG est un générateur d'applications destinées à manipuler une ou plusieurs tables d'un "user". Après la session SQLFORMS (décrite plus loin) qui remplit implicitement les réponses aux questions proposés par IAG (ceci est invisible, lorsqu'on définit l'application), il produit deux fichiers dans VM (s'il n'y a pas d'erreurs). L'un, de type INP, correspond au dialogue ayant eu lieu (c'est une sorte de code source au sens des langages de programmation classiques), et l'autre de type FRM, (fichier exécutable). L'appel de l'application se fera par une commande IAP dans VM (ancienne version) ou bien

par un commande RUN dans SQLFORMS (nouvelle version) qui exécuteront le code contenu dans le fichier de type FRM.

L'application engendrée est à disposition de l'utilisateur qui pourra travailler sur un écran de type "pleine page" et manipuler les données contenues dans les tables d'un "user" en se promenant sur cet écran par l'intermédiaire de clés prédéfinies du clavier (PF1 − > PF24).

5.3.3 Les touches SQLFORMS.

L'utilisateur dialogue avec l'application par l'intermédiaire de 24 fonctions déclenchées sous SQLFORMS par les clés PF1 à PF24 qui peuvent être redéfinies. Les définitions possibles de ces fonctions sont les suivantes :

- champ suivant : permet de déplacer le curseur sur le champ suivant
- champ précédent
- rangée suivante : permet de déplacer le curseur sur le premier champ de la rangée suivante si celle-ci existe.
- rangée précédente
- bloc suivant : permet de déplacer le curseur sur le premier champ de la première rangée du bloc suivant (ou du premier bloc si on se trouvait sur le dernier).
- bloc précédent
- effacer champ, rangée, bloc (3 clés différentes) : efface de l'écran la valeur du champ, de la rangée, du bloc sur lequel se trouvait le curseur. A ne pas confondre avec une suppression.
- supprimer rangée : supprime de la base la rangée courante. Cette suppression sera en fait prise en compte lors du commit (cf clé commit).
- entrer question : signale à SQLFORMS que les valeurs qui vont être entrées dans les champs ne sont pas destinées à être insérées dans la base mais sont des critères de sélection.
- exécuter question : lance l'exécution de la question.
- commit transaction : c'est la clé qui effectue le lien entre le travail fait par l'utilisateur à l'écran et sa répercussion sur la base. En fait, quelles que soient les manipulations effectuées par l'utilisateur au travers des clés autres que commit, la base ne sera pas affectée tant que celui-ci n'aura pas pressé la clé commit. Si une des valeurs fournie par l'utilisateur ne correspond pas à la définition de la base, SQLFORMS le signalera seulement au moment du commit par l'intermédiaire d'un message du type "Oracle Error occurred ...", à moins que le concepteur n'ait prévu d'effectuer des contrôles au niveau des champs ou en pré-insertion, pré-suppression ou pré-modification (cf triggers au paragraphe suivant). Si une telle erreur se produit, aucune des manipulations effectuées par l'utilisateur depuis le dernier commit ne sera répercutée sur la base. Néanmoins, elles seront mémorisées dans des buffers afin que l'utilisateur n'ait à corriger que celle ayant entraîné l'erreur.

– afficher liste de valeurs : affiche à l'écran une liste de valeurs possibles qui font référence à une table (avec une seule colonne) préalablement créée dans la base. Ceci est utile lorsqu'un utilisateur ne se souvient plus d'une valeur qu'il doit rentrer dans un champs obligatoire pour pouvoir insérer la rangée dans la base.

– afficher dernière erreur : cette clé est bien utile car elle permet d'afficher plus en détails la dernière erreur commise par l'utilisateur. En effet, SQLFORMS ne communique les erreurs détectées par Oracle, par IAG, ou par les triggers que sur une ligne au bas de l'écran pour laisser un maximum de place au concepteur dans la définition des pages de son application.

– sortie : c'est la clé qui permet de sortir de l'application et de retourner dans VM/CMS.

D'autres clés existent, mais elles sont moins importantes que celles décrites ci-dessus.

5.3.4 Des outils importants : les triggers

Les triggers sont des adresses mémoire contenant des phrases logiques habilement placés à des points stratégiques d'une application. Si l'utilisateur fait "tourner" une forme, lorsqu'un trigger est rencontré, celui-ci sera déclenché et son contenu (instructions, macro-instructions) exécuté.

Ils sont classifiés d'abord, comme étant à trois niveaux, et pour chaque niveau, on les différencie selon un type :

– Au niveau de la forme

PRE_FORM et POST_FORM

USER_NAMED

– Au niveau du bloc

PRE_BLOCK et POST_BLOCK

PRE_RECORD et POST_RECORD

PRE_INSERT et POST_INSERT

PRE_UPDATE et POST_UPDATE

PRE_DELETE et POST_DELETE

PRE_QUERY et POST_QUERY

KEY_TRIGGERS (ex : KEY_PREV_BLOCK, KEY_NEXT_BLOCK)

USER_NAMED

– au niveau du champs

PRE_FIELD et POST_FIELD

POST_CHANGE

KEY_TRIGGERS (ex : KEY_PREV_FIELD, KEY_NEXT_FIELD)

USER_NAMED.

Or il faut préciser les différents types de triggers :

– PRE_QUERY : déclenché avant l'exécution d'une sélection.
– POST_QUERY : déclenché après l'exécution d'une sélection pour chaque rangée sortie de la table. On peut ainsi accéder à l'information sortie de la base, ce qui n'était pas le cas pour le PRE_QUERY.

– PRE_FORM (resp. POST_FORM) : associé à une forme spécifique, il est déclenché quand l'utilisateur entre dans la forme (resp. sort de la forme).

– PRE_BLOCK (resp. POST_BLOCK) : associé à un bloc précis, et il est déclenché quand l'utilisateur entre dans un bloc (resp. sort d'un bloc).

– PRE_RECORD (resp. POST_RECORD) : associé à un record et il est déclenché quand l'utilisateur rentre dans un record (resp. sort d'un record).

– PRE_FIELD (resp. POST_FIELD) : associé à un champs et il est déclenché quand l'utilisateur rentre dans un champs (resp. sort d'un champs).

– POST_CHANGE : associé au champs sur lequel il est défini. Il est déclenché quand l'utilisateur abandonne le champs, après qu'il ait effecté une mise à jour à une valeur non nulle.

– PRE_INSERT (resp. POST_INSERT) : déclenché avant (resp. après) l'insertion d'une rangée dans la table.

– PRE_UPDATE (resp. POST_UPDATE) : déclenché avant (resp. après) la modification d'une rangée de la table.

– PRE_DELETE (resp POST_DELETE) : déclenché avant (resp. après) la suppression d'une rangée.

– KEY_TRIGGERS : ils servent à redéfinir la quasi-totalité des touches-fonction du clavier. Ceci est commode car on peut par exemple interdire certaines opérations aux utilisateurs en définissant la touche à une opération nulle. On définit la touche avec des macro-instructions.

– USER_NAMED_TRIGGERS : ce sont des triggers invocables par d'autres triggers. Pour cette raison, il faut leur donner un nom. On les invoque avec la fonction EXETRG. Ce mécanisme évite d'avoir à récrire plusieurs fois le même trigger lorsqu'on en a besoin à des endroits différents de la forme.

Les requêtes SQL peuvent être insérées dans tous les différents types de trigger, sauf le KEY_TRIGGER.
Le concepteur peut par exemple s'en servir pour effectuer des contrôles sur la validité des valeurs fournies par l'utilisateur et, si nécessaire, effectuer l'annulation (rollback) de toute une transaction, si toutefois le "commit" n'a pas encore été declenché. Il peut aussi, lors d'une sélection de données depuis une table, afficher à l'écran des informations complémentaires après avoir analysé les données extraites de la base ou déclencher d'autres sélections complémentaires par le biais de macros.

Les macros sont des commandes particulières pouvant être insérées dans n'importe quel trigger. Elles ont pour effet de simuler par programme la manipulation des clés du clavier par l'utilisateur. Ainsi, par exemple, lorsque l'utilisateur désire effectuer une sélection sur une table, il doit appuyer sur la clé "enter query" puis insérer les critères de sélection dans les champs puis appuyer sur la clé "execute query". Cette manipulation peut être simulée par la macro #EXEMACRO ENTQRY;EXEQRY; à condition que l'on ait pris soin d'insérer précédemment les valeurs correspondant aux critères de sélection dans les champs par une requête tel que : SELECT * FROM * WHERE * , ou bien par une requête du type : SELECT * INTO * FROM * WHERE * . Cette dernière permet d'afficher à l'écran, des valeurs ne provenant pas directement de la base, mais des écrans dessinés dans l'application (d'où les valeur sont tirées de la base). Cette requête permet de copier des données d'un bloc à un autre. On obtient le même effet avec une requête ayant le SELECT imbriqué dans un INSERT.

Il y aurait encore beaucoup à dire sur les possibilités offertes par les macros, mais cela deviendrait une accumulation annuyeuse de détails.

On se rend bien compte de la complexité qu'on peut générer à partir des triggers. En général, plus l'application devient riche, plus il faut faire de gymnastique mentale pour pouvoir en gérer les modifications.

CHAPITRE 6
IMPLANTATION PHYSIQUE

6.1 Familiarisation avec le matériel, les logiciels et le projet

Quand nous sommes arrivés au CERN, les seules notions utiles que nous avions, étaient des idées vagues sur le projet sur lequel nous devions travailler. En effet, nous ne savions strictement rien sur le type de machine que nous devions adopter et le système d'exploitation inhérent. C'est pourquoi nous avons du d'abord étudier les manuels et faire des exercices pratiques pour apprendre VM/CMS, le système d'exploitation sur l'IBM 3090 (pas du tout "amical", il arrivait d'avoir des blocages "inextricables") son editeur et un système de traitement de texte "script" (dans lequel nous rédigeons ce mémoire).

Nous avons aussi appris à se servir des imprimantes qui, par chance, étaient situés assez près du bureau où nous avions à disposition les terminaux liés par réseaux (avec n'importe quel terminal, il est possible de se connecter à n'importe quel ordinateurs du CERN, et même à extérieur!)

Ensuite, nous avons commencé à avoir des contacts sérieux avec les responsables du projet, pour essayer d'avoir une idée la plus claire possible des exigences de nos "clients". Et cela a été un "brainstorming" continu, car au début, les responsables aussi n'avaient pas une conception précise du projet, mais cela est tout à fait normal, en considérant la complexité du "tout" (cf chapitre Description générale du projet). De plus, pour compliquer la chose, nous posions quelques objections sur certains détails, selon la conception des SGBD que nous avions appris en cours.

En même temps, nous nous exercions sur les utilitaires d'Oracle. Le premier a été SQLPLUS car c'est l'interface principale de ce SGBD et il nous faisait moins peur que les autres, puisque nous en avions déjà eu quelques notions en cours.

Il est à noter que nous avons commencé avec la version 5.0 d'Oracle, le version 5.1 n'ayant été réceptionné par le groupe GBD qu'au mois de juin. Mais cela a peu d'importance, concernant SQLPLUS, car il est resté pratiquement le même d'une version à l'autre.

Nous avons essayé de bien analyser le projet pour comprendre quelles étaient les informations utiles et quelles étaient le règles les plus importantes régissant la Base de Données.

Une fois l'analyse réalisée, les informations que devait contenir la base ayant été définies et structurées, (mais nous étions loin d'avoir la version définitive de la liste complète et correcte des informations!) nous avons pu passer à l'étape suivante de la construction.

6.2 Construction de la Base de Données

6.2.1 La structure de la Base

Pour définir la structure d'accueil des données, SQLPLUS est amplement suffisant, pour cette raison, nous avons étés capables très vite de créer une première structure de la base.
Ainsi, nous avons définit les tables et les colonnes (avec leur domaine et longueur de champs) avec la commande :

CREATE TABLE nom_table (nomcol1 dom1(lgr), nomcol2 dom2(lgr), ...)

Puis nous avons définit les clé (indexes uniques) avec la commande :

CREATE UNIQUE INDEX nom_index ON nom_table (nomcol1, nomcol2, ...)

Avec les nombreux changements qu'il y avait à effectuer sur la base, il devenait très lourd de détruire et de récréer les tables avec des simples commandes dans SQLPLUS. Ainsi nous décidions de se servir de la facilité offerte par les fichiers de commande. Nous en avons crées trois :

- le premier contenant les commandes de création des tables
- le deuxième contenant les commandes de création des indexes
- le troisième contenant les commandes de destruction des tables, du type :

DROP TABLE nom_table

Cette commande est suffisante pour "dropper" en même temps la table, les indexes la concernant et les données qui peuvent y être stockées.

On peut remarquer que nous aurions pu réunir les commandes des trois fichiers en un seul fichier, en respectant la sequentialité des traitements.
Le fichier de commande est exécutable dans SQLPLUS en précédant le nom du fichier avec la commande START (Exemple : START COMCREA).

6.2.2 Les données

Les données ont été introduites lorsque nous étions pratiquement sûr que la structure de la base ne subirait plus de changements et ceci par le biais de l'utilitaire ODL (cf LES UTILITAIRES D'ORACLE).

Toutes les données destinées à la base ont été récoltées par un responsable UCO (User Consultancy Office) qui les a organisées dans des fichiers ayant une structure ressemblant fortement à celle des tables, ce qui a facilité notre travail de chargement des données dans les tables (effectué par rangées). Pour que nous puissions effectuer le chargement, ils nous a envoyé ses fichiers dans notre directory VM. Nous nous sommes vite familiarisés avec ODL, qui est assez simple à utiliser, néanmoins nous avons rencontré quelques difficultés :

– Les NOT NULL définis sur les colonnes des tables (signifiant que la colonne doit être remplie obligatoirement) nous provoquaient des erreurs car les données fournies ne remplissaient pas toutes les colonnes de la table. Il a fallu les supprimer, pour consentir des champs nuls.

– Les données n'étaient pas toujours consistantes avec la définition de la base, car elles contenaient :

* des rangées doubles qui étaient systématiquement refusés par les indexes uniques existants sur les tables

* des rangées qui n'étaient pas en accord avec la définition de clé, c'est-à-dire qu' à une clé donnée correspondaient plusieurs champs d'une autre colonne de la table. Ces rangées étaient aussi refusées par les indexes uniques. (Mais cela était utile car cet outil nous fait automatiquement une première validation des données)

* des champs qui étaient incompatibles avec la définition de la colonne dans la base (des valeurs ayant une longueur de champ plus longue ne peuvent pas être acceptés dans la base).

Il a fallu corriger nous-même les données non consistantes ou bien arranger la définition de la longueur des champs.

– Les données chargées avec ODL sont toutes déclarées caractère, or dans la base, les colonnes de type CHAR et NUMBER les acceptent sans problèmes, alors que les colonnes de type DATE (type spécial d'Oracle qui fait automatiquement des validations sur le jour, le mois, l'année. Il est possible d'y rajouter l'heure, les minutes, les secondes avec un format spécial) refusent catégoriquement toute tentative d'insertion.

Pour cela, il a fallu faire toute une gymnastique consistent à créer d'abord des tables temporaires où les types DATE étaient déclarées comme CHAR, pour y charger les données, puis insérer ces données dans les tables définitives (avec type DATE) à l'aide de la commande INSERT * INTO * SELECT * FROM * et de la fonction TO_DATE qui sert à convertir les données de type CHAR en données de type DATE.

Au terme de la session ODL, sept tables ont pus être chargées, (les plus importantes étant celles concernant la procédure de gestion des utilisateurs : USER_DEF et PHONE_BOOK) pour les autres les données n'étaient pas disponibles.

Après avoir constaté des inconsistances dans les données, nous avons fait quelques test au moyen des requêtes SQL, pour voir si les clés étaient correctes par rapport à l'ensemble des données. Nous avons tiré la conclusion que les clés étaient correctes et que l'analyse avait été bien menée. En revanche, quelques indexes pour améliorer le temps de réponse pourraient être rajoutés.

Avec l'implémentation de la version 5.1 d'Oracle, notre manière de travailler a changé (en mieux !) dans l'élaboration de l'application destinée aux utilisateurs : SQLFORMS a remplacé IAF (ce dernier proposait un système de dialogue interactif "lourd et monotone"). Ainsi nous avons appris à fabriquer une application de toutes pièces, à l'aide de fenêtres et de ménus selectionnées à l'aise de touches-fonction.

Mais la chose qu'il ne fallait pas oublier de faire était d'exporter – importer les tables et les données (notre "user") de la version précédente à la nouvelle version, ce qui est facile à faire grâce à la facilité Oracle, justement appelée Export – Import. (cf Les utilitaires d'Oracle et cf Figure 8)

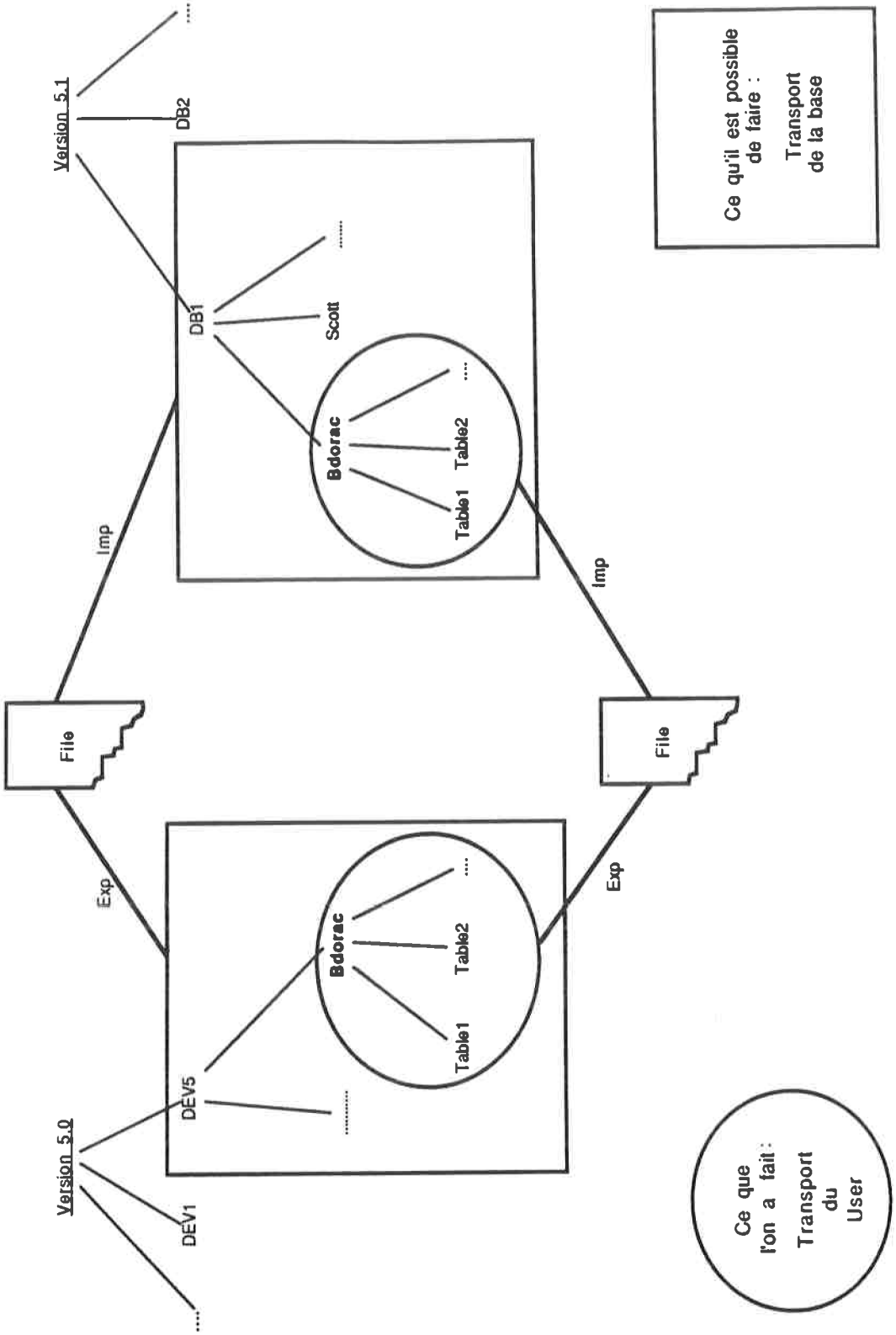

Fig. 8. Export - Import

C'est après cette dernière action que l'on pouvait finalement passer à l'élaboration de l'application.

6.3 Procédure de gestion des utilisateurs

Cette procédure doit être mise en place pour que des profanes des Bases de Données (appartenant à une catégorie donnée, ayant des privilèges donnés) puissent mettre à jour la nouvelle Base, sans en compromettre la consistance.
SQLFORMS va donc nous permettre de créer une application remplie de règles déterminant un chemin de parcours obligatoire (avec quelques options) sur les valeurs intéressantes de la Base.
Certaines touches de fonction (avec lesquelles les utilisateurs interagissent avec l'application) seront redéfinies pour éviter certaines actions sur la Base considérée comme "dangereuses".
En plus, on essaye de construire l'application de telle manière qu'elle soit "agréable" à l'utilisateur, en lui évitant le plus possible de se trouver en face d'ambiguités ou tout simplement de rester bloqué à la suite de l'utilisation d'une mauvaise touche de fonction.

Cette application est assez complexe, il faut donc procéder de façon très méthodique, en déterminant d'abord un cahier de charges où l'on fait un compte rendu de toutes les règles à respecter, en définissant les écrans le plus clairement possible pour l'utilisateur et en appliquant les règles sus mentionnées, (préalablement traduites en SQL et/ou en Macros) une à une, pour voir si effectivement la règle ne contient pas d'erreurs (synthaxiques ou logiques) et si elle marche dans le sens que l'on veut.

6.3.1 Classification des utilisateurs.

Il y a deux manières d'effectuer les classifications pour l'accès à la Base de Données :

– extérieurement par les différents "username et password" par lesquels on accède aux Bases de Données Oracle. Cela permet de sélectionner les tables ou les rangées dont l'utilisateur peut disposer, grâce aux commandes (de SQLPLUS) GRANT effectuées depuis la Base centrale.

– intérieurement par la colonne PRIV_CODE contenue dans la table LOGID_DEF déterminant un chemin (ou des chemins) dans la procédure de gestion des utilisateurs. Les utilisateurs sont partagés dans quatre catégories de privilège distinctes, qui sont les suivantes.

1) L'utilisateur courant (GU) :

il peut seulement consulter les tables concernant son information personnelle et modifier quelques données dans ces tables. Lors de L'appel de la procédure, il obtient devant lui la serie totale des informations le concernant. C'est l'utilisateur qui possède le moins de privilèges sur la Base.

2) L'administrateur de groupe (GA) :

il peut consulter et modifier toutes les données des personnes appartenant à son groupe, néanmoins, il doit se soumettre strictement à la procédure pour qu'il ne crée pas d'inconsistances.
Il peut aussi agir comme un utilisateur courant.

3) L'administrateur de système (SA) :

il peut modifier, additionner, supprimer les rangées dans n'importe quelle table (on espère qu'il sache ce qu'il fait...!) sauf GROUP_DEF, BUD_ALLOC et BUD_PRIOR. Pour cette raison, il n'y a pas de forme (application) pour cette catégorie de personnes qui devront ainsi apprendre le langage SQL. Le problème est d'empêcher l'administrateur de toucher aux tables qui lui sont interdites. Pour cela on doit vérifier son privilège (colonne stockée dans l'une des tables, à rendre absolument non modifiable!) d'après son login.

4) Le manager de ressources (RM) :

il a les mêmes privilèges que la catégorie précédente, mais en plus il peut manipuler les tables concernant les budgets. Il a donc tous les privilèges ! Ceci est dangereux dans la mesure où le manager peut créer des inconsistances sans s'en rendre compte. Pour cette raison il faudra intégrer à la Base (au moyen d'une autre forme) certaines règles évitant les erreurs (humaines) du manager.

Avec les deux niveaux de classification interne et externe, il est donc possible qu'un utilisateur ne puisse exercer ses privilèges que sur une partie des données de la Base. En effet, si on prend le cas de l'administrateur de groupe, on sait que chaque groupe peut avoir des administrateurs différents pour chaque système, donc l'administrateur ne pourra exercer ses privilèges que pour le système qui le concerne.

6.3.2 L'interface Base de Données-utilisateurs : les écrans

Après avoir déterminé la classification des utilisateurs avec leurs rôles respectifs nous allons établir le descriptif des règles de l'application écran par écran. Ces écrans sont généralement l'équivalent de blocs au sens de SQLFORMS. La forme est destinée à l'utilisateur courant (GU) et aux administrateurs de groupe et de système (GA,SA) qui peuvent valident l'information et/ou enregistrer les nouvelles personnes désirant utiliser un ordinateur ou posséder un nouveau login.
La procédure est implémentée uniquement sur le système d'exploitation VM d'IBM. Le GU n'ayant pas d'"account" sur VM accède à la procédure par l'intermédiaire du GA ou du SA qui procédera à la consultation et aux changements de son information.

Ecran 1 :

cet écran d'entrée dans la procédure permet à identifier la personne qui y accède. Le login est extrait du système et il va s'afficher automatiquement sur l'écran, sans possibilité pour l'utilisateur de le modifier (pour éviter la consultation d'information associée à un autre login). Il choisit de faire soit une consultation des données personnelles (direction écran 5), soit une consultation des données concernant une personne qui se trouve sous la hiérarchie d'un GA ou SA, en vue d'être enregistré en tant qu'utilisateur. Ce deuxième cas est donc destiné aux administrateurs qui doivent fournir de l'information complémentaire pour être identifiés : le système et le group_code à partir desquels on peut déterminer le code privilège ("1" ou "2"). donnant le pouvoir d'effectuer les opérations qui suivent (direction écran2). Leur différence de privilège vient du fait que le GA peut seulement enregistrer des utilisateurs internes et des étudiants d'été, tandis que le SA peut aussi enregistrer des utilisateurs "non humain" (entreprises, clubs) et des utilisateurs externes.

Ecran2 :

s'il accède à cet écran, c'est donc que l'utilisateur est bien un administrateur qui a le privilège de suivre la procédure selon cette voie. Il a le choix entre deux options (A ou B).
"A" signifie que l'utilisateur possède déjà un "account", l'administrateur fournit les informations du login et du système sur lequel il agit (table LOGID_DEF). Avec ces informations, on extrait un CCID unique de la Base. Si la validation avec la table est négative, il peut modifier l'information entrée ou accéder à l'écran 5.
"B" indique que la personne n'a apparemment pas d'accès à un système (mais il peut l'avoir oublié par manque d'utilisation...), c'est pourquoi l'administrateur rentre le nom et prénom de la personne pour essayer de l'identifier (à partir de la table PHONE_BOOK) et extraire de la Base les Données correspondant aux informations fournies (si elles existent!). Le résultat de la validation apparaît à l'écran 3A qui est le suivant.

Ecran 3 :

toutes les informations répondant aux critères de sélection sont affichées à l'écran. L'utilisateur doit maintenant spécifier la rangée qui correspond à son identité, mais il peut aussi ne rien spécifier du tout s'il trouve que ces informations ne sont pas correctes. S'il sélectionne une rangée, on vérifie qu'il existe une rangée équivalente, dans la table USER_DEF, (par le biais de la présence d'une valeur dans le CCID du PHONE_BOOK qui montre que la personne est déjà utilisatrice d'un ordinateur). Dans ce cas il passe directement à l'écran 5, sinon on va dans l'écran 4.

Ecran 4 :

cet écran marche selon le même principe que l'écran 3, mais la table sur lequel il est basé est USER_DEF. On va donc rechercher les rangées qui pourraient correspondre à l'identité de la personne.

Il existe deux situations d'entrée dans cet écran :

1) L'utilisateur n'avait rien sélectionné de l'écran 3.

- Dans cet écran par contre il sélectionne une rangée, il a donc un CCID et il est dirigé vers l'écran5.
- Ici, une fois de plus, il ne sélectionne pas de rangée, donc on est maintenant clair que cette personne n'est pas connue du CERN qu'il faut lui attribuer un CCID temporaire et l'envoyer à l'écran 6 de fa$con a ce qu'il fournisse des renseignements personnels pour UCO.

2) L'utilisateur avait sélectionné une rangée de l'écran 3.

- L'utilisateur sélectionne une rangée qui l'identifie,
  * si ces données correspondent aux mêmes données sélectionnées dans l'écran 3 alors c'est nécessairement qu'aucune correspondance n'avait été effectuée par le groupe UCO et cette personne dispose d'un CCID. On l'envoi à l'écran 5 et on effectue la nouvelle correspondance (cf 6.3.3).

  * si ces données ne correspondent pas tout a fait aux données sélectionnées dans l'écran 3 (erreur dans la division...), alors alors on fera une mise à jour par rapport à la rangée de l'écran 3. On l'envoi dans l'écran 5.

- L'utilisateur ne sélectionne pas de rangée dans cet écran, il faudra donc lui assigner un CCID par rapport à son CERN-ID. On l'envoi à l'écran 6.

Ecran 5 :

dans cet écran on affiche toute l'information connue sur un utilisateur (possédant un CCID). On donne l'opportunité de remplir des champs où l'information n'existe pas ou de modifier certaines informations pas tout à fait correctes. Ces informations étant selectionnées de la table USER_DEF, les modifications sont effectives sur la table après l'execution de l'ordre 'Commit' (si aucune règle d'intégrité n'est violée!) En effet, les possibilités de modifications dépendent des vérifications effectuées avec les tables PHONE_BOOK et USER_DEF.

Ecran 6 :

après avoir attribuer le CCID (et d'autres informations qui ont été trouvés dans la table PHONE_BOOK). La direction prise est celle de l'écran 6.

L'attribution du CCID temporaire se trouvera entre 200001 et 299999.

Le CCID définitif est calculé à partir du CERN_ID :

(CCID = 100000 − 2 * CERN_ID), mais uniquement pour ceux qui possède déjà un CERN_ID.

Ecran 7 :

Cet écran sert à afficher les "accounts" et LOGIN_ID dejà en possession de l'utilisateur, et eventuellement à attribuer de nouveaux "accounts" et LOGIN_ID. L'utilisateur n'aura pas la possibilité de changer son GROUP_CODE.

Ici s'arrête la gestion des utilisateurs proprement dite. A partir de ce denier écran, ce sont d'autres collaborateurs qui installent des "user-exit" nécessaires à l'envoi, aux systèmes, des messages contenant l'enregistrement effectué.

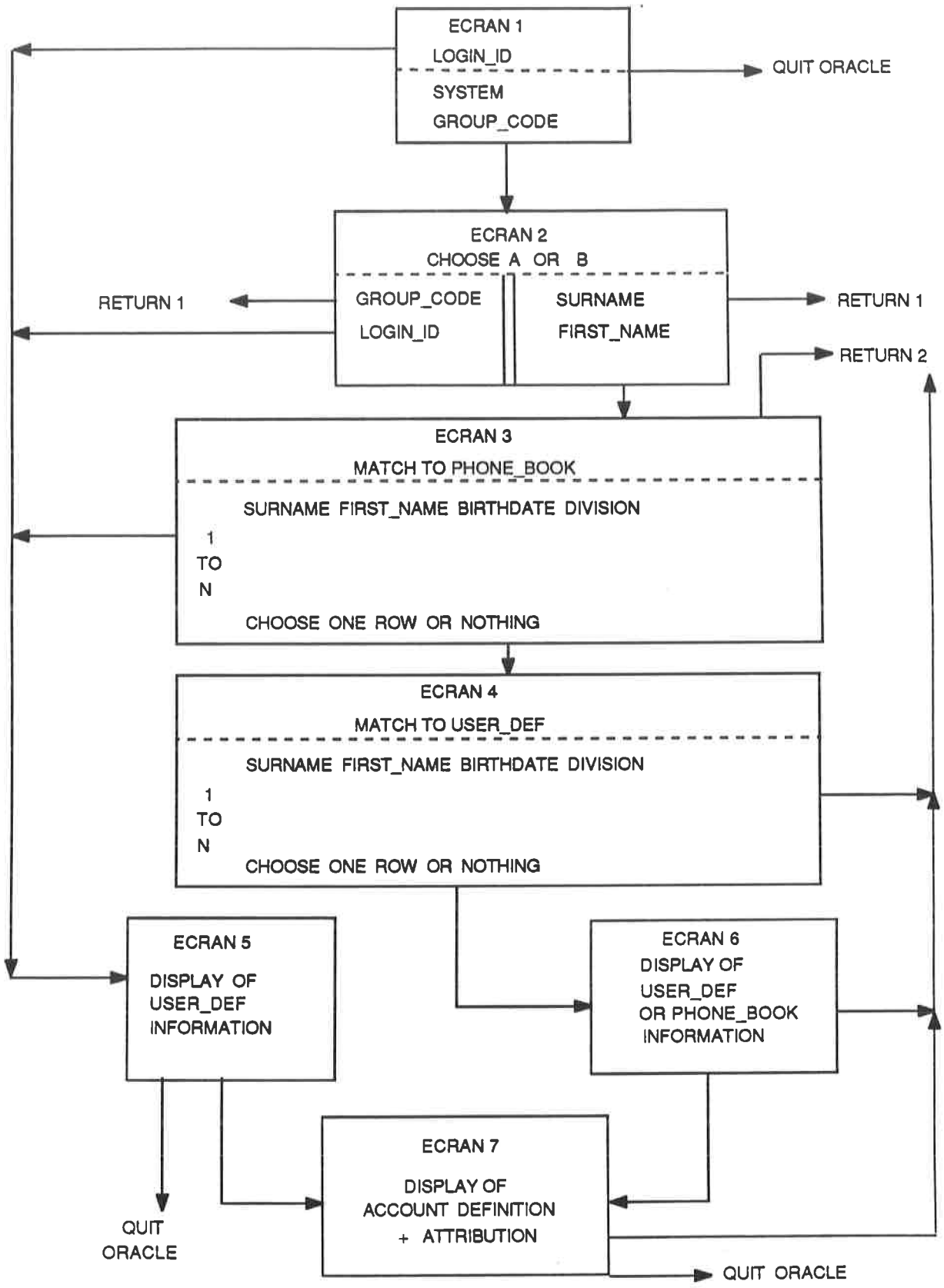

Fig. 9. Procedure de gestion des utilisateurs

6.3.3 Resolution des problèmes de consistence et de sécurité : les triggers.

Dans cette section, nous allons énumérer tous les traitements que nous avons dû implémenter de façon à réaliser notre procédure. Nous expliquerons en français le traitement effectué, puis sa traduction en langage SQL (cf. Chapitre 5.3.4. pour la définition des triggers), ainsi que le niveau où se situent les triggers.

Remarque : Nous avons dû créer un bloc spécial dans l'écran 5 que l'on a appelé ECRAN5A où nous y copions la valeur d'un CCID. Ce CCID sera ensuite recopier dans le champ ECRAN5.CCID grâce à un PRE−QUERY. La raison pour laquelle nous avons fait ce champ ECRAN5.CCID était que divers entrées sont possibles dans cet écran (le cas d'un GA qui désire faire une interrogation sur une personne donnée).

L'enregistrement d'un utilisateur n'est possible que par l'intermediaire d'un GA ou SA seulement.

_ E C R A N 1 _

AU NIVEAU DU PREMIER CHAMP

D'après le LOGIN_ID , si l'utilisateur veut passer à l'écran 5 il faudra rechercher son CCID dans la table LOGID_DEF.

REQUETE SQL :

```
    ;Field name: **KEY – NXTBLK          (ceci est une redéfinition de la touche)
;SQL>
First Step
    SELECT LOGID_DEF.CCID INTO ECRAN5A
    FROM LOGID_DEF
    WHERE ECRAN1.LOGIN_ID = LOGID_DEF.LOGIN_ID
```

Nous allons dans l'écran 5 et on y effectue une interrogation à l'aide de la commande EXEQRY.

```
Second Step
    #EXEMACRO GOBLK ECRAN5;EXEQRY;
```

Après avoir séléctionné le CCID de l'utilisateur, si son MATCH_FLAG est '0' on l'envoi au champ FIRST_NAME de l'écran 5, si c'est '1' on l'envoi au champs PREF_NAME.

```
Third Step
    #EXEMACRO CASE ECRAN5.MATCH_FLAG IS
      WHEN '0' THEN GOFLD ECRAN5.FIRST_NAME;
      WHEN '1' THEN GOFLD ECRAN5.PREF_NAME;
      WHEN OTHERS THEN NOOP;
    END CASE;
```

AU NIVEAU DU TROISIEME CHAMP

D'après le LOGIN_ID , le SYSTEM et le GROUP_CODE entrés à l'écran, nous cherchons

dans la table LOGID_DEF de la base de données une ligne où ces informations existent. Et si on trouve une telle ligne, on extrait le PRIV_CODE pour le mettre dans un champs que nous avons defini mais qui n'apparait pas dans nos écrans. La forme de SELECT que nous allons faire permet d'implanter une valeur extraite de la base (ici LOGID_DEF.PRIV_CODE) dans un champs d'un bloc (ici ECRAN1.PRIV_CODE). Ceci nous permettra de diriger l'utilisateur d'après la valeur de son PRIV_CODE.

REQUETE SQL :

```
    ;Field name: **KEY-NXFLD          (ceci est une redéfinition de la clé)
;SQL>
First Step
    SELECT LOGID_DEF.PRIV_CODE
    INTO ECRAN1.PRIV_CODE
    FROM LOGID_DEF
    WHERE LOGID_DEF.LOGIN_ID =ECRAN1.LOGIN_ID AND
    LOGID_DEF.SYSTEM =ECRAN1.SYSTEM AND
    LOGID_DEF.GROUP_CODE =ECRAN1.GROUP_CODE
Second Step
    #EXEMACRO CASE ECRAN1.PRIV_CODE IS
      WHEN '0' THEN GOFLD GROUP_CODE;     (il ne dispose pas du privilège cor-
rècte)
      WHEN '1' THEN GOBLK ECRAN2;
      WHEN OTHERS THEN NOOP;
    END CASE;
```

E C R A N 2

AU NIVEAU DU PREMIER BLOC

En fait cet écran est divisé en trois blocs. Le premier sert à l'administrateur pour faire son choix: écran A ou B.

```
    ;Field name: **KEY-NXTBLK        (ceci est une redéfinition de la clé)
;SQL>
    #EXEMACRO CASE ECRAN2.CHOIX IS
      WHEN 'A' THEN GOFLK ECRAN2A;NXTFLD;
      WHEN 'B' THEN GOFLD SCREEN2B.SURNAME;
      WHEN OTHERS THEN NOOP;
    END CASE;
```

E C R A N 2A

AU NIVEAU DU CHAMP GROUP_CODE

Dans le champs GROUP_CODE on copie automatiquement le GROUP_CODE du GA qui avait été préalablement entré dans l'écran1 car il peut uniquement opérer sur une personne qui est dans le même groupe que lui.

REQUETE SQL :

;Field name> **PRE-FIELD (déclenché quand l'utilisateur entre dans ce champ)

#COPY ECRAN1.GROUP_CODE ECRAN2A.GROUP_CODE

De nouveau, une validation nous cherchons dans la table LOGID_DEF une ligne où le LOGIN_ID, GROUP_CODE et le SYSTEM entrés à l'écran existent. Quand l'utilisateur appuiera sur la touche <NEXT BLOCK> le trigger sera déclenché. Si la rangée n'existe pas il ne pourra continuer la procédure mais pourra changer des champs (à cause d'une faute de frappe par exemple), autrement si le trigger reussit, on passera au step 2 où l'on sélectionne le CCID.

;Field name> **KEY-NXTBLK (redéfinition de la clé)

SQL>

First Step

```
SELECT GROUP_CODE,LOGIN_ID
FROM LOGID_DEF
WHERE LOGID_DEF.LOGIN_ID =ECRAN2A.LOGIN_ID AND
LOGID_DEF.GROUP_CODE =ECRAN2A.GROUP_CODE AND
LOGID_DEF.SYSTEM =ECRAN2A.SYSTEM AND
ECRAN1.LOGIN_ID = LOGID_DEF.LOGIN_ID
```

Second Step

```
SELECT CCID
INTO ECRAN5A
FROM LOGID_DEF
WHERE LOGID_DEF.LOGIN_ID = ECRAN2A.LOGIN_ID
FROM LOGID_DEF
```

Nous allons dans l'écran 5 et nous y effectuons une intérrogation.

Third Step

#EXEMACRO GOBLK ECRAN5;EXEQRY;

E C R A N 2B

AU NIVEAU DU BLOC

Dans ce bloc, nous n'avons aucune validation à faire. Il suffit que le GA entre le SURNAME et FIRST_NAME. La touche <NEXT BLOCK> l'ammènera à l'écran3 et grâce à une Macro exécutera l'interrogation qui est dans l'écran 3 dans un trigger PRE_QUERY.

REQUETE SQL

;Field name: **KEY−NXTBLK (redéfinition de la clé)
;SQL>
#EXEMACRO GOBLK ECRAN3;EXEQRY;

ECRAN 3

AU NIVEAU DU BLOC

Nous allons spécifier les conditions de l'intérrogation.

REQUETE SQL

```
    ;Field name: **PRE_QUERY
;SQL>
    SELECT SURNAME,FIRST_NAME
    INTO ECRAN3.SURNAME, ECRAN3.FIRST_NAME
    FROM PHONE_BOOK
    WHERE PHONE_BOOK.SURNAME = ECRAN2B.SURNAME AND
    PHONE_BOOK.FIRST_NAME = ECRAN2B.FIRST_NAME
```

Il s'affichera à l'écran une ou des rangées correspondantes à l'interrogation et extraite(s) du PHONE_BOOK (mais il peut aussi s'en afficher aucune si la personne n'est pas connue du CERN par exemple). L'utilisateur devra maintenant choisir UNE ou AUCUNE concordance. Evidemment il a aussi le droit de modifier sa recherche et dans ce cas il le précisera par un 'M' (pour Modify). Le trigger s'executera lorsqu'il appuiera sur la touche <NEXT BLOCK>.

REQUETE SQL :

```
    Field name: **KEY_NXTBLK          (redéfinition de la touche)
;SQL>
    #EXEMACRO CASE ECRAN3.MATCH IS
      WHEN 'Y' THEN EXETRG MATCH_UDEF;   (on execute un trigger que l'on a
nommé)
      WHEN 'N' THEN GOBLK ECRAN4;EXETRG;
      WHEN 'M' THEN CLRBLK;GOBLK ECRAN2B;CLRBLK;PRVFLD;  (on ret-
ourne a l'écran 2B)
      WHEN OTHERS THEN NOOP;
    END CASE;
```

S'il sélectionne une concordance (par un 'Y'), on exécute le trigger MATCH_UDEF. Ce trigger consistera à extraire la valeur qui est dans la colonne du CCID pour la mettre dans un champ que nous avons défini mais qui n'apparait pas dans nos écrans. Si la valeur est un '0' cela voudra dire que la personne n'a aucune concordance avec la table USER_DEF et on le dirige obligatoirement à l'écran 4. S'il existe une valeur autre que '0' (cela ne peut être que son CCID), on l'extrait, on la copie dans l'écran 5A, et on envoi l'utilisateur à l'écran 5.

REQUETE SQL

```
;Field name : MATCH_UDEF
;SQL>
First Step
    SELECT CCID
    INTO ECRAN3.CCID
    FROM PHONE_BOOK
    WHERE PHONE_BOOK.SURNAME = ECRAN2B.SURNAME AND
    PHONE_BOOK.FIRST_NAME = ECRAN2B.FIRST_NAME

Second Step
    #EXEMACRO CASE ECRAN3.CCID IS
      WHEN '0' THEN GOBLK ECRAN4;EXETRG;
      WHEN OTHERS THEN EXETRG COPY_CCID;
    END CASE;

;Field name : COPY_CCID
;SQL>
    #COPY ECRAN3.CCID ECRAN5A.CCID
    #EXEMACRO GOBLK ECRAN5;EXEQRY;GOFLD PREF_NAME;
```

ECRAN 4

AU NIVEAU DU BLOC

Même principe que pour l'écran 3 car il s'agit également d'extraire une ou des rangées d'après le SURNAME et FIRST_NAME entrés à l'écran 2A, mais la sélection se fera dans la table USER_DEF cette fois (car il est nécessaire d'effectuer la meilleure concordance possible entre la personne concernée et les informations contenues dans la Base). Nous allons spécifier les conditions de l'interrogation.

Il est a noter que L'on ne sélectionne que les rangées qui ne sont pas concordées avec le PHONE_BOOK, c'est à dire où le MATCH_FLAG = '0'.

REQUETE SQL

```
Field name> **PRE_QUERY
SQL>
SELECT SURNAME,FIRST_NAME
INTO ECRAN4.SURNAME, ECRAN4.FIRST_NAME
FROM USER_DEF
WHERE USER_DEF.SURNAME = ECRAN2B.SURNAME AND
USER_DEF.FIRST_NAME = ECRAN2B.FIRST_NAME AND
USER_DEF.MATCH_FLAG = 0
```

D'après les(s) rangée(s) affichée(s) à l'écran, l'utilisateur choisit UNE ou AUCUNE concordance ou peut aussi modifier sa recherche en retournant à l'écran 2B.

Dans le cas où il choisit une correspondance il y a différentes possibilités de traitements envisageables :

* s'il avait choisit une concordance dans l'écran 3 (mais UCO n'en n'avait pas trouvé avec USER_DEF et auquel cas son CCID était '0') il faudra effectuer la nouvelle concordance en assignant le CCID de USER_DEF au CCID du PHONE_BOOK de la rangée correspondante.

Si tout de même certaines informations n'étaient pas tout à fait correctes entre les deux rangées qui ont été sélectionnées dans l'écran 3 et l'écran 4 (differente DIVISION par exemple), nous prendrons par défaut les valeurs de la rangée choisit dans le PHONE_BOOK.

* s'il n'avait pas choisit une rangée dans l'écran 3, il faudra créer une rangée dans le

PHONE_BOOK d'après les informations de USER_DEF et lui assigner un CERN_ID entre 400001 − > 449999

Dans les deux possibilités, nous l'envoyons à l'écran 5.

S'il ne choisit pas de concordance (ou aucune ne s'affiche) deux possibilités encore se présentent :

* il avait choisit une concordance dans l'écran 3, mais son CCID = '0'. Visiblement cette personne est connue du CERN mais ne possède pas de concordance. On lui calculera son CCID d'après son CERN_ID dans le PHONE_BOOK (100000 − 2 * CERN_ID).

* il n'avait non plus choisit de concordance dans l'écran 3, cette personne est donc nouvelle au CERN et on lui assigne in CCID temporaire dans un intervalle de 200001 − > 299999.

Dans les deux possibilités, nous l'envoyons à l'écran 6.

REQUETE SQL

```
;Field name : **KEY_NXTBLK      (redéfinition de la touche)
;SQL>
    #EXEMACRO CASE ECRAN4.MATCH IS   (choix de l'écran 4)
      WHEN 'Y' THEN EXETRG SEND_5;
      WHEN 'N' THEN EXETRG SEND_6;
        WHEN 'M' THEN CLRBLK;GOBLK ECRAN_3;CLRBLK;GOBLK
ECRAN_2;CLRBLK; PRVFLD;
      WHEN OTHERS THEN NOOP;
    END CASE;

;Field name : SEND_5
;SQL>
    #EXEMACRO CASE ECRAN3.CHOIX IS  (quel était le choix dans l'écran 3)
      WHEN 'Y' THEN EXETRGMAJCCID;
      WHEN 'N' THEN EXETRGINSCCID;
      WHEN OTHERS THEN NOOP;
    END CASE;

;Field name :MAJCCID
    SQL>
First Step
    UPDATE PHONE_BOOK
    SET CCID = USER_DEF.CCID
    WHERE PHONE_BOOK.SURNAME = ECRAN2B.SURNAME AND
```

```
        PHONE_BOOK.FIRST_NAME = ECRAN2B.FIRST_NAME AND
        PHONE_BOOK.BIRTHDATE = ECRAN2B.BIRTHDATE AND
        USER_DEF.SURNAME = ECRAN2B.SURNAME AND
        USER_DEF.FIRST_NAME = ECRAN2B.FIRST_NAME AND
        USER_DEF.BIRTHDATE = ECRAN2B.BIRTHDATE

 Second Step
      COPY ECRAN4.CCID ECRAN5A.CCID
 Third Step
      #EXEMACRO GOBLK ECRAN5;EXEQRY;


 ;Field name :INSCCID
      SQL>
 First Step
      SELECT CERN_ID INTO ECRAN4.CERN_ID
      FROM EXTERNAL

 Second Step
      UPDATE EXTERNAL        (Table d'où nous sélectionnons le CCID Externe )
      SET CERN_ID = CERN_ID + 1  (les valeurs de cette table se situent entre 400001 et
499999)

 Third Step
      INSERT INTO PHONE_BOOK
      SELECT CCID,SURNAME,FIRST_NAME,DIVISION,BIRTHDATE,CERN_ID
      FROM ECRAN4
      WHERE USER_DEF.SURNAME = ECRAN2B.SURNAME AND
      USER_DEF.FIRST_NAME = ECRAN2B.FIRST_NAME AND
      USER_DEF.BIRTHDATE = ECRAN2B.BIRTHDATE



 Fourth Step
     #COPY ECRAN4.CCID ECRAN5A.CCID

     #EXEMACRO GOBLK ECRAN5;EXEQRY;
```

;Field name : SEND_6
;SQL>
#EXEMACRO CASE ECRAN3.CHOIX IS
WHEN 'Y' THEN EXETRG**CREACCID;**
WHEN 'N' THEN EXETRG**TEMPCCID;**
WHEN OTHERS THEN NOOP;
END CASE;

;Field name :**CREACCID** (ici une concordance a été effectué, nous changeons la valeur du CCID)
SQL>
First Step
UPDATE PHONE_BOOK
SET CCID = 1000000 − 2 * PHONE_BOOK.CERN_ID
WHERE PHONE_BOOK.SURNAME = ECRAN3.SURNAME AND
PHONE_BOOK.FIRST_NAME = ECRAN3.FIRST_NAME AND
PHONE_BOOK.BIRTHDATE = ECRAN3.BIRTHDATE

Second Step
SELECT CCID INTO CCID.ECRAN4
FROM PHONE_BOOK
WHERE PHONE_BOOK.SURNAME = ECRAN3.SURNAME AND
PHONE_BOOK.FIRST_NAME = ECRAN3.FIRST_NAME AND
PHONE_BOOK.BIRTHDATE = ECRAN3.BIRTHDATE
Third Step
#COPY ECRAN4.CCID ECRAN6A.CCID
Fourth Step
#EXEMACRO GOBLK ECRAN6; EXEQRY;

;Field name :**TEMPCCID**
SQL>
First Step
SELECT CCID INTO ECRAN_4 .CCID (table de valeurs entre 200001 et 299999)
FROM TEMPORARY

Second Step

```
    UPDATE TEMPORARY
    SET CCID = CCID + 1

Third Step
    COPY ECRAN4.CCID ECRAN6.CCID

Fourth Step
    #EXEMACRO GOBLK ECRAN6A;EXEQRY;
```

E C R A N 5

Cette écran est basé sur la table USER_DEF

AU NIVEAU DU BLOC

Nous allons spécifier les conditions de l'interrogation.

REQUETE SQL
;Field name : **PRE−QUERY
;SQL>
#COPY ECRAN5A.CCID ECRAN5.CCID

D'après le PRIV_CODE, seulement l'utilisateur ayant '1' pourra passer à l'écran 7.

REQUETE SQL
;Field name : **KEY−NXTBLK (redéfinition de la clé)
;SQL>
#EXEMACRO CASE ECRAN1.PRIV_CODE IS
WHEN '1' THEN GOBLK ECRAN7;EXEQRY;
WHEN OTHERS THEN EXIT;
END CASE;

AU NIVEAU DU CHAMP FIRST_NAME

Ce trigger sert à éviter que la personne qui n'a pas le privilège correcte puisse mettre à jour ce champ. C'est un trigger qui se déclenche aussitôt que l'utilisateur attente de modifier ce champ. Dans le cas où son MATCH_FLAG = '0' (il n'a pas encore de concordance avec la le PHONE_BOOK), il ne pourra pas mettre à jour ce champ.

;Field name : **PRE−FIELD
;SQL>
SELECT 'X' (on replace l'ancienne valeur de la base)
FROM USER_DEF

```
WHERE USER_DEF.CCID = ECRAN5.CCID AND
ECRAN5.MATCH_FLAG = 0
```

Même opération **AU NIVEAU DU CHAMP DIVISION et BIRTHDATE**

Après avoir effectué (ou non) des modifications , le GA ou SS peut accéder à l'écran 7 ou sortir de la procédure. Le GU doit obligatoirement sortir.

E C R A N 6

Cet écran est basé sur le PHONE_BOOK

AU NIVEAU DU BLOC

Nous allons spécifier les conditions de l'interrogation

REQUETE SQL
;Field name :**PRE – QUERY
;SQL >
#COPY ECRAN6A.CCID ECRAN6.CCID

;Field name :**NXT – BLK
;SQL >
#EXEMACRO GOBLK ECRAN7;EXEQRY;**AU NIVEAU DU CHAMP USER_CLASS**

Seulement les GA et GU pourront remplir ou mettre à jour ce champ.

REQUETE SQL
Field name : **POST – FIELD
;SQL >

First Step
SELECT USER_CLASS
INTO ECRAN6.CLASS
FROM USER_DEF
WHERE ECRAN1.LOGIN_ID = LOGID_DEF.LOGIN_ID AND
LOGID_DEF.CCID = USER_DEF.CCID

Second Step
#EXEMACRO CASE ECRAN6.CLASS IS
WHEN 'GA' THEN EXETRG GU – SS; (on vérifie les privilèges du GA)

```
      WHEN 'SS' THEN GOFLD CCID_RESP;    (il dispose du privilège corrècte)
      WHEN OTHERS THEN NOOP;             (ne dispose pas du bon privilège)
    END CASE;

Field name : GU-SS
;SQL>
    #EXEMACRO CASE ECRAN6.USER_CLASS IS
      WHEN 'GU' THEN GOFLD CCID_RESP;
      WHEN 'SS' THEN GOFLD CCID_RESP;
      WHEN OTHERS THEN NOOP;  (le GA n'a pas le droit d'insérer d'autres
    END CASE;                          classe. L'insertion ne sera pas prit en compte)
```

AU NIVEAU DU CHAMP CCID_RESP

REQUETE SQL

```
Field name : **POST-FIELD
;SQL>
First step
    SELECT CCID       (pour vérifier si ce CCID existe)
    FROM USER_DEF
    WHERE ECRAN6.CCID_RESP = USER_DEF.CCID
```

(Vérifier que le SYSTEM et GROUP_CODE de la personne concernée soit les mêmes que le responsable de cette personne)

```
Second step
    SELECT SYSTEM,GROUP_CODE
    FROM LOGID_DEF
    WHERE ECRAN6.CCID = LOGID_DEF.CCID AND
    ECRAN6.SYSTEM = ECRAN1.SYSTEM AND
    ECRAN6.GROUP_CODE = ECRAN1.GROUP_CODE
```

E C R A N 7

Il faudra afficher tous les accounts que la personne posséde déjà.
AU NIVEAU DU BLOC

REQUETE SQL
Field name : **PRE−QUERY
;SQL>
#COPY ECRAN7A.CCID ECRAN7.CCID

Ici on attribue un LOGIN_ID si le user le veut.

Field name : **POST−INSERT
;SQL>
INSERT INTO LOGID_DEF
SELECT LOGIN_ID,SYSTEM,USER_CODE,GROUP_CODE,CCID
FROM SCREEN7

Pour donner un account (avant de donner le LOGIN_ID), on propose un UUU donné par défaut (celui qui existe déjà par les accounts en sa possession). Si l'utilisateur ne l'aime pas, il peut le changer en sélectionnant un UUU parmis les dix qui se présentent à l'écran.
AU NIVEAU DU CHAMP USER_CODE

REQUETE SQL
field name : **PRE−FIELD
;SQL>
SELECT COUNT (USER_CODE)
FROM USER_CHOICE WHERE
COUNT (USER_CODE) < 10

Après le choix du USER_CODE, celui-ci sera inséré dans la base.

CHAPITRE 7
CONCLUSION

D'après ce que nous avons observé en travaillant avec une Base de Données, c'est que celle-ci n'évolue pas dans un environnement parfaitement adapté pour elle. Dans des projets informatiques de grande envergure, il faut toujours accepter des compromis car le matériel et les logiciels existants posent des contraintes importantes, le but étant de minimiser les coûts. En effet, la maintenance des SGBD est honéreuse mais ces derniers procurent des avantages décisifs. En plus, les SGBD sont considérés comme quelque chose d'encore relativement nouveau (par rapport à la gestion classique programmes-fichiers) et l'homme ne s'adapte pas à des changements trop grands ou trop rapides, il a besoin d'un temps d'adaptation. Le problème d'accepter ou non le changement est du domaine de la politique d'Entreprise, à laquelle les informaticiens sont rarement mêlés...

Grâce à la bien veillance du CERN et de ses collaborateurs, nous avons pu mettre en pratique les connaissances que nous avions acquis dans le cours d'analyse informatique, durant nos études. En effet, nous n'avions pratiquement jamais fait de travaux dans cette matière pourtant facile à exercer si le SGBD utilisé permet de créer des applications simples. Nous avons eu le privilège de faire le tour du Système de Gestion de Bases de Données Oracle, un dés SGBD plus répandu et voué à l'avenir le plus prometteur. En effet, nous avons pu constater que les inscriptions aux cours Oracle sont de plus en plus nombreux.

Nous avons pu en outre travailler dans un environnement scientifique de haut niveau et s'initier au travail d'entreprise.

D'autre part, nous nous sommes initiés à un nouveau système d'exploitation (VM/CMS), très utilisés en milieu industriel, ce qui peut toujours être utile dans le cadre d'une recherche d'emploi.

CHAPITRE 8
BIBLIOGRAPHIE

8.1 Documentation

- Cours d'analyse informatique de 2ème et 3èeme année (M. Léonard)
- Manuel d'initiation à VM – CMS (Cern – DD)
- Introduction à Oracle (Mme Ferran – DD)
- Volumes Oracle (Oracle corporation) :
  * User's Manuel
  * SQL – PLUS
  * SQL – FORMS operator's Guide
  * Oracle RDBMS
- Cernpaper user's Guide (Cern – DD)

8.2 Figures

– Fig.9 : Procédure de gestion des utilisateurs

* Remarque :

Nous avons réalisé les figures avec les logiciels MacDraw et MacDraft sur Macintosh Plus. L'impression a été effectué sur LaserWriter Plus.

CHAPITRE 9
ANNEXES

- listing contenant les ordres de création dans la Base des tables temporaires.

- listing contenant les ordres de création dans la Base des tables définitives.

- listing contenant les tables, leurs constituants, indexes et données (extraits pour ces dernières)

- listing contenant les macro-commandes et les ordres SQL relatifs à l'implantation des règles d'intégrité.

MEMOIRE DE LICENCE EN INFORMATIQUE DE GESTION

Auteurs: Silvio BOLOGNA et Catherine Jean DIXON

Page	Ligne	ERRATA	CORRIGE
21	6	ressorces	ressources
21	18	dans le cercle	dans l'oval
64	3	les triggers	Les triggers
70	13	et même a extérieur	et même a l'exterieur