MASTER EN SCIENCES INFORMATIQUES

La rétro-ingénierie des bases de données : étude de cas et analyse critique

AERTS, Thierry

Award date:
1995

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique

Année académique 1994-1995

La rétro-ingénierie des bases de données : étude de cas et analyse critique

Thierry AERTS

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

Résumé

Ce mémoire a pour but d'évaluer une méthode de rétro-ingénierie et l'outil CASE « DB-MAIN ». Dans un premier temps, une méthode de conception de bases de données est passée en revue. En s'appuyant sur celle-ci, une méthode de rétro-ingénierie des bases de données est approfondie. Celle-ci consiste dans un premier temps à récupérer toutes les informations disponibles concernant la base de données et à construire un schéma les exprimant plus ou moins directement. Ensuite, ce schéma subit des transformations, de manière à en dériver un schéma conceptuel normalisé. Les transformations de schéma jouent un rôle important dans ces méthodes et sont étudiées de plus près. Enfin, l'atelier logiciel « DB-MAIN », qui implémente une grande partie des concepts repris dans ces deux méthodes, est décrit.

La deuxième partie de ce mémoire est une étude de cas. Elle applique la méthode de rétro-ingénierie à une base de données provenant d'une entreprise. L'atelier « DB-MAIN » sert de support à toute l'étude. Enfin, à la lumière de ce travail approfondi, certaines suggestions de perfectionnement de la méthode et de l'atelier sont proposées.

Abstract

The purpose of this thesis is to evaluate a reverse-engineering method and the CASE tool « DB-MAIN ». A database design process is first reviewed. A database reverse engineering method, based on this design process, is deepened. This reverse engineering method consists first of collecting all the available information about the database to be able to construct a schema that expresses them more or less directly. In a second stage, this schema is then transformed to derive a normalised conceptual schema from it. The schema transformations play an important role inside these methods and are studied too. Finally, the tool « DB-MAIN », which implements the main part of the methods' concepts, is described.

The second part of this thesis is a case study. It applies the reverse engineering method to a database coming from a firm. The tool « DB-MAIN » supports the whole study. Finally, at the light of this deep work, some suggestions to improve both the method and the tool are proposed.

Je voudrais remercier tout d'abord mon promoteur, M. Hainaut, pour les conseils et les remarques judicieuses qu'il m'a fournis au long de ce mémoire. Il m'a permis d'accéder à un niveau relativement élevé dans le domaine des bases de données, alors que ce concept n'avait encore aucune signification pour moi il y a deux ans...

Je tiens à remercier également les personnes que j'ai rencontrées lors de ma présence au sein de l'entreprise « Baxter ». Tout d'abord, M. Moulin, qui m'a accueilli et a consacré plusieurs heures de son temps précieux à m'expliquer les rouages de son département, et M. Wautier, qui m'a introduit auprès des personnes compétentes lorsque j'avais un problème. Ensuite, M. Lauriente, qui m'a beaucoup aidé dans mes recherches et sans qui je ne serais sans doute pas parvenu au résultat obtenu, et Mme Wuilbaut, qui a contribué à ma compréhension du fonctionnement du département « achat » de Baxter. Enfin, tout le personnel des départements « achat » et « IS », qui était très disponible et très accueillant.

Il me reste à remercier l'équipe DB-MAIN, qui m'a tiré de certains mauvais pas et me permettait d'être à jour dans les versions de l'atelier.

Table des matières

Chapitre 1 : Introduction *7*

Chapitre 2 : La méthode et l'outil employés *9*

2.1 Le développement d'une base de données **9**
2.1.1 L'analyse conceptuelle 9
2.1.2 La conception logique 11
2.1.3 La conception physique 12

2.2 La rétro-ingéniérie des bases de données **13**
2.2.1 L'extraction des structures de données 15
2.2.2 La conceptualisation des structures de données 17
2.2.2.1 La conceptualisation de base 20
2.2.2.2 La normalisation conceptuelle 26

2.3 Les transformations de schéma **29**
2.3.1 Définition générale 29
2.3.2 Réversibilité d'une transformation 29
2.3.3 Exemples 30

2.4 L'atelier DB-MAIN **32**
2.4.1 Le modèle de spécification 32
2.4.1.1 Le projet 32
2.4.1.2 Le schéma 32
2.4.1.3 Le type d'entités 32
2.4.1.4 La collection 33
2.4.1.5 Le type d'associations 33
2.4.1.6 L'attribut 33
2.4.1.7 Le groupe 34
2.4.1.8 Autres caractéristiques 34
2.4.2 L'interface de DB-MAIN 34
2.4.3 Les fonctionnalités de DB-MAIN 35
2.4.3.1 Les transformations élémentaires 35
2.4.3.2 Application simultanées de plusieurs transformations 37
2.4.3.3 Le « parser » de code source 37
2.4.3.4 Les facilités de « pattern matching » 37

Chapitre 3 : Enoncé du problème *40*

3.1 Le contexte du problème **40**
3.1.1 Le département « achat » 40

3.2 Le problème **42**

3.3 La configuration de travail **43**

3.4 La partie abordée dans ce mémoire **44**

Chapitre 4 : Les schémas bruts *45*

4.1 L'extraction du schéma global **45**

4.2 L'analyse des fichiers logiques **48**

4.3 L'atelier DB-Main **50**

Chapitre 5 : Les schémas enrichis *52*

5.1 Les champs obligatoires **53**

5.2 Les contraintes sur le domaine **57**

5.3 Les identifiants **59**

5.4 Les contraintes de référence **63**

5.5 Les différentes décompositions **66**

5.6 Les autres contraintes **67**

5.7 L'intégration des vues externes **68**

Chapitre 6 : Le schéma conceptuel de base *70*

6.1 La préparation du schéma **70**

6.2 La détraduction et la désoptimisation du schéma **71**

6.2.1 Les en-têtes et les lignes de commandes 72

6.2.2.1 Contraintes de « PURCHASE MASTER » 74

6.2.2.2 Contraintes de « PURCHASE ITEM » 75

6.2.2 Les fournisseurs 83

6.2.3 Les termes de paiement et de livraison et les modes de livraison 85

Chapitre 7 : Le schéma normalisé *91*

7.1 Vue générale **91**

7.2 Le type d'entités « PURCHASE ITEM » **92**

7.3 Le type d'entités « PURCHASE MASTER » **92**

7.4 Le type d'entités « PART MASTER » **95**

7.5 Le type d'entités « VENDOR IDENTIFICATION » **95**

Chapitre 8 : Analyse critique de la méthode et de l'outil employés *98*

8.1 Rappel de la méthode **98**

8.2 Généralités **99**

8.3 Critique de la méthode et de l'atelier **99**

8.3.1 L'extraction du schéma brut 99

8.3.2 L'analyse des programmes 100

8.3.3 La simplification du schéma 102

8.3.4 La détraduction et la désoptimisation 102

8.3.5 La normalisation conceptuelle 105

8.4 En bref **105**

Chapitre 9 : Conclusion *108*

Chapitre 1 : Introduction

La rétro-ingénierie des bases de données est une branche de l'informatique, et plus particulièrement de la gestion des bases de données, qui prend de plus en plus d'importance. En effet, l'informatique est à un tournant de son histoire : les systèmes, les logiciels, les composantes, les architectures,... sont de plus en plus performants. Les bases de données n'échappent pas à la règle et sont contraintes elles-aussi d'évoluer. Pour y arriver, il est nécessaire d'en comprendre les différents rouages. C'est là que la rétro-ingénierie joue un rôle : il s'agit, à partir d'une base de données existante, de retrouver sa signification, ses contraintes, ses caractéristiques physiques,... Bref, tout ce qui peut contribuer à une compréhension la plus complète possible de cette base de données.

Les finalités de l'exécution d'un processus de rétro-ingénierie sont multiples. Citons, entre autres, la redocumentation d'une base de données, la ré-ingénierie d'un système, l'intégration de plusieurs systèmes, la conversion d'une base de données vers un autre système, la maintenance d'une base de données, l'extraction d'une partie réutilisable de celle-ci,... Les exemples ne manquent donc pas.

La rétro-ingénierie est, à peu de choses près, le processus inverse de la conception d'une base de données. Il est donc intéressant de s'attarder quelque peu sur ce concept avant de s'intéresser à la rétro-ingénierie elle-même. Nous allons donc introduire, dans un premier temps, une méthode de conception d'une base de données. Celle-ci commence par analyser les besoins de l'utilisateur pour en dériver un schéma conceptuel, qui est, en quelque sorte, la « première version » de la base de données. Ensuite, ce schéma est traduit, pour le rendre conforme au SGBD[1] utilisé, et optimisé, afin d'améliorer ses performances futures. Enfin, il reste à dériver les vues externes et à régler les paramètres physiques pour obtenir une base de données utilisable.

C'est de cette méthode de conception des bases de données que s'inspire la méthode de rétro-ingénierie que nous allons décrire dans ce mémoire. Celle-ci peut être considérée comme le processus inverse de la conception : elle utilise les produits disponibles (description des structures de données, programmes utilisant la base de données, paramètres physiques, vues externes,...) pour construire un schéma brut : celui-ci est constitué du schéma global augmenté des contraintes, retrouvées entre autres dans les programmes. Ce schéma brut est ensuite conceptualisé, grâce à l'application de transformations de schéma qui permettent de passer de structures conformes au SGBD utilisé, ou optimisant les performances de la base de données, à des structures plus générales. Le schéma obtenu est alors normalisé pour enfin produire un schéma conceptuel qui rend compte de la sémantique de la base de données de départ, indépendamment de toute considération spécifique à un modèle ou un raisonnement particulier.

Dans ces deux méthodes, les transformations de schéma jouent un rôle très important. Nous nous y intéresserons donc de plus près. Nous en donnerons une définition rigoureuse et nous examinerons les concepts de transformations non-réversibles, simplement réversibles et symétriquement réversibles.

[1] Système de Gestion de Bases de Données

Ces méthodes théoriques ont servi de base au développement d'un atelier logiciel : DB-MAIN. Celui-ci est un projet de l'Institut d'Informatique des Facultés qui a démarré en 1993. Le produit est un logiciel qui supporte la conception, la maintenance, la rétro-ingénierie et la ré-ingénierie des bases de données. Comme tout nouveau produit, il n'est pas parfait et doit encore recevoir de nombreuses améliorations. Ce mémoire est l'occasion d'évaluer et de critiquer cet outil, tant positivement que négativement.

La deuxième partie de ce mémoire va donc mettre cette méthode et cet outil à l'épreuve lors d'une étude de cas. Celle-ci n'a rien à voir avec les exemples pédagogiques et méthodologiques qui sont abordés lors des cours universitaires. C'est un cas « grandeur nature », qui provient d'une grosse entreprise et qui nous permettra d'évaluer la méthode de rétro-ingénierie exposée et l'atelier DB-MAIN. Cette base de données provient du département « achat » de l'entreprise « Baxter » de Lessines, en Belgique. C'est une base de donnnées relationnelle. Toutefois, le SGBD dans lequel elle est implémentée est très pauvre, ce qui sera pour nous des occasions supplémentaires de tester la méthode et l'atelier lors de la recheche de toutes les contraintes.

Ce mémoire est donc la rencontre de deux pôles : la méthode et l'outil, d'une part, et la problème posé par l'entreprise, d'autre part. C'est en réunissant les deux que nous allons tenter de faire apparaître quelques enseignements qui permettront à l'informatique de demain d'être encore meilleure...

Chapitre 2 : La méthode et l'outil employés

Ce chapitre a pour but de présenter la méthode de rétro-ingénierie et l'atelier logiciel utilisés pour effectuer l'étude de cas. Dans un premier temps, nous allons présenter une méthode de conception des bases de données. Ensuite, nous aborderons la méthode de rétro-ingénierie des bases de données, qui s'appuie sur cette méthode de conception.

Les transformations de schéma jouent un rôle important dans ces deux méthodes. Nous en verrons donc une définition, ainsi que certaines propriétés.

Enfin, nous introduirons l'atelier logiciel DB-MAIN, qui s'inspire de tous ces concepts. Ce mémoire continuera ensuite par une étude de cas qui permettra d'évaluer la méthode de rétro-ingénierie et l'atelier.

2.1 Le développement d'une base de données

Le développement d'une base de données (ou forward engineering ou encore Database design) est composé d'étapes successives correspondant chacune à un niveau d'abstraction : l'analyse conceptuelle (conceptual design), la conception logique (logical design) et la conception physique (physical design). Ces grandes étapes sont reprises à la Figure 1 et seront détaillées par la suite.

2.1.1 L'analyse conceptuelle

L'analyse conceptuelle consiste à générer un schéma conceptuel normalisé à partir des besoins de l'utilisateur. Le schéma produit sera en fait une représentation de la base de données indépendante du SGBD utilisé. Cette étape est divisée en deux phases : l'analyse des besoins, qui produit un schéma conceptuel de base à partir des besoins de l'utilisateur, et la normalisation conceptuelle, qui a pour but de normaliser le schéma obtenu, c'est-à-dire le mettre sous la 3ème forme normale[2]. Ces deux phases sont reprises à la Figure 2.

[2]Pour rappel, un schéma est sous 3ème forme normale si deux conditions sont vérifiées :

1. Les attributs des types d'entités possèdent un domaine simple (les attributs ne peuvent pas être décomposables)
2. Si un attribut A d'un type d'entité R dépend fonctionnellement d'un groupe d'attributs X, ce dernier doit être un identifiant de R ou A doit être un composant d'un identifiant de R.

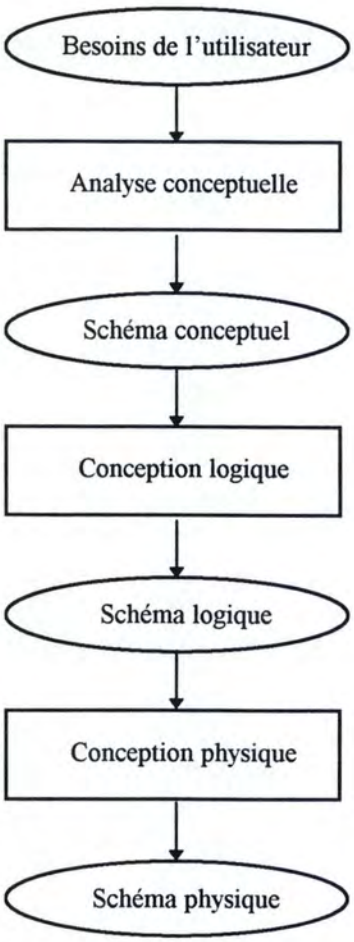


Figure 1 - La conception d'une base de données

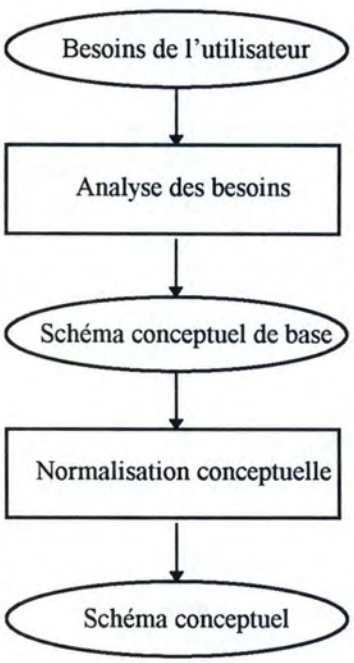


Figure 2 - L'analyse conceptuelle

L'analyse des besoins produit un schéma entités-associations à partir des besoins de l'utilisateur. Nous ne détaillons pas cette étape ici car elle concerne la conception d'un système d'information, ce qui nous éloigne quelque peu des bases de données. Le lecteur intéressé pourra se référer à [BODART, 1989].

La phase de normalisation conceptuelle a pour but de rendre le schéma normalisé, minimal et clair. On notera que cette phase peut être perçue comme une suite de transformations de schéma. Toutefois, nous ne les détaillerons pas ici car elles seront abordées lors l'exposé de la méthode de la rétro-ingéniérie.

2.1.2 La conception logique

La conception logique a pour but de traduire la schéma conceptuel en un schéma optimisé et conforme à un SGBD. Le schéma produit sera donc exprimé dans le modèle du SGBD (par exemple le modèle relationnel) et sera optimisé de manière à satisfaire aux critères de performance tels que l'espace occupé ou le temps de réponse. Cette étape comprend trois phases : la simplification du schéma, l'optimisation et la traduction. Ces phases sont reprises à la Figure 3.

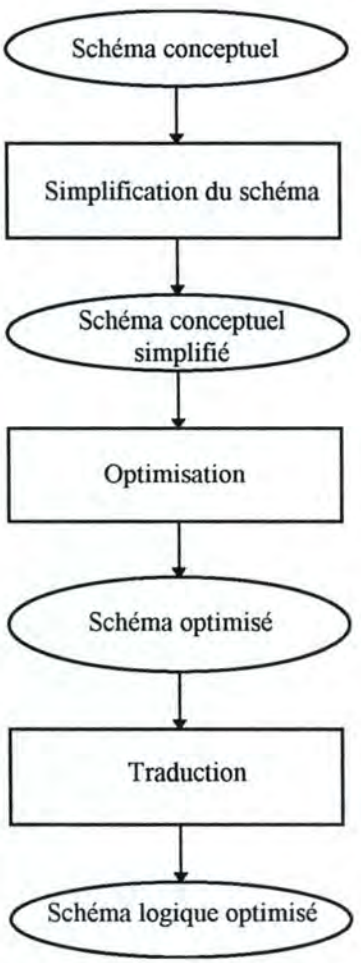


Figure 3 - La conception logique

La phase de simplification consiste à transformer les structures complexes du schéma conceptuel en structures plus simples et plus facilement traduisibles en structures conformes au SGBD. On citera surtout les relations IS-A et les types d'associations N-aires. Ceci se fera bien sûr en utilisant des transformations de schéma.

La phase d'optimisation consiste à agir au niveau du schéma pour améliorer les performances de la base de données (principalement l'espace occupé et le temps de réponse). Les techniques employées ici sont principalement la dénormalisation, la redondance structurelle et la restructuration, c'est à dire la fragmentation horizontale ou verticale. L'optimisation d'une base de données peut se faire de deux façons : indépendamment du SGBD utilisé ou en fonction de celui-ci.

La phase de traduction traduit le schéma en un schéma conforme au SGBD. Pour cela, les structures ne faisant pas partie du modèle du SGBD sont traduites, via des transformations de schéma, en structures conformes. On citera, par exemple pour le modèle relationnel, la transformation des types d'associations fonctionnels en contraintes de référence. Cette phase produit également une partie qui n'est pas supportée par le SGBD. Celle-ci contient généralement des contraintes d'intégrité complexes ou n'ayant pas de lien avec le SGBD (espace occupé, temps de réponse,...)

Ces deux phases sont exécutées en parallèle. En effet, il arrive que l'application d'une transformation optimisante fasse apparaître des structures non-conforme au SGBD, qu'il faudra donc traduire. De même, la traduction d'une structure peut faire apparaître une possibilité d'optimisation qui n'était pas explicite.

La conception logique dans son ensemble est constituée également de transformations de schémas. Toutefois, les transformations intéressantes diffèrent selon le SGBD choisi. Ainsi, par exemple, le modèle CODASYL inclut les types d'associations fonctionnels alors que le modèle relationnel ne le fait pas. Il faudra donc dans ce cas les transformer en contraintes de référence, alors que ce n'est pas nécessaire en CODASYL.

2.1.3 La conception physique

La conception physique est composée de quatres phases : le codage du schéma logique en instructions compréhensibles par le SGBD (typiquement des instructions SQL), le codage des structures ne faisant pas partie du modèle du SGBD (typiquement en fragments de code procédural), le réglage physique, consistant à définir les index, les tailles des tampons, les clusters et autres paramètres techniques, et la dérivation des vues, qui consiste à générer les vues externes qu'auront les utilisateurs et les programmes sur la base de données et à les traduire en instructions de la même façon que le schéma global (instructions du SGBD et fragments de code procédural). Ces phases sont reprises à la Figure 4.

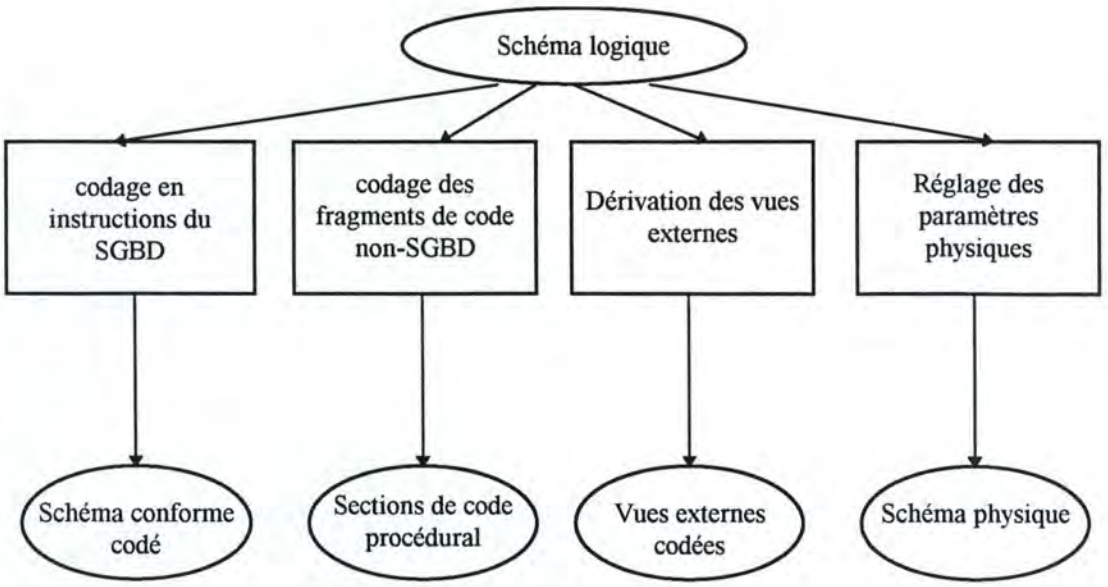


Figure 4 - La conception physique

On ne parle plus de transformations de schéma à ce niveau, étant donné que l'étape de la conception physique consiste à traduire un schéma en instructions ou à régler des paramètres.

La méthode de conception d'une base de données que nous venons de parcourir va servir de référence pour la méthode de rétro-ingéniérie des bases de données que nous allons examiner. Cependant, nous nous attarderons plus longtemps sur cette dernière, notamment concernant les transformations de schéma utilisées au long du processus.

2.2 La rétro-ingéniérie des bases de données

La rétro-ingéniérie d'une base de données est le processus inverse de la conception. Il s'agit, à partir du schéma codé, des fragments de code, des données elles-mêmes et du schéma physique, de reconstituer un schéma conceptuel qui pourrait être celui qui a servi à construire la base de données. La forme conditionnelle « pourrait » est employée car ce schéma n'est généralement pas le schéma conceptuel de départ : ce dernier, s'il existe ou a existé, n'est certainement pas le même. En effet, depuis sa création, la base de données a évolué : on a ajouté, éliminé, modifié des éléments la composant, altérant ainsi sa structure. De plus, les bases de données employées actuellement n'ont généralement pas été construites en utilisant des méthodes rigoureuses comme celle présentée à la section précédente, ce qui implique la non-existence d'un schéma conceptuel.

Le but est de reconstituer un schéma conceptuel qui contient l'information contenue dans la base de données, afin de percevoir la signification de chacun de ses éléments, d'en retrouver les contraintes, les caractéristiques physiques,... La méthode proposée ci-dessous se base sur la méthode de conception présentée à la section précédente. Globalement, on peut dire qu'il suffit de « renverser » le processus, en partant des produits de la conception physique, qui sont généralement disponibles, et de remonter les différentes phases pour retrouver un schéma conceptuel. Toutefois, ce n'est pas si simple et il est nécessaire d'introduire des nuances à cette affirmation, comme nous le verrons au cours de l'exposé de cette méthode.

Nous partons donc d'une vue très générale, qui consiste à récupérer le schéma conforme codé, le code des programmes utilisant la base de données, dans lesquels nous rechercherons les fragments de code procédural utilisés pour vérifier certaines contraintes, le contenu de la base de données et le schéma physique. Cette vue générale est présentée à la Figure 5.

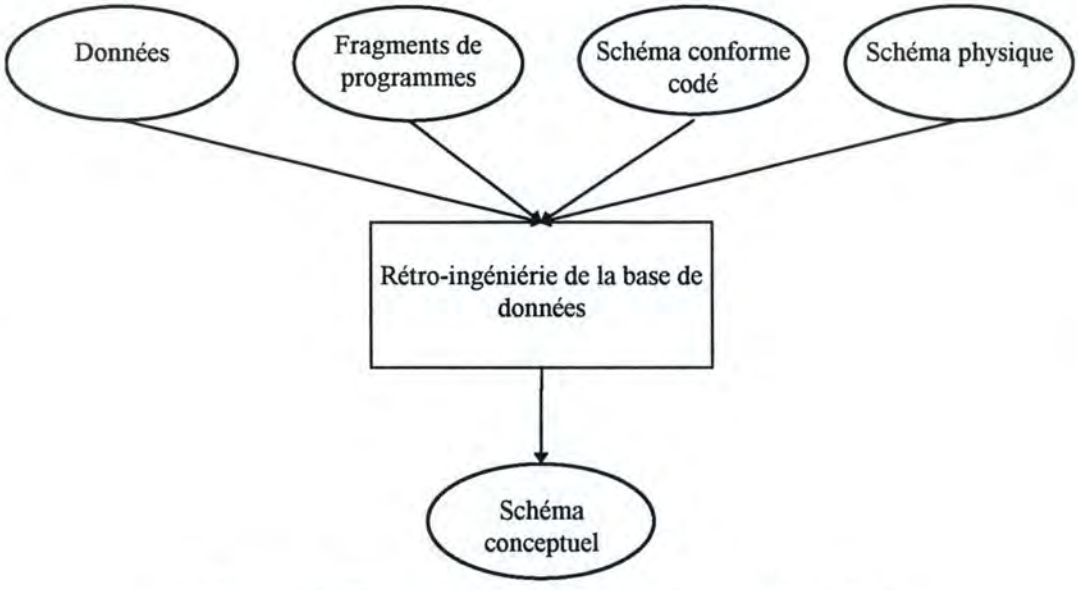


Figure 5 - Rétro-ingéniérie d'une base de données

Tout comme la méthode de conception comportait deux parties principales (la conception logique et la conception physique), la méthode de rétro-ingéniérie sera divisée en deux grandes étapes : l'extraction des structures de données et la conceptualisation des structures de données. Elles correspondent plus ou moins à l'inverse de la conception physique et de la conception logique. Il n'y a évidemment pas d'étape correspondant à l'analyse conceptuelle. En effet, il n'y a pas de sens à rechercher les besoins d'un utilisateur qui étaient à la base de la conception de la base de données, d'autant plus que ces besoins ne sont certainement plus d'actualité... Les étapes composant la méthode de rétro-ingénierie des bases de données que nous allons détailler sont reprises à la Figure 6.

L'extraction des structures de données a pour but de fournir une description complète de la base de données à partir des informations disponibles. Cette étape est plus ou moins facile à réaliser, selon le nombre de documents disponibles à propos de la base de données. Elle produira un schéma logique optimisé comprenant d'une part la description des structures implémentées sous forme d'un schéma conforme au SGBD utilisé et d'autre part une partie comportant les structures non supportées par le SGBD (cette partie provient principalement des fragments de code et du schéma physique). On peut considérer que cette étape correspond à peu de choses près à l'inverse de la conception physique.

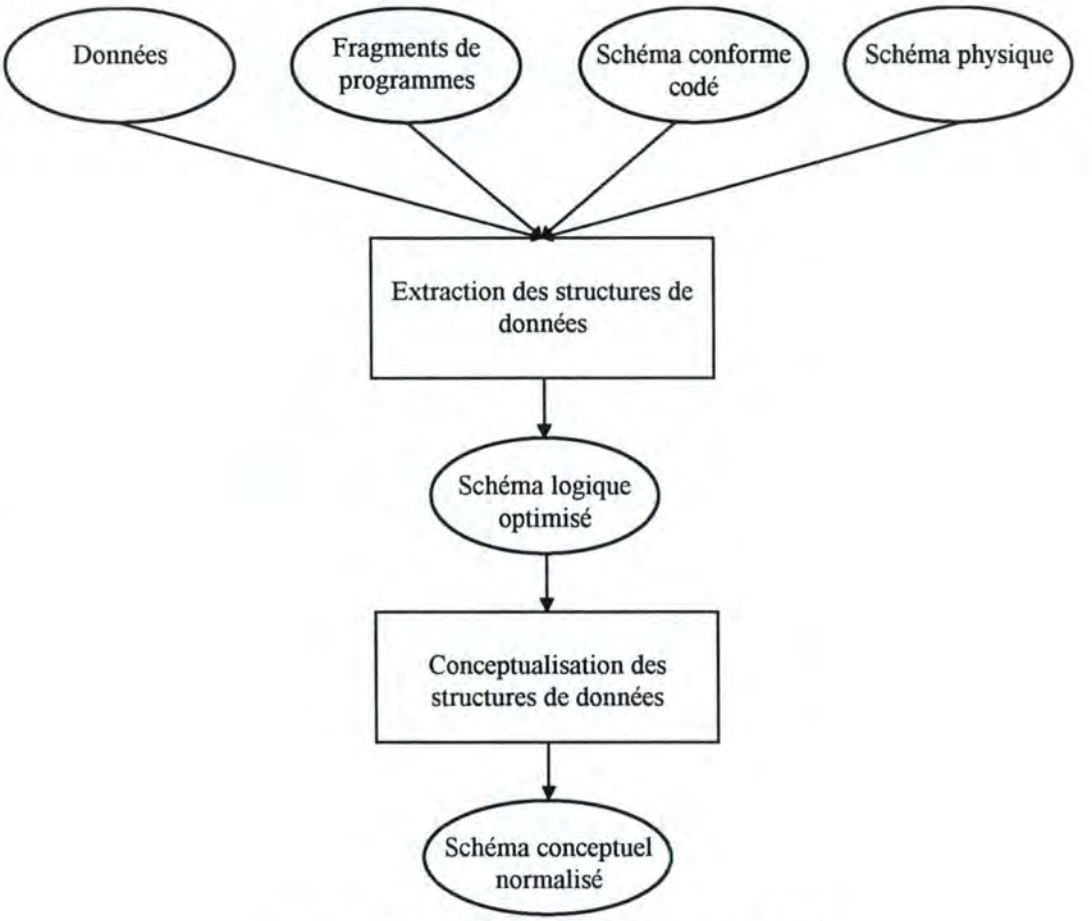


Figure 6 - Les étapes de la rétro-ingéniérie

La conceptualisation des structures de données a pour but de transformer le schéma logique produit à l'étape précédente en un schéma conceptuel normalisé, de façon à retrouver la sémantique du schéma logique. Elle consiste principalement à éliminer les structures propres au SGBD, comme les contraintes de référence dans le modèle relationnel, et les structures résultants des optimisations. On peut considérer que cette étape correspond à peu de choses près à l'inverse de la conception logique et à la normalisation conceptuelle. On remarque déjà une divergence par rapport à l'idée d'inverser complètement la processus de conception : l'étape de normalisation conceptuelle intervient de façon directe et pas de façon « inverse » comme les autres étapes.

Nous allons maintenant détailler ces deux parties.

2.2.1 L'extraction des structures de données

L'extraction des structures de données est la partie qui dépend fortement du SGBD utilisé. On tient compte de structures dépendant de celui-ci, des optimisations apportées,...

Les informations dont nous disposons sont le schéma global, le contenu des fichiers, qui pourront éventuellement valider ou invalider certaines hypothèses, le schéma physique, qui contient les paramètres physiques tels que les index, les clusters,... et les programmes contenant les fragments de code validant certaines contraintes. Quatre phases composeront cette étape : l'extraction du schéma global à partir de son codage et du schéma physique,

l'analyse des programmes pour en extraire les contraintes qui y sont vérifiées, l'analyse des données pour valider ou invalider certaines contraintes incertaines et l'intégration des vues quand le schéma global est divisé en plusieurs parties. Ces phases sont reprises à la Figure 7.

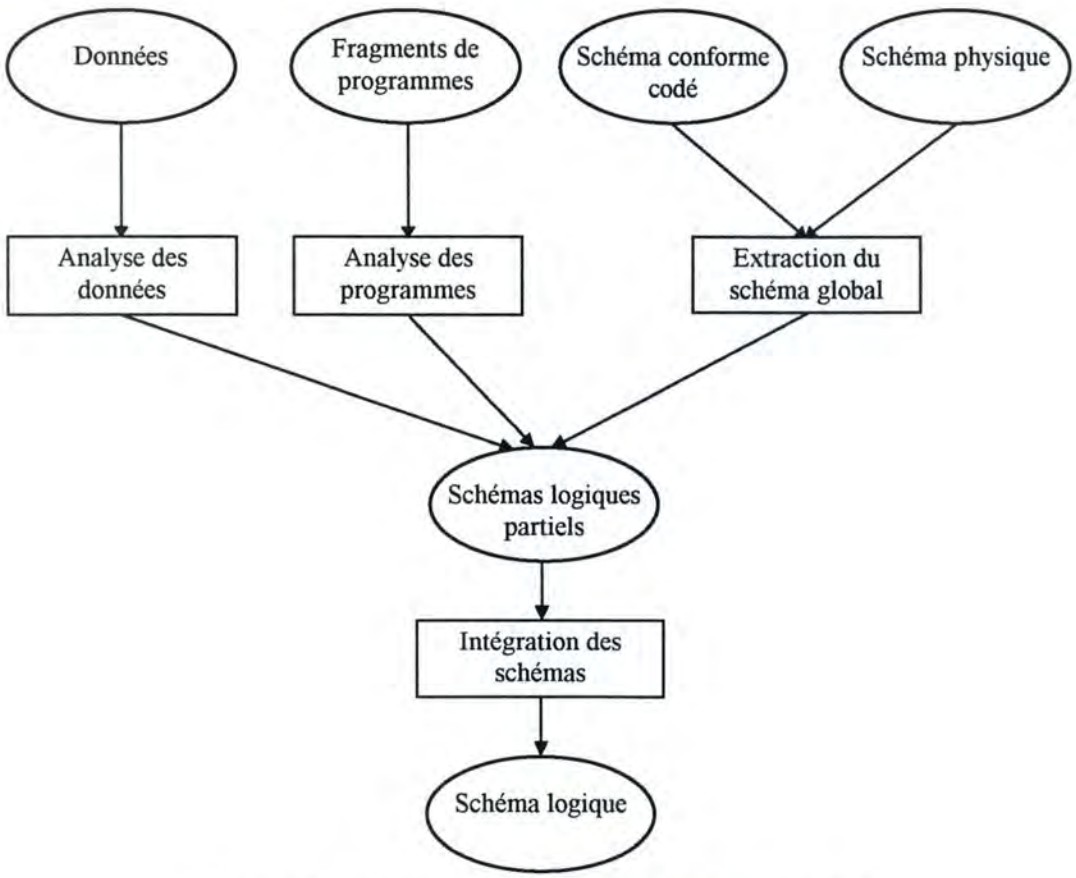


Figure 7 - Extraction des structures de données

L'extraction du schéma global analyse en fait le texte généré par les instructions définissant les structures primaires de la base de données. Deux cas peuvent se présenter : soit les instructions définissent le schéma global directement, soit ce dernier est éparpillé dans les différents programmes, et chacun de ceux-ci contient une partie du schéma. Dans ce dernier cas, l'extraction du schéma sera multiple et il faudra alors intégrer ces différentes vues. Cette analyse produit un schéma dit « brut » qui sera raffiné par l'analyse des autres composantes de la base de données et sur lequel viendront se greffer les structures annexes, comme les contraintes d'intégrité. Cette phase est plus ou moins l'inverse de la phase de production du code spécifique au SGBD.

L'analyse des programmes permet de compléter ce schéma avec les structures et les contraintes qui y sont présentes. Il s'agit généralement de celles qui ne sont pas supportées par le SGBD. Voici une liste des contraintes les plus souvent rencontrées à ce stade dans le cas du modèle relationnel :

- Les contraintes d'égalité de champs autres que des contraintes de référence
- Les contraintes sur le domaine d'un attribut
- Certaines contraintes de référence
- Les différentes décompositions d'un type d'entités
- Parfois des identifiants

- Les autres contraintes, regroupant tout ce qui n'est pas directement exprimable sur un schéma classique

Toutes ces contraintes sont retrouvées en analysant minutieusement les programmes dont nous disposons. Une des techniques possibles consiste à simuler leur exécution et à prendre note des contraintes au fur et à mesure. Nous verrons lors de l'étude de cas comment améliorer cette technique. Cette phase est l'inverse de la production des fragments de code procédural destinés à implémenter les structures non supportées par le SGBD.

L'analyse des données intervient principalement quand certaines structures ou contraintes sont décelées, mais pour lesquelles il subsiste un doute quant à leur validité. Dans ce cas, une analyse des données permet souvent de trancher (positivement ou négativement). Cette phase permet donc également de raffiner le schéma global. Elle n'a pas d'équivalent dans le processus de conception, vu que cette dernière produit une base de données vide.

Enfin, l'intégration des schémas est nécessaire quand le schéma global n'est pas disponible directement. Il se peut que chaque programme utilise une partie de la base de données et qu'il n'existe nulle part de description complète de celle-ci. C'est le cas, par exemple, des fichiers COBOL, pour lesquels chaque programme possède sa propre vue, qui n'a peut-être rien à voir avec celle du programme voisin. Dans ce cas, l'extraction du schéma global sera multiple et produira plusieurs vues, partielles ou totales, de la base de données générale. Il faudra alors intégrer ces vues afin de produire un schéma unique. On peut dire que cette phase correspond plus ou moins à la dérivation des vues externes.

Cette étape d'extraction des structures fournit le matériau brut. Le schéma produit à ce moment est appelé « enrichi » : il s'agit du schéma brut, que nous avons enrichi des contraintes retrouvées grâce à l'analyse des programmes. Il s'agit maintenant de le travailler, à l'aide des transformations de schéma, afin d'en dériver un schéma général et débarrassé de tout ce qui ne contribue pas à son expressivité.

2.2.2 La conceptualisation des structures de données

Le schéma logique étant extrait, il faut maintenant le rendre lisible. En effet, bien souvent, le schéma trouvé à ce niveau-ci est complexe et illisible, ceci étant dû aux nombreuses modifications, ajouts ou retraits de structures dans la base de données au cours de son existence.

Pour rendre ce schéma plus agréable, la technique qui s'impose est la conceptualisation : on élimine les structures dépendant du SGBD et des optimisations effectuées sur la base de données lors de la phase de conceptualisation de base et on normalise le schéma via la normalisation conceptuelle : ce sont les deux phases de cette étape, qui sont reprises à la Figure 8.

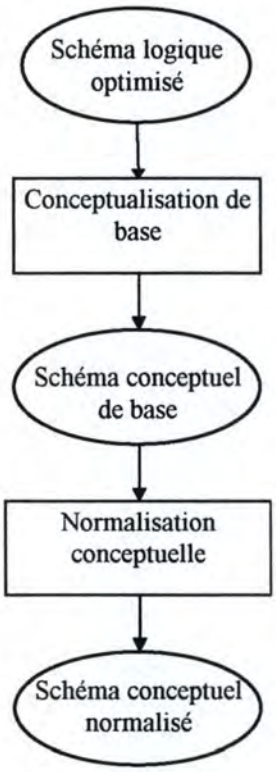
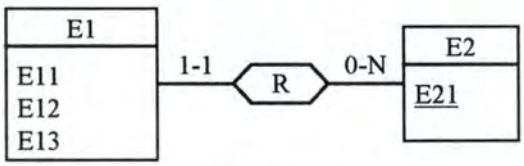


Figure 8 - Conceptualisation des structures de données

A partir de ce moment, nous allons utiliser des transformations de schéma. Celles-ci servent à mettre en correspondance deux constructions. Ces dernières peuvent être un type d'entités, un type d'associations, ou une combinaison de plusieurs d'entre-eux. Les transformations que nous utilisons permettent de passer de la première construction à la deuxième et inversément sans perte d'information. C'est ce qui permet d'affirmer que le schéma conceptuel que nous allons construire est équivalent au schéma logique optimisé de départ. Nous verrons à la section 2.3 une définition rigoureuse de la notion de transformation de schéma et nous en donnerons quelques propriétés.

Les différents schémas que nous allons examiner possèdent des contraintes écrites dans un pseudo-langage. Nous allons expliciter quelque peu les termes les plus fréquemment utilisés.

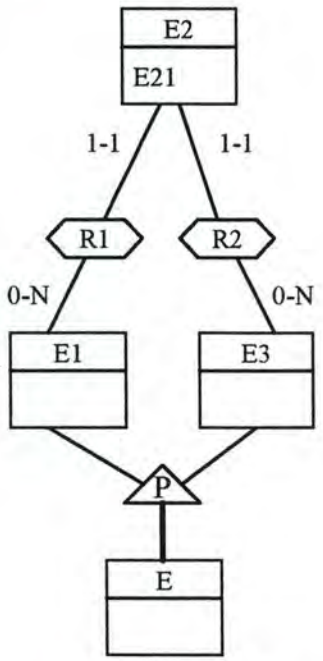
Lorsque deux types d'entités E1 et E2 sont reliés par un type d'associations R, la formule *E1.R.E2* représente l'ensemble des entités de E2 accessibles, via R, à partir de E1. Le même raisonnement s'applique aux attributs. Un exemple est donnée à la Figure 9.



E12 = E1.R.E2.E21

Figure 9 - Exemple de contrainte (1)

Si R1 et R2 sont deux types d'associations, *R1* ∪ *R2* représente l'union de ces deux types d'associations. Ceci est possible est considérant qu'un type d'associations est un ensemble de couples d'entités. Un exemple est donnée à la Figure 10. Dans cet exemple, on peut également remarquer la relation de sous-typage, représentée par un triangle, et l'expression d'un identifiant, sous la forme *id(E)* = ... Citons également l'expression du domaine d'un attribut sous la forme *dom(E21)* = ...



id(E2) = (R1 ∪ R2).E, E21

Figure 10 - Exemple de contrainte (2)

Si E1, E2 et E3 sont des types d'entités et R1 et R2 des types d'associations, *R2 o R1* représente la composée des types d'associations R1 et R2. En fait, *E1.(R2 o R1).E3 = E1.R1.E2.R2.E3*. Un exemple évident se trouve plus loin dans ce chapitre, à la transformation T - 12.

Si R1 et R2 sont deux types d'associations, *R1 in R2* signifie que R1 est inclus dans R2. A nouveau, on considère un type d'associations sous la forme d'un ensemble de couples.

Enfin, le symbole '->' indique une contrainte de référence, avec laquelle nous avons pris certaines libertés : le groupe d'attributs référencé n'est pas toujours identifiant, ce qui nous permet d'exprimer pas la même occasion une contrainte de dénormalisation. De nombreux exemples sont traités dans le chapitre 6.

2.2.2.1 La conceptualisation de base

La conceptualisation de base est une phase critique. C'est elle qui traduit les structures spécifiques en structures générales. Le point délicat est de savoir quelles sont les structures à transformer et comment les transformer. Pour cela, cette phase peut se découper en trois parties : la préparation du schéma, qui « nettoie » le schéma logique, la dé-traduction, qui est l'inverse de la traduction et la désoptimisation, qui est l'inverse de la phase d'optimisation du schéma logique. Cette phase est détaillée à la Figure 11.

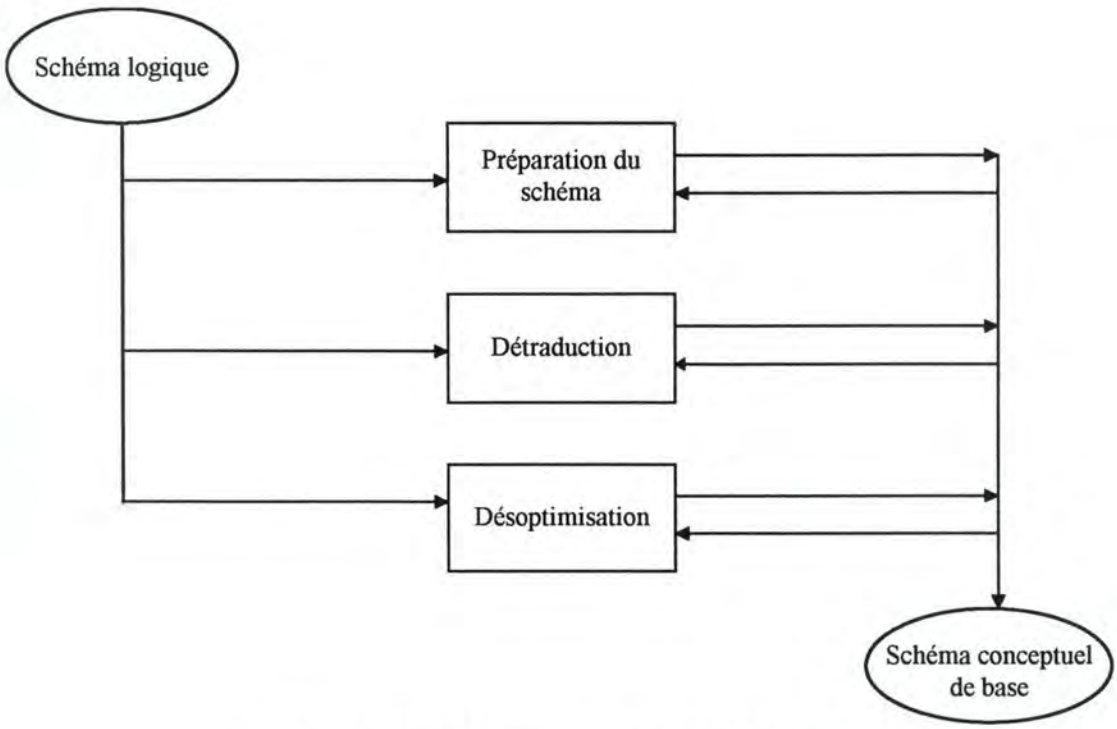


Figure 11 - La conceptualisation de base

La préparation du schéma

La préparation du schéma consiste à en éliminer les structures d'accès, les structures redondantes, à renommer éventuellement les objets,... Il s'agit en fait de faire un nettoyage préliminaire du schéma afin d'en éliminer ce qui n'a rien à voir avec la conceptualisation. Les transformations de schéma interviennent peu à ce stade-ci et sont plutôt utiles lors des deux parties suivantes.

La détraduction

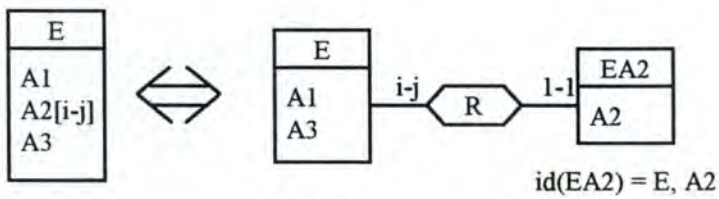
La détraduction remplace les structures spécifiques au SGBD par des structures générales. Les transformations de schémas foisonnent à ce stade-ci. Toutefois, elles ne sont pas les mêmes dans le cas de fichiers COBOL que dans les SGBD relationnels ou encore les SGBD CODASYL... Nous allons nous intéresser en particulier aux SGBD relationnels et aux transformations qui y sont liées.

Les caractéristiques du modèle relationnel sont les suivantes :

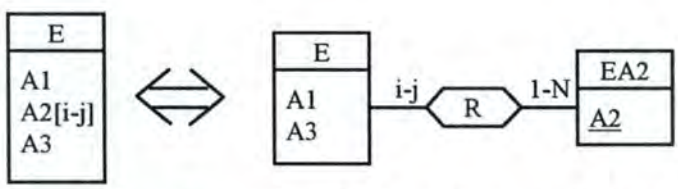
- Pas de types d'associations
- Les attributs sont monovalués et atomiques. Ils peuvent être obligatoires ou facultatifs
- Les seules contraintes d'intégrité supportées sont les identifiants et les attributs de référence
- Chaque identifiant est une clé d'accès

A la lumière de ces caractéristiques, on peut déduire quelques transformations intéressantes à ce niveau-ci.

Un attribut multivalué peut être représenté de différentes manières. On peut le représenter par un type d'entités, et ce, de deux façons différentes. Soit on crée un type d'entités contenant chaque instance de l'attribut - on parle alors de représentation par les instances - (voir transformation T - 1) - soit on crée un type d'entités contenant les différentes valeurs prises par l'attribut - on parle alors de représentation par les valeurs - (voir transformation T - 2). Dans ces deux cas, lors du processus de rétro-ingénierie, on cherchera les structures correspondant à ces représentations pour leur appliquer la transformation inverse, de façon à retrouver la structure de départ. Ces structures se repèrent facilement, car elles contiennent un type d'entités ne possédant qu'un seul attribut. On pourra donc appliquer la transformation inverse correspondante de manière plus ou moins systématique.



T - 1 : Transformation d'un attribut multivalué en type d'entités (représentation par les instances)



T - 2 : Transformation d'un attribut multivalué en type d'entités (représentation par les valeurs)

Il existe deux autres manières de représenter un attribut multivalué : la représentation par concaténation (voir transformation T - 3) ou par instanciation (voir transformation T - 4). Ces deux représentations ne nécessitent pas la création de type d'entités supplémentaire. C'est en examinant les domaines et la signification des attributs qu'on pourra déceler ces structures. Dans ces cas-ci, la décision d'appliquer la transformation est plus subjective, car elle dépend de l'interprétation du schéma et plus uniquement de sa structure.

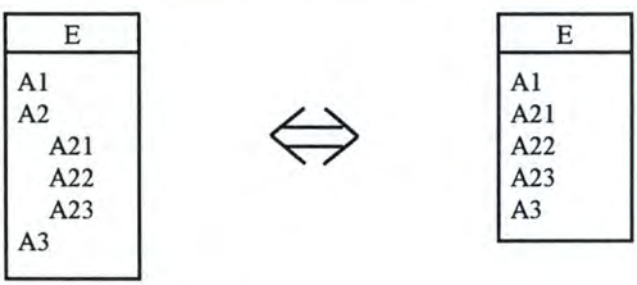
$dom(E.A2) = X$ $dom(E.A2') = X^5$

T - 3 : Représentation d'un attribut multivalué par concaténation

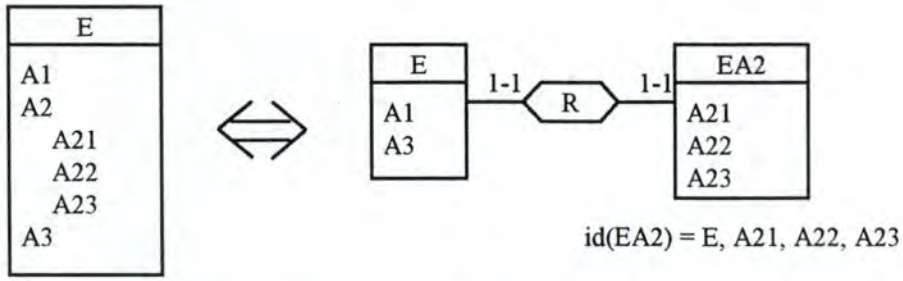
$dom(E.A2_i) = dom(E.A2_j)$, $1 \leq i,j \leq 4$

T - 4 : Représentation d'un attribut multivalué par instanciation

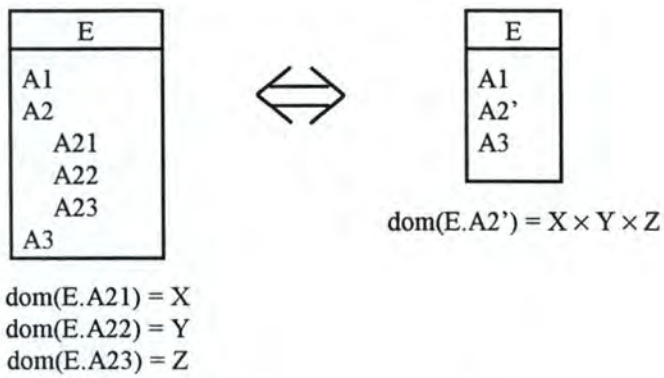
Un attribut décomposable peut être représenté de quatre façons. Tout d'abord, la représentation par désagrégation (voir transformation T - 5) s'inspire quelque peu de la représentation par instanciation d'un attribut multivalué : chaque composante de l'attribut devient un attribut à part entière. La représentation par concaténation (voir transformation T - 6) est l'équivalent direct de la même représentation dans le cas d'un attribut multivalué. C'est également le cas pour les représentations par les instances (voir transformation T - 7) et par les valeurs (voir transformation T - 8).



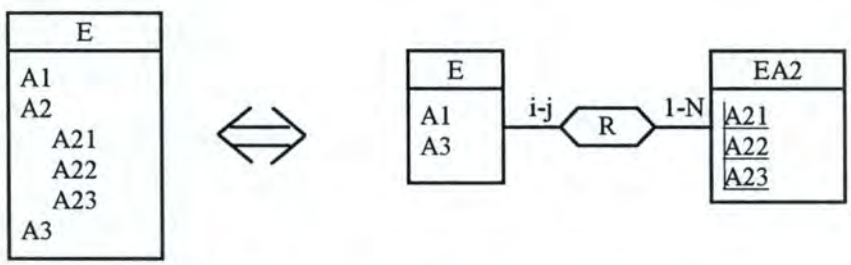
T - 5 : Représentation d'un attribut décomposable par désagrégation



T - 6 : Représentation d'un attribut décomposable par un type d'entités (représentation par les instances)



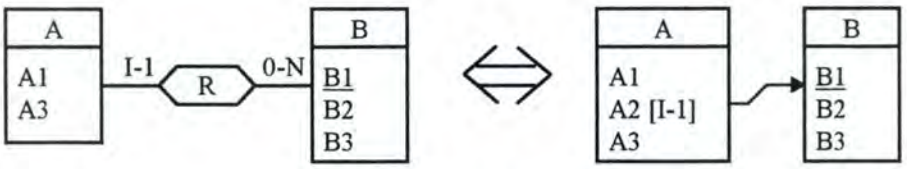
T - 7 : Représentation d'un attribut décomposable par concaténation



T - 8 : Représentation d'un attribut décomposable par un type d'entités (représentation par les valeurs)

De nouveau, les deux premières représentations supposent une interprétation sémantique des attributs concaténés ou des instanciations de l'attribut dont il est question pour déceler un rapport entre la représentation choisie et l'attribut décomposable. Les deux autres (représentations par les instances ou les valeurs) sont moins facilement décelables que dans le cas d'un attribut multivalué. En effet, dans ce cas, les types d'entités représentant l'attribut ne comportent eux-mêmes qu'un seul attribut, alors que dans le cas d'un attribut décomposable, les types d'entités comportent autant d'attributs qu'il n'y a de composantes dans l'attribut général. Il pourrait parfois y avoir matière à discussion, notamment lorsque le nombre de composantes est élevé, et un choix s'impose, qui sera à nouveau guidé par les caractéristiques sémantiques de la structure analysée.

Enfin, les types d'associations fonctionnels sont représentés par des attributs de référence (voir transformation T - 9). Il s'agit de remplacer le type d'associations R par l'ajout de l'identifiant du type d'entités B dans le type d'entités A. Une contrainte de référence indiquera la provenance de l'attribut ajouté.



T - 9 : Représentation d'un type d'associations fonctionnel en attribut de référence

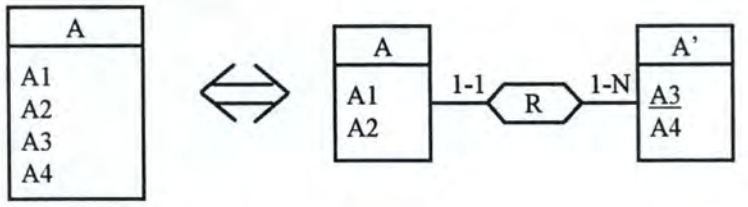
Dans le processus de rétro-ingénierie, les contraintes de référence sont immédiatement reconnaissables, à condition que le SGBD les supporte et qu'il soit possible d'en connaître la liste. Dans le cas contraire, c'est généralement l'analyse des programmes qui les fera apparaître. Souvent, la sémantique du schéma fournit des bonnes indications (format des données, noms et structures similaires,...) quant à l'existence d'une telle contrainte. L'analyse des programmes et des données permet alors de confirmer ou d'infirmer la supposition.

La désoptimisation

La désoptimisation a pour but de détecter et éliminer les structures qui ne sont pas « naturelles ». Celles-ci résultent généralement de modifications apportées au schéma en vue d'accroître ses performances. Cette phase sera bien sûr l'inverse de l'optimisation du schéma.

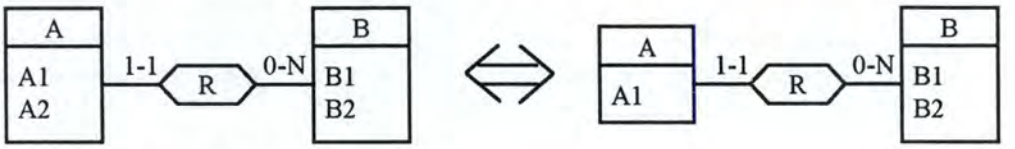
L'optimisation d'un schéma a souvent pour conséquence directe de le rendre beaucoup moins clair : afin de gagner en temps de réponse ou en espace occupé, on « détruit » un beau schéma pour le rendre illisible. Voici un relevé des transformations qui sont à la base d'une bonne partie des optimisations apportées à un schéma.

La dénormalisation d'un type d'entités (voir transformations T - 10 et T - 11) est chose courante lors des optimisations : pour ne pas devoir faire de jointures, on introduit un (des) attribut(s) supplémentaire(s) lors de la transformation d'un type d'associations fonctionnels en attribut de référence. La conséquence est la création de dépendances fonctionnelles au sein du type d'entités principal. Or, les dépendances fonctionnelles sont indésirables dans un schéma normalisé, d'où le nom de dénormalisation pour cette optimisation. Les dépendances fonctionnelles ne sont pas toujours faciles à déceler. Elles font généralement partie des contraintes retrouvées lors de l'analyse des programmes.



A : A3 → A4

T - 10 : Elimination de la dénormalisation d'un type d'entités (1)

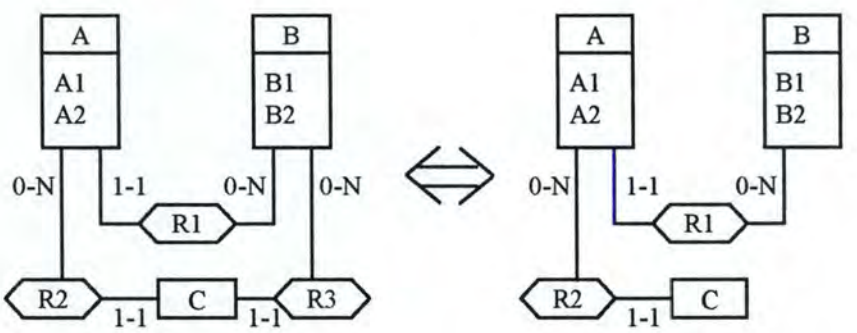


A.A2 = A.R.B.B2

T - 11 : Elimination de la dénormalisation d'un type d'entités (2)

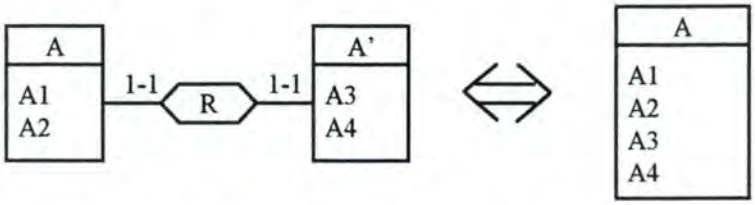
La redondance structurelle consiste à ajouter un type d'associations redondant, de façon à court-circuiter un chemin plus long (voir transformation T - 12). Le terme de « redondance » s'explique par le fait que cette optimisation ajoute un élément redondant (le type d'associations ajouté étant la composée des deux autres).

Une dernière transformation importante utilisée lors de l'optimisation d'un schéma est l'éclatement d'un type d'entités en deux parties. Ceci permet de réduire la taille des structures (mais pas la taille totale du schéma) et donc d'accélérer le temps d'accès à ces structures (voir transformation T - 13). Ce structures sont aisément reconnaissables lorsque la détraduction et la recherche dans les programmes sont bien faites : elles sont composées de types d'associations « one-to-one ».



R3 = R1 o R2

T - 12 : Elimination d'un type d'associations redondant



T - 13 : Elimination de l'éclatement d'un type d'entités

Les optimisations les plus courantes ont été reprises ci-dessus. Les optimisations dépendant du modèle du SGBD utilisent des transformations spécifiques au modèle, mais également et surtout à la marque, voire à la version du SGBD choisi. Mais on entre ici dans un niveau beaucoup trop spécifique pour pouvoir détailler tous les cas de figure. On se réfèrera pour cela au manuel du SGBD pour de plus amples informations.

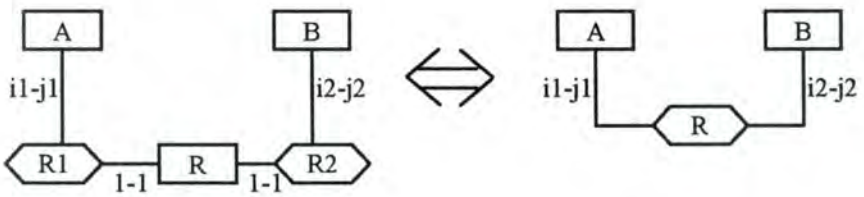
La détraduction et la désoptimisation se font en parallèle : en fait, il ne s'agit pas à proprement parler de processus mais plutôt de types de raisonnement. On peut même appliquer une transformation sans pouvoir la classer dans une des deux parties... L'important à ce stade-ci est de revenir à un schéma qui ne porte plus la marque du SGBD : on ne doit plus trouver, par exemple, de contraintes de référence qui sont la « marque de fabrique » des SGBD relationnels.

2.2.2.2 La normalisation conceptuelle

La phase de normalisation conceptuelle a le même but que celle du processus de conception des bases de données : il s'agit de rendre le schéma le plus clair possible, minimal et normalisé. Il s'agit également de retrouver des structures qui auraient pu disparaître lors de la phase de simplification, c'est à dire les relations IS-A et les types d'associations N-aires.

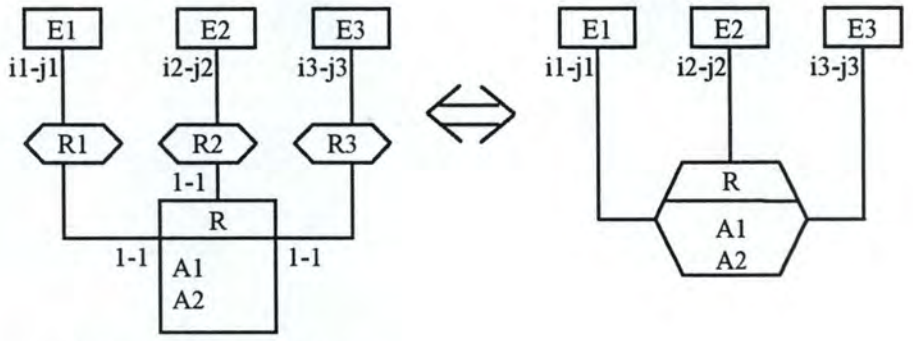
Les transformations utilisées à ce stade-ci sont nombreuses. Nous allons décrire les principales et celles utilisées dans la suite de ce mémoire. Le lecteur intéressé par une liste plus exhaustive pourra se référer à [HAINAUT, 1993b].

Les trois premières transformations T - 14, T - 15 et T - 16) ont pour but de transformer un type d'entités en type d'associations. Lors de la conception de la base de données, ces types d'associations ont subi des transformations afin de les rendre fonctionnels et il s'agit de leur rendre leur apparence première.



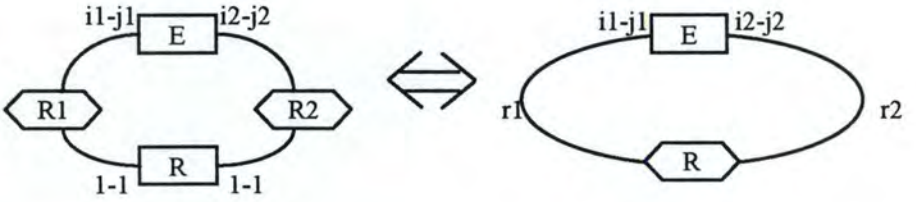
id(R) = R1.A, R2.B

T - 14 : Transformation d'un type d'entités en type d'associations binaire



id(R) = R1.E1, R2.E2, R3.E3

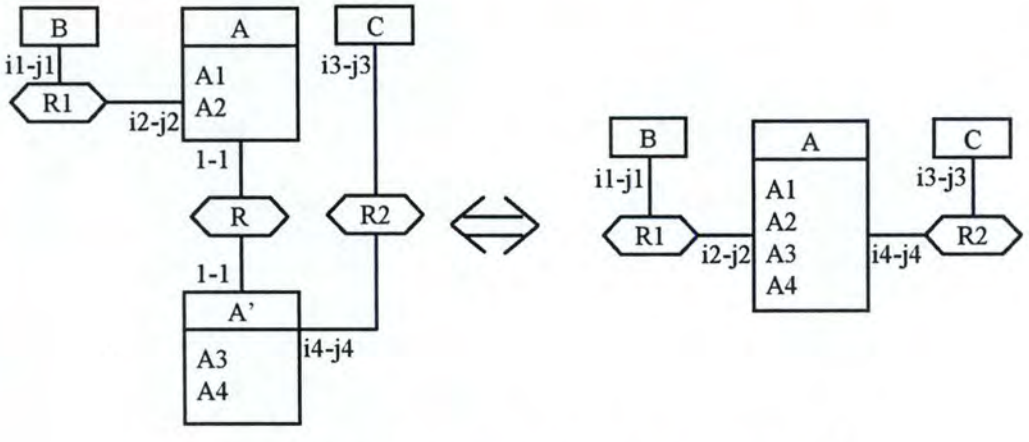
T - 15 : Transformation d'un type d'entités en type d'associations ternaire



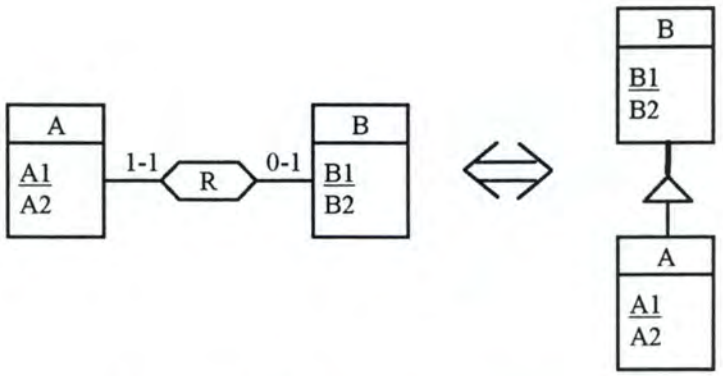
id(R) = R1.E, R2.E

T - 16 : Transformation d'un type d'entités en type d'associations récursif

La présence de types d'associations « one-to-one » est souvent le signe de l'éclatement d'un type d'entités ou de la présence d'un sous-type. Les deux transformations suivantes (**Erreur! Source du renvoi introuvable.** et T - 18) permettent de régulariser ces situations.

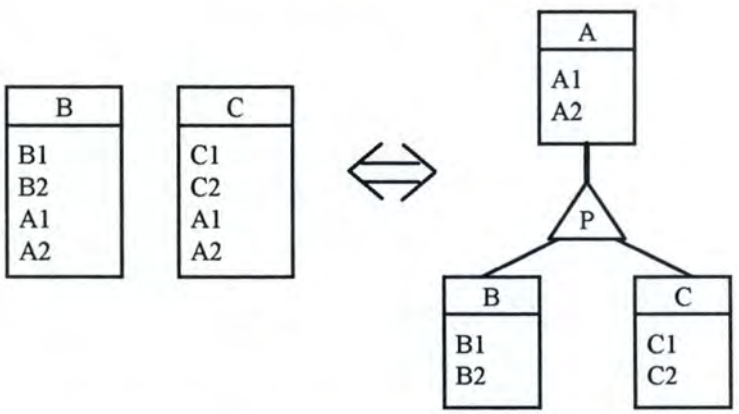


T - 17 : Fusion des deux parties d'un type d'entités



T - 18 : Transformation d'un type d'associations « one-to-one » en relation de sous-typage

La présence d'attributs communs à plusieurs types d'entités peut également suggérer un partitionnement d'un type d'entités en deux sous-types (voir transformation T - 19).



T - 19 : Partitionnement d'un type d'entités

La normalisation conceptuelle termine le processus de conceptualisation des structures de données. A la fin de cette étape, un schéma conceptuel normalisé est disponible.

Cette méthode de rétro-ingénierie des bases de données est celle qui sera utilisée lors de l'étude de cas présentée dans ce mémoire. Une analyse critique de la méthode sera faite ensuite à la lumière des problèmes rencontrés lors de son application au problème traité.

Ces deux méthodes utilisent des transformations de schéma. Avant de continuer, nous allons nous attarder quelque peu sur cette notion.

2.3 Les transformations de schéma

Les transformations de schéma sont omniprésentes dans le domaine des bases de données. Elles servent à prouver l'équivalence de deux schémas, à raffiner un schéma conceptuel, à intégrer deux schémas, à produire un schéma conforme à un SGBD donné,...

Nous introduirons le concept de transformation de schéma en général. Les aspects particuliers à certaines utilisations ont déjà été abordés lors de l'exposé de la méthode de rétro-ingéniérie des bases de données. Les transformations les plus fréquemment appliquées lors des étapes de cette méthode ont également été citées à ce moment.

2.3.1 Définition générale

Une transformation est composée de deux correspondances T et t.

T est une correspondance « structurelle » qui remplace une structure C d'un schéma S en une autre structure C', ce qui est noté C' = T(C). C et C' sont en fait des classes de constructions qui peuvent être définies en terme de prédicats structurels. Dès lors, T peut également être défini par une précondition P, que doit vérifier toute construction C pour pouvoir être transformée par T, et une postcondition Q, que doit vérifier T(C). T est en quelque sorte la syntaxe de la transformation.

t est une correspondance « au niveau des instances » qui indique comment produire l'instance de T(C) correspondant à l'instance valide de C. Si c est l'instance de C, c' est l'instance correspondante de C', ce qui est noté c' = t(c). t est en quelque sorte la sémantique de la transformation.

Une transformation sera donc notée $\Sigma = (T, t)$ ou encore $\Sigma = (P, Q, t)$.

2.3.2 Réversibilité d'une transformation

Une transformation est réversible lorsque les deux constructions C et C' expriment les mêmes concepts, avec la même signification. Le seul changement se situe au niveau de la présentation, la syntaxe.

Une transformation $\Sigma_1 = (T_1, t_1) = (P_1, Q_1, t_1)$ est réversible si et seulement si il existe une transformation $\Sigma_2 = (T_2, t_2) = (P_2, Q_2, t_2)$ telle que pour toute construction C et pour toute instance c de C, on ait

$$P_1(C) \Rightarrow \left[T_2(T_1(C)) = C\right] \text{ et } \left[t_2(t_1(c)) = c\right]$$

Dans ce cas, Σ_2 est la transformation inverse de Σ_1. Toutefois, l'affirmation inverse n'est pas nécessairement vraie.

Si la transformation Σ_2 est également réversible, alors Σ_1 (et Σ_2) sont symétriquement réversibles. Dans ce cas, $\Sigma_2 = (Q_1, P_1, t_2)$ et les deux transformations peuvent être notées de

façon unique $\Sigma = (P, Q, t_1, t_2)$. On dit alors que Σ est une transformation SR, pour symétriquement réversible.

2.3.3 Exemples

La Figure 12 nous montre un exemple de transformation non-réversible. Il s'agit de la projection d'un type d'entité sur deux sous-ensembles de ses attributs.

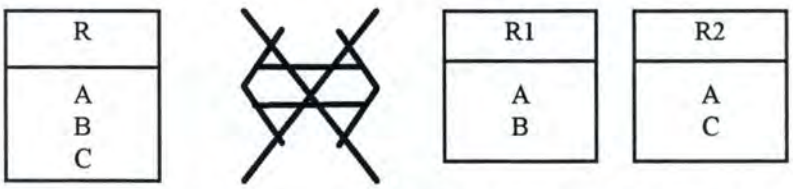


Figure 12 - Exemple de transformation non réversible

Cette transformation n'est pas réversible. En effet, dans le cas suivant, la composée des transformations produira un ensemble de triplets différent de celui de départ :

R = {(a1, b1, c1), (a1, b2, c2), (a2, b1, c2)}

↓

R1 = {(a1, b1), (a1, b2), (a2, b1)}
R2 = {(a1, c1), (a1, c2), (a2, c2)}

↓

R1 * R2 = {(a1, b1, c1), **(a1, b1, c2), (a1, b2, c1),** (a1, b2, c2), (a2, b1, c2)} ≠ R

On remarque que deux triplets ont fait leur apparition.

La Figure 13 nous montre un exemple de transformation simplement réversible (c'est-à-dire réversible, mais pas symétriquement réversible). Il s'agit à nouveau de la projection sur un sous-ensemble d'attributs, mais dans le cas où le type d'entités R est le siège d'une dépendance multivaluée.

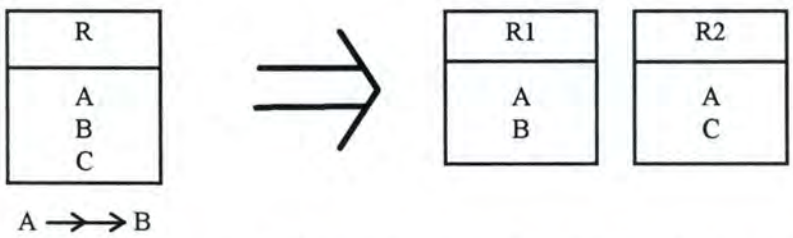


Figure 13 - Exemple de transformation simplement réversible

Cette transformation est réversible, car la dépendance multivaluée nous assure de retrouver les mêmes triplets que R après avoir appliqué la composée des transformations. Toutefois, elle n'est pas symétriquement réversible, car dans le cas où R1[A] ≠ R2[A], on créera des triplets supplémentaires lors de l'application des transformations. On peut le voir dans le cas suivant :

R = {(a1, b1, c1), (a1, b2, c1), (a2, b1, c1), (a2, b2, c1)}

↓

R1 = {(a1, b1), (a1, b2), (a2, b1), (a2, b2)}
R2 = {(a1, c1), (a2, c1)}

↓

R1 * R2 = {(a1, b1, c1), (a1, b2, c1), (a2, b1, c1), (a2, b2, c1)} = R

La transformation étant réversible, on retrouve les triplets de R.

R1 = {(a1, b1), (a1, b2)}
R2 = {(a1, c1), (a2, c1)}

↓

R1*R2 = {(a1, b1, c1), (a1, b2, c1)}

↓

R1*R2[A, B] = {(a1, b1), (a1, b2)} = R1
R1*R2[A, C] = {(a1, c1)} ≠ R2

On remarque que le couple (a2, c1) n'est plus présent, ce qui montre que la transformation n'est pas symétriquement réversible.

Un exemple de transformation symétriquement réversible est donné à la Figure 14. Il s'agit de la transformation de projection-jointure complète : il existe une dépendance multivaluée dans R et les projections de R1 et de R2 sur A sont égales. Dans ce cas, les deux transformations sont inverses l'une de l'autre.

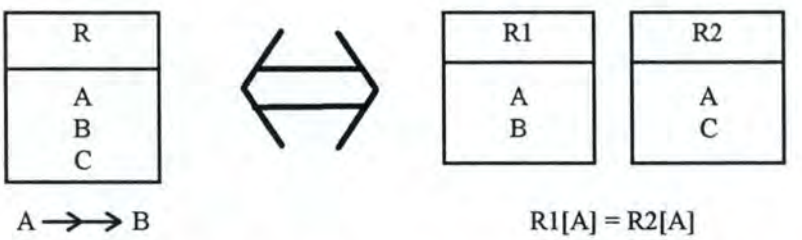


Figure 14 - Exemple de transformation symétriquement réversible

Les transformations citées dans l'exposé de la méthode de rétro-ingénierie sont toutes des transformations symétriquement réversibles. Si tel n'était pas le cas, des ambiguïtés naîtraient lors de l'application d'une transformation : on ne serait pas certain que schéma conceptuel soit équivalent au schéma brut.

Il nous reste à introduire l'atelier DB-MAIN, qui implémente de façon assez directe l'ensemble des concepts dont nous venons de parler.

2.4 L'atelier DB-MAIN

DB-MAIN, qui tire son appellation de « DataBase MAINtenance », est un projet en cours aux Facultés Universitaires Notre-Dame de la Paix depuis 1993. Le but est de construire un outil-CASE de support à l'ingénierie, la rétro-ingénierie, la ré-ingénierie et la maintenance des bases de données. Il est donc le plus neutre possible vis-à-vis des différents modèles de données (relationnel, COBOL, IMS,...) et est pensé dans l'esprit des méthodes de conception et de rétro-ingénierie présentées dans les deux sections précédentes.

Nous allons présenter le modèle de spécification de l'atelier ainsi que ses fonctionnalités principales.

2.4.1 Le modèle de spécification

Un des buts de DB-MAIN est la généricité : il ne doit pas être « orienté relationnel, COBOL,...» et doit pouvoir supporter un nombre important de modèles utilisés en pratique. Pour cela, un modèle générique de spécification des structures de données est utilisé. Il est organisé d'une façon similaire à celle du modèle entité-association, auquel on a ajouté des structures supplémentaires.

2.4.1.1 Le projet

L'élément de base est le projet. Toutes les informations concernant un projet sont contenues dans un fichier dont l'extension est « LUN ». Un projet comprend un ou plusieurs schémas et éventuellement des fichiers de texte (code source de programmes, rapports descriptifs,...).

2.4.1.2 Le schéma

Un schéma décrit un ensemble de structures de données. Il peut provenir éventuellement d'une copie d'un autre schéma. Un schéma est composé de types d'entités, de types d'associations et de collections. Sur la Figure 15, le schéma est « BIBLIOTHEQUE/1 »[3] représente l'ensemble des objets dessinés. Il est également représenté en haut de celui-ci, dans un ovale.

2.4.1.3 Le type d'entités

Un type d'entités représente une classe d'entités du monde réel, mais peut également représenter des structures propres à un modèle de données, telles que des types d'enregistrements, des tables, des segments, ...

Un type d'entités peut être un sous-type d'un ou plusieurs autres types d'entités. On dit que la collection des sous-types d'un type d'entités E est totale si chaque entité E doit appartenir à un de ses sous-types. La même collection est dite exclusive si une entité d'un sous-type de E ne

[3] La seconde partie du nom (se trouvant après le caractère '/') est la version du schéma : celle-ci peut être un nombre ou un mot quelconque (par exemple 'Conceptuel').

peut appartenir à un autre sous-type de E. Lorsque cette collection est totale et exclusive, elle forme une partition du type d'entités E.

Un type d'entités peut contenir des attributs et des contraintes (via les groupes), il peut apparaître dans des types d'associations et peut faire partie d'une collection.

Un exemple de type d'entités se trouve dans la Figure 15 plus loin dans ce chapitre : il s'agit du type d'entités « OUVRAGE ». Les types d'entités sont reconnaissables à leur forme rectangulaire.

2.4.1.4 La collection

Une collection est un ensemble d'entités[4]. Elle est utilisée pour représenter un fichier, un espace de stockage,... Elle comprend des entités de plusieurs types d'entités différents, et des entités d'un même type peuvent être présentes dans des collections différentes.

Il n'y a pas de collection représentée dans la Figure 15. Sa représentation graphique ressemble au dessin d'un disque, tel qu'il est représenté habituellement. On peut la voir sur le bouton situé à l'extrême droite de la barre d'outils.

2.4.1.5 Le type d'associations

Un type d'associations représente une classe d'associations entre entités. Il est composé de types d'entités qui apparaissent chacun au sein d'un rôle, qui est caractérisé également par sa cardinalité. Celle-ci se présente sous la forme d'un couple d'entiers [i-j]. Ce couple représente le nombre minimal et maximal de rôles que peut jouer chaque entité du type d'entités concerné par ce rôle. Un type d'associations peut en outre comprendre des attributs et peut faire l'objet de contraintes, via les groupes.

Sur la Figure 15, on peut remarquer, par exemple, le type d'associations « EMP_CLOTURE ». Les types d'associations sont reconnaissables à leur forme typique : un diamant.

2.4.1.6 L'attribut

Un attribut représente une propriété d'un type d'entités ou d'un type d'associations. Il existe deux sortes d'attributs : les premiers possèdent un domaine de valeurs et sont caractérisés par un type de données (numérique, caractère, booléen, date,...) et une longueur (1, 2, .. ,N). Ces attributs sont appelés atomiques. Les autres attributs, appelés attributs décomposables, sont composés d'autres attributs, atomiques ou décomposables. Leur longueur vaut la somme des longueurs de leurs composants. Chaque attribut, atomique ou décomposable, est caractérisé par sa cardinalité. Celle-ci se présente sous la forme d'un couple de valeurs [i - j] et indique que le parent de l'attribut (le type d'entités, le type d'associations ou l'attribut décomposable qui le possède) contient entre i et j valeurs de celui-ci. Selon la valeur de sa cardinalité, un attribut est appelé :

- monovalué si j = 1
- multivalué si j > 1

[4] Remarquons qu'il s'agit d'entités et pas de types d'entités.

- facultatif si i = 0
- obligatoire si i = 1

Un attribut fait partie d'un type d'entités, d'un type d'associations ou d'un autre attribut (décomposable).

Sur la Figure 15, l'attribut « ADRESSE » est un attribut décomposable du type d'entités « EMPRUNTEUR ».

2.4.1.7 Le groupe

Un groupe représente une construction attachée à un objet (un type d'entités ou un type d'associations). Il est utilisé pour représenter les contraintes relatives à l'objet en question (identifiants, contraintes de référence ou autres,...). Un groupe est composé d'attributs et/ou de rôles et/ou d'autres groupes, tous appartenant à l'objet auquel il est rattaché.

Les groupes représentés sur la Figure 15 ont tous la propriété de définir un identifiant. C 'est le cas, par exemple, du groupe formé de l'attribut « MATRICULE », identifiant du type d'entités « EMPRUNTEUR ».

2.4.1.8 Autres caractéristiques

Des caractéristiques communes à tous les objets du modèle existent. Ainsi, les schémas, les types d'entités, les collections, les attributs, les types d'associations ont un nom, peuvent avoir une abbréviation, une description sémantique et une description technique.

Celles-ci sont accessibles en cliquant deux fois sur l'objet désiré.

Les objets décrits ci-dessus permettent d'exprimer un grand nombre de contructions de la plupart des modèles de données existants. Nous allons maintenant décrire l'interface de l'atelier ainsi que ses principales fonctionalités.

2.4.2 L'interface de DB-MAIN

L'interface d'un outil tel que celui-ci est très importante car elle est à la fois un moyen de visualisation et un moyen d'action sur les objets étudiés. Celle de DB-MAIN comporte deux modes : un mode textuel et un mode graphique. Le mode textuel permet de travailler sur la liste des objets, décrits sous forme textuelle. Plusieurs options sont disponibles pour faciliter la tâche de l'utilisateur : vue compacte, normale, étendue ou par ordre alphabétique. Le mode graphique permet de travailler directement sur les objets. Plusieurs options sont également disponibles : vue avec ou sans attributs, une option « auto-draw » qui place les objets de façon automatique et une option « independant » qui permet de déplacer les objets indépendamment les uns des autres. La Figure 15 donne un exemple de présentation en mode graphique. On peut remarquer en passant qu'une barre d'outils permet de créer un nouveau projet, de charger et de sauver un projet existant. Elle permet également de passer du mode texte au mode graphique, d'ajouter directement des structures telles que des types d'entités, des types d'associations, des rôles, des collections,...

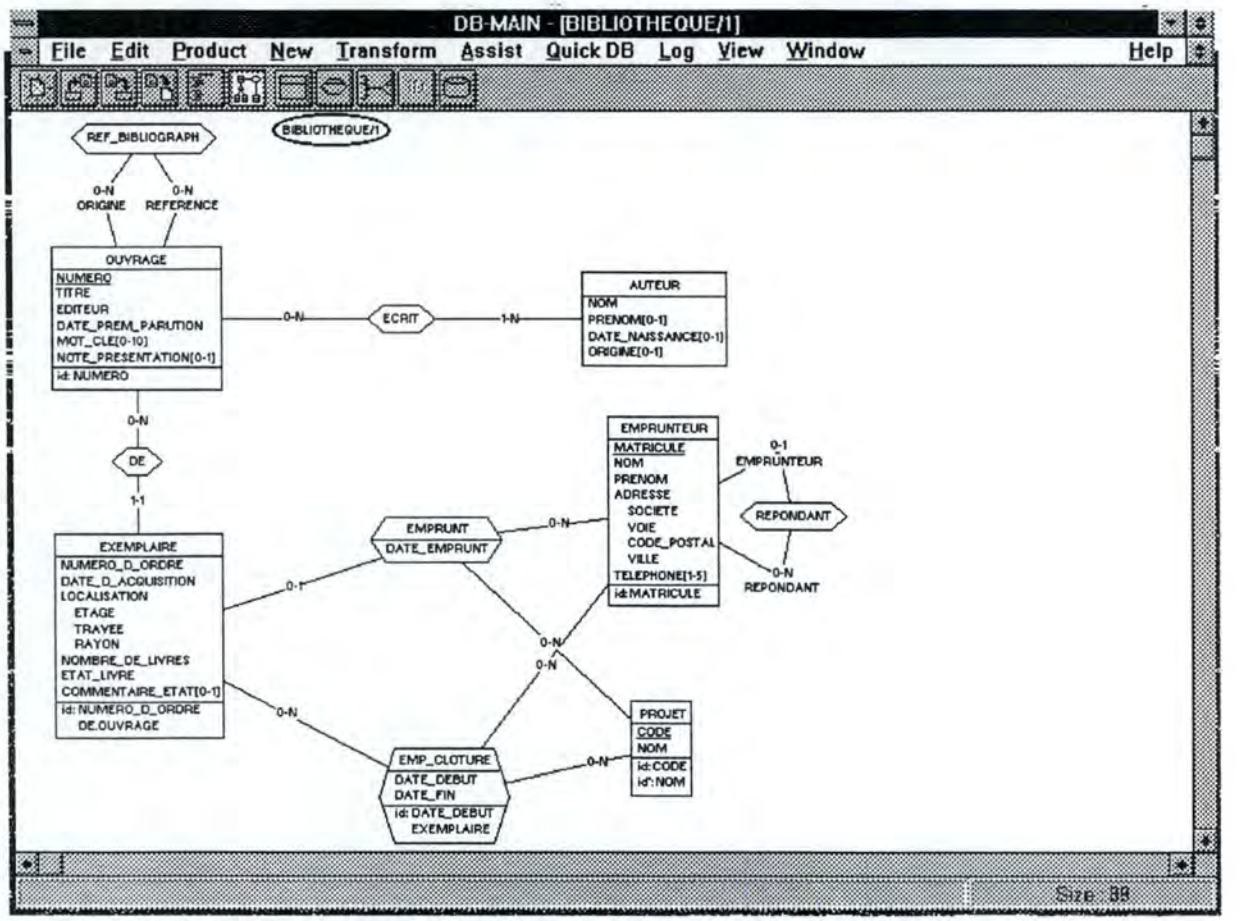


Figure 15 - L'interface graphique de DB-MAIN

Nous allons parler maintenant des différentes fonctionnalités de DB-MAIN, afin de nous faire une idée précise de l'outil qui sera évalué dans ce mémoire.

2.4.3 Les fonctionnalités de DB-MAIN

Les fonctionnalités principales de DB-MAIN se situent au niveau des transformations de schéma. Celles-ci interviennent lors des processus de conception et de rétro-ingénierie des bases de données que nous avons développés ci-dessus. Les autres fonctionnalités importantes sont des assistants, permettant d'appliquer plusieurs transformations simultanément, le « parser » de code source et les facilités de « pattern matching ».

2.4.3.1 Les transformations élémentaires

Les transformations élémentaires consistent à appliquer une transformation T à un objet particulier O. L'atelier supporte un grand nombre de ces transformations : elles sont la base de toute évolution d'un schéma de données. Voici, sous forme de liste, les transformations les plus usuelles présentes dans l'atelier.

- Transformation concernant les types d'entités
  - Transformation d'un type d'entités en type d'associations (transformations T - 14, T - 15 et T - 16)
  - Transformation d'un type d'entités en attribut (transformations T - 1, T - 2, T - 7 et T - 8)
  - Transformation d'une relation IS-A en type d'associations « one-to-one » et inversément (transformation T - 18)
  - Eclatement et fusion d'un type d'entités (transformations T - 13, **Erreur! Source du renvoi introuvable.** et T - 19)
  - Ajout d'un identifiant primaire technique

- Transformation concernant les types d'associations
  - Transformation d'un type d'associations en type d'entités (transformations T - 14, T - 15 et T - 16)
  - Transformation d'un type d'association en attribut de référence (transformation T - 9)

- Transformations concernant les attributs
  - Transformation d'un attribut en type d'entités (représentation par les instances ou par les valeurs) (transformations T - 1, T - 2, T - 7 et T - 8)
  - Désagrégation d'un attribut décomposable (transformation T - 5)
  - Concaténation d'un attribut décomposable (transformation T - 6)
  - Instanciation d'un attribut multivalué (transformation T - 4)
  - Concaténation d'un attribut multivalué (transformation T - 3)
  - Passage d'un attribut monovalué à un attribut multivalué

- Transformations concernant les groupes d'attributs et/ou de rôles
  - Transformation d'un attribut de référence en type d'associations transformation T - 9)
  - Transformation d'un groupe d'attributs en attribut décomposable (transformation T - 5)

- Transformation des noms
  - Changement, ajout ou retrait d'un préfixe aux noms
  - Remplacement du nom, ou d'une partie du nom, d'objets sélectionnés

On remarquera que ces transformations sont celles qui sont reprises dans l'exposé de la méthode de rétro-ingénierie. Celles qui ne se trouvent pas dans l'atelier sont en général triviales, comme par exemple l'élimination d'un type d'associations redondant, ou bien contiennent des structures qui ne sont pas représentables dans l'atelier, comme les dépendances fonctionnelles dans le cas de la transformation. De plus, l'atelier propose des transformations concernant les noms (notamment au niveau des préfixes), l'ajout d'un identifiant primaire technique, comme c'est parfois utile dans le cas du modèle relationnel ou COBOL et le passage d'un attribut monovalué à multivalué. Ces transformations sont regroupées dans le menu « Transform » de l'atelier.

2.4.3.2 Application simultanées de plusieurs transformations

La possibilité d'appliquer plusieurs transformations simultanément existe également. Il suffit de générer un script, via la boîte de dialogue présentée à la Figure 16, appelée grâce à l'item « Global transformation » du menu « Assist ».

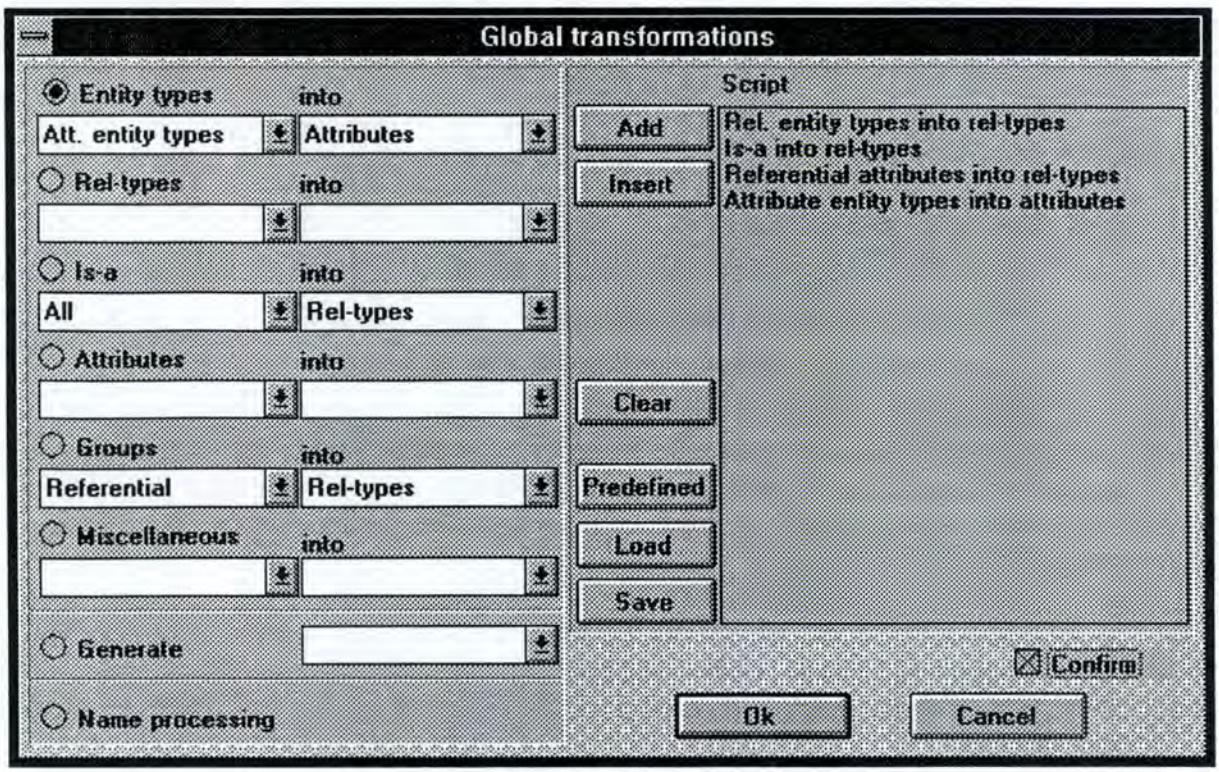


Figure 16 - Génération d'un script de transformations

En plus de ces transformations de schéma, l'atelier offre d'autres fonctionnalités qu'il est utile de citer.

2.4.3.3 Le « parser » de code source

Le « parser » de code COBOL, SQL, CODASYL et IMS permet de créer automatiquement, à partir du texte DDL, le schéma correspondant au code source. Ceci permet d'éviter notamment toutes les erreurs d'encodage et de gagner un temps considérable, surtout lorsque la base de données contient un nombre important de fichiers et de champs.

2.4.3.4 Les facilités de « pattern matching »

Le « pattern matching » recherche, dans des textes externes comme le code source d'un programme, des « patterns » particuliers. Cette recherche permet de retrouver plus facilement certaines contraintes dans le code source des programmes. Pour utiliser ce module, on commence par ajouter un fichier au projet courant, via l'item « Add file... » du menu « Product ». Ensuite, on définit un ou des pattern(s) que l'on veut retrouver dans le fichier. Pour cela, on utilise un éditeur de texte neutre, comme « Notepad » ou « Emacs ». L'annexe 1 présente la syntaxe des patterns. Une fois le pattern définit, on sauve le fichier. Dans

DB-MAIN, on charge les patterns définis grâce à l'item « Load pattern... » du menu « File », qui fait apparaître la boîte de dialogue de la Figure 17.

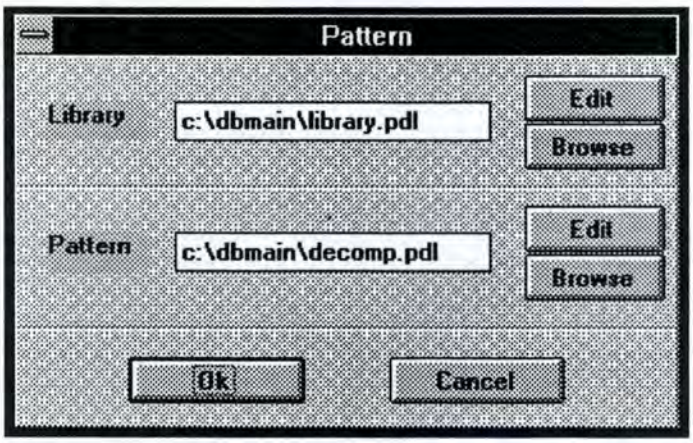


Figure 17 - Chargement d'un fichier de patterns

Le fichier dont le nom de trouve en face de « Library » contient une librairie de patterns de base utilisés pour en construire d'autres. Il s'agit des identificateurs, des séparateurs,... définis une fois pour toute. L'autre fichier contient les patterns définis spécifiquement pour l'usage que l'on veut en faire.

Lorsque les fichiers de définition des patterns sont chargés, il reste à ouvrir le fichier texte et à commencer les recherches. Celles-ci s'effectuent en sélectionnant l'item « Search - Source » du menu « Edit ». A ce moment, la boîte de dialogue de la Figure 18 permet de choisir le pattern que l'on veut utiliser.

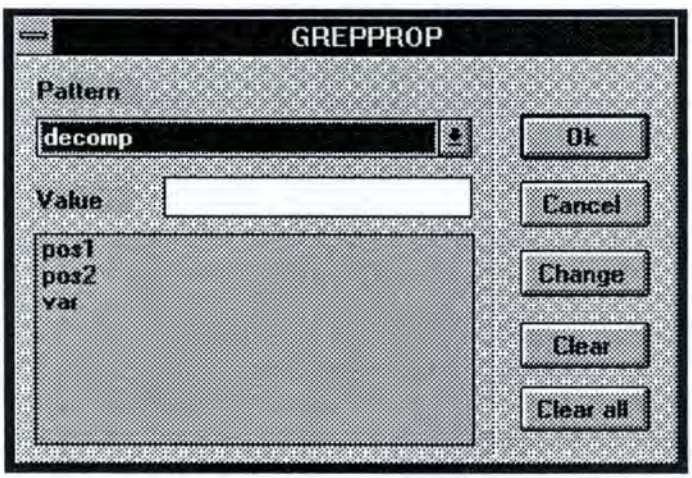


Figure 18 - Sélection d'un pattern

Après avoir sélectionné le pattern voulu, on peut forcer une valeur dans les patterns variables affichés dans la zone grisée sous l'item « Value ». Ceci permet de restreindre la recherche aux patterns dont une partie est fixée. Par exemple, la pattern « decomp » possède trois parties variables : « pos1 », « pos2 » et « var ». Si on ne veut que les patterns dont la partie « var » contient le mot « attribut », on changera la valeur de « var » en le sélectionnant, en tapant la valeur désirée dans la zone de saisie à droite de « Value » et en cliquant sur le bouton « Change ». Le bouton « Clear » permet d'effacer une valeur introduite dans la partie variable

sélectionnée et le bouton « Clear all » permet d'effacer les valeurs de toutes les parties variables.

Après avoir défini les paramètres de recherche, on clique sur « OK » et l'atelier s'arrête sur la première ligne satisfaisant la définition donnée. En appuyant sur la touche « F3 », on relance la recherche. Cet outil a été testé lors de l'étude de cas qui suit ce chapitre.

La description de l'atelier n'est pas exhaustive, et tel n'est pas le but de ce mémoire. Le lecteur intéressé peut se référer à l'atelier lui-même, à [HAINAUT, 1994] ou encore à [HAINAUT, 1995a].

Nous allons maintenant tester cette méthode lors d'une étude de cas. Cette dernière résulte d'un problème posé par une entreprise, que nous allons décrire dans le chapitre suivant.

Chapitre 3 : Enoncé du problème

3.1 Le contexte du problème

Le problème sous-jacent à ce mémoire a été posé par les départements « achats » et « IS » (Information System) de l'entreprise Baxter. Il s'agit d'un problème du monde économique, ce à quoi ont rarement accès les universités en Belgique.

Baxter est une entreprise internationale présente dans le monde entier. Son activité principale est la production et la vente de matériel et de produits médicaux. Sa clientèle se retrouve essentiellement parmi les hôpitaux. En Belgique, elle possède une usine à Lessines, un entrepôt à Ternat, une unité de Recherche et Développement à Nivelles et des bureaux à Evere. C'est une entreprise qui donne une importance primordiale à la qualité de ses produits et de ses services. Elle est d'ailleurs certifiée ISO9002.

3.1.1 Le département « achat »

Les achats d'une entreprise telle que Baxter représentent une partie importante de son activité. Toutefois, les départements diffèrent selon les pays, au niveau de leurs compétences, leur importance et leur pouvoir...

Le département « achat » de l'usine belge est très important. En effet, il n'est pas seulement responsable de l'approvisionnement des produits utilisés dans le processus de fabrication, mais de tout ce qui est acheté à l'extérieur de l'usine : de l'hydrogène liquide au bic de la secrétaire en passant par l'ordinateur central et les chaises de bureau. Cette manière de procéder permet de garder une trace de tout ce qui entre dans l'usine. Les différents produits achetés sont divisés en trois grandes classes : les matières premières (Raw material), appelés également « codifiés » (car ils sont tous repris dans la base de données et possèdent donc un code), les produits dits MRO (Maintenance Requirement and Operation supplies) et les produits non-codifiés. Le premier groupe de produits (Raw material) comprend les matières premières du processus de fabrication. Les produits MRO comprennent toutes les pièces de rechange et de maintenance des différentes machines de l'usine. Enfin, les produits non-codifiés comprennent toutes les autres fournitures (matériel de bureau,...)

Les deux premiers types de produits font partie d'une organisation informatisée. Les premiers sont utilisés en production et représentent la partie la plus coûteuse : l'effort est donc principalement porté sur les négociations avec les fournisseurs au niveau des prix, des termes de paiement et de livraison,... Les seconds font l'objet d'une organisation minutieuse : un programme informatique gère complètement le stock des pièces de rechange de l'usine. Les codes des produits sont répartis en 113 groupes, eux-mêmes divisés en sous-groupes. Une étiquette d'identification de la localisation accompagne chaque exemplaire d'un produit. Celle-ci contient le magasin, la rangée d'étagères, l'armoire, la rangée dans l'armoire et le tiroir dans lequel l'exemplaire se trouve. Lorsqu'une pièce est utilisée, le système met à jour le stock et avertit le cas échéant le responsable de la nécessité de passer une commande, via un système

de stock mini et maxi (nombre minimal et maximal d'exemplaires de la pièce devant se trouver dans l'usine).

Ce département est composé de 8 personnes : le directeur des achats, sa secrétaire, une personne responsable par type de produits et trois employés. Le directeur a en outre la responsabilité de la coordination des différents départements « achat » en Europe et fait également partie du conseil d'administration de la société.

Les départements « achat » des autres pays ne sont généralement pas aussi bien organisés et n'ont pas tous autant de responsabilités : certains ne s'occupent que des matières premières, d'autres ne s'occupent pas du suivi des fournisseurs,...

Nous allons maintenant nous attarder un peu sur le fonctionnement de ce département, afin de clarifier le contexte dans lequel le problème se pose.

L'activité principale d'un département « achat » est la gestion des bons de commandes. Il s'agit de déterminer les droits et les devoirs en la matière et d'en contrôler l'usage.

Tout d'abord, n'importe qui ne peut pas passer une commande de n'importe quoi. En outre, des acheteurs doivent valider chaque commande avant que celle-ci ne soit envoyée au fournisseur concerné. Il existe quatre acheteurs : le directeur des achats (pour les matières premières) ainsi qu'un pour chaque type de produits (matières premières, MRO et non-codifiés). Pour chaque type, il existe une liste de personnes habilitées à passer une commande. Cette liste est plus ou moins restreinte pour les deux premiers types, contrairement au troisième pour lequel n'importe qui peut passer une commande. De plus, un système de gestion de stock des matières premières génère des commandes automatiques en fonction des besoins de l'usine, qui sont plus ou moins constants. Le système de gestion du stock de pièces détachées (MRO) propose également des commandes de pièces. Seules les commandes de matériel non-codifié n'est pas automatique, et ne pourrait d'ailleurs pas l'être, vu que les besoins sont imprévisibles pour ce genre de matériel...

Lorsqu'une commande est passée à un fournisseur, ce dernier renvoie un accusé de réception dans lequel il confirme les conditions auxquelles est passée la commande (termes de paiement et de livraison,...). Ces accusés de reception font partie du système informatique de gestion des commandes, ce qui permet de les comparer aux commandes et d'y déceler des éventuelles divergences. Quand la marchandise arrive, elle fait l'objet d'un vérification et les résultats des divers contrôles (quantités, date d'arrivée, qualité de la marchandise,...) sont également encodés. Grâce à ceux-ci, les fournisseurs sont suivis de très près et le moindre écart aux conditions définies au préalable est immédiatement signalé. Les fournisseurs sont cotés en fonction du respect de ces conditions : le département « achat » calcule un « rating », dans lequel interviennent les différents paramètres faisant la qualité d'un fournisseur. Ces derniers peuvent alors devenir approuvés, certifiés ou intégrés pour un type de produit. Ces trois stades consécutifs sont liés à une qualité croissante du fournisseur, conjointement à une diminution des contrôles effectués sur la marchandise à son arrivée.

Ce sont là les caractéristiques principales du département. Il en existe d'autres (gestion de contrats d'approvisionnement, de marchandises en transit vers une autre usine,...) mais qui ne sont pas primordiales dans la compréhension du problème, que nous allons maintenant préciser.

3.2 Le problème

Depuis quelques années, un processus d'intégration est lancé en Europe. Il s'agit de regrouper des données à un niveau européen afin de développer une politique commune. Dans ce cadre, le Directeur des achats de l'usine belge veut consolider les informations contenues dans les bases de données des différentes compagnies. Ces données serviront à analyser la qualité du travail des fournisseurs, à éliminer les moins performants, à négocier des contrats à un niveau européen, à faire des analyses et des statistiques, à cibler les actions de réduction des coûts,... De plus, elles formeront la base numérique d'un rapport trimestriel concernant les achats en Europe. La Figure 19 indique le résultat final de ce projet.

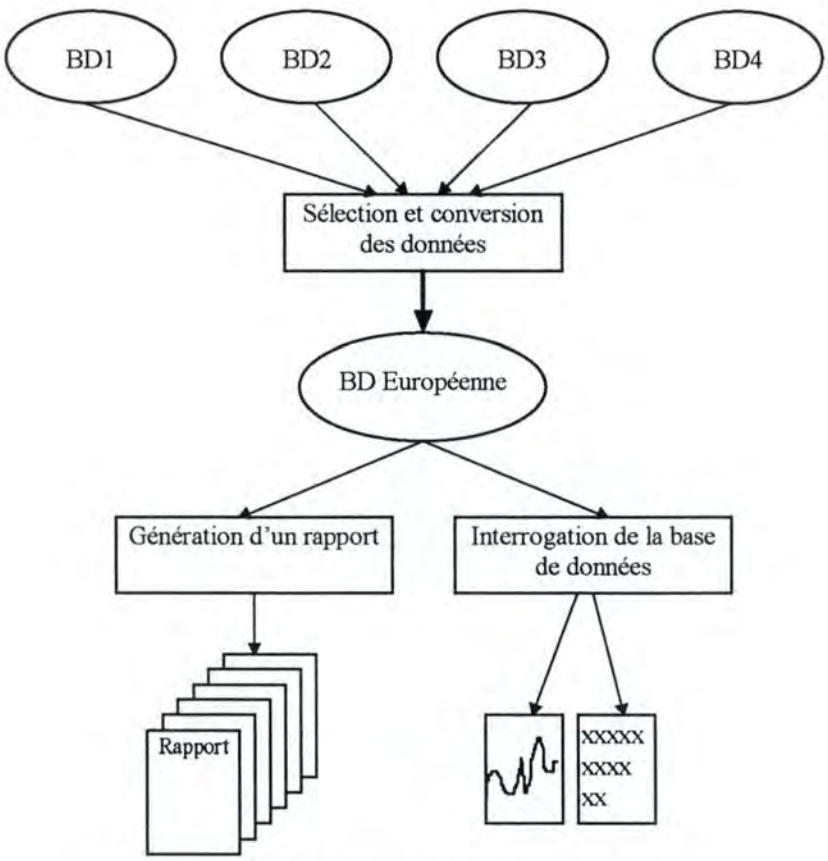


Figure 19 - Le projet de l'entreprise Baxter

Pour obtenir ce résultat, il faut procéder à l'analyse des bases de données existantes, construire la base de données européenne, définir des protocoles de conversion des données et construire un générateur de rapport et un logiciel d'interrogation de la base de données. La Figure 20 reprend ces différentes étapes.

Toutefois, vu le temps disponible, nous nous limiterons dans ce mémoire à l'étude d'une seule base de données. En effet, ceci nous permettra de tester et de critiquer la méthode d'analyse décrite à la section 2.2. La base choisie est celle de la Belgique. Elle a l'avantage d'être complète, facilement accessible, mais elle est également une des plus anciennes et ayant subi un grand nombre de modifications, ce qui est toujours intéressant dans ce type d'études de cas. Etant donné que la base de données belge est complète, le résultat de son analyse peut servir de support sur lequel pourront venir se greffer les résultats de l'analyse des autres bases. De plus, la base de données européenne ne doit contenir qu'une partie des informations contenues

dans les différentes bases de données. Il suffira alors de dériver une vue ne reprenant que les structures intéressantes pour voir apparaître une représentation brute de la base de données européenne.

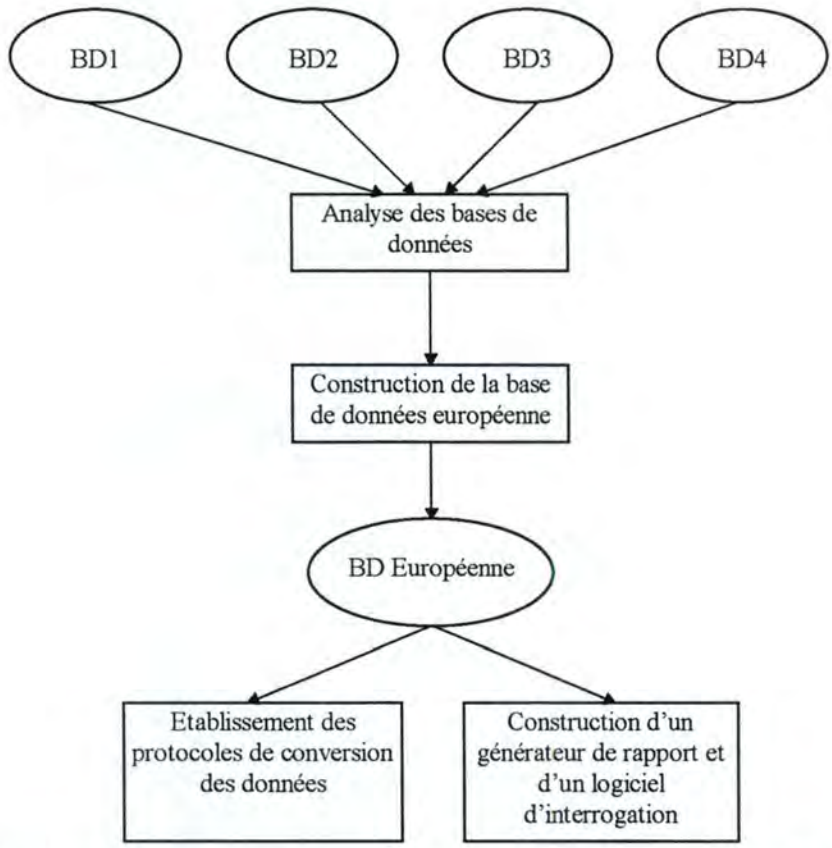


Figure 20 - Description des étapes de résolution du problème posé

3.3 La configuration de travail

Nous allons décrire dans cette section la configuration informatique dans laquelle se trouve la base de données belge. Les éléments principaux de cette configuration comprennent le hardware, le SGBD et les programmes.

Le matériel de base utilisé par Baxter en Belgique est composé de deux ordinateurs IBM AS400, de terminaux et de PC. Un réseau local relie toutes ces machines entre-elles et permet également de se connecter vers l'extérieur, principalement sur une machine d'un autre site de Baxter dans le monde. La base de données à laquelle nous nous intéressons se trouve sur un AS400.

Le SGBD utilisé est celui de l'AS400. Nous allons nous intéresser à ses caractéristiques ayant un lien avec le travail auquel nous allons nous livrer sur la base de données.

Il existe tout d'abord deux types de fichiers : les fichiers physiques et les fichiers logiques. Les premiers contiennent les enregistrements proprement dits. Une description de chacun de ceux-ci est disponible sous forme d'un fichier texte se contentant d'énumérer les champs faisant partie du fichier. Le format de chaque champ est défini comme une référence à un enregistrement d'un fichier décrivant des formats de champ. Un texte de description du champ

est également présent. Les fichiers logiques, quant à eux, contiennent les index sur les fichiers physiques. Chaque fichier logique correspond à un seul fichier physique. Une description des champs composant l'index est également disponibles sous forme d'un fichier de texte, reprenant la liste des champs, le fait que l'index soit unique ou pas et une ou des condition(s) sur les enregistrements accessibles via cet index.

Il est intéressant de faire remarquer que ce SGBD ne supporte pas les contraintes de référence. Celles-ci seront donc à découvrir via l'analyse des programmes. De plus, rien ne permet de dire si un champ est obligatoire ou pas. Il existe bien une valeur (*BLANKS) correspondant à la valeur NULL, mais rien n'indique dans la description des fichiers si un champ peut prendre cette valeur ou si elle lui est interdite. Nous supposerons donc que tous les champs sont a priori obligatoires.

En résumé, ce SGBD utilise le modèle relationnel à l'exception des contraintes de référence, du caractère facultatif d'un champ et du fait qu'il n'existe pas de moyen pour savoir si un index est primaire ou secondaire. On ne pourra donc pas faire la différence entre différents identifiants pour décider lequel est l'identifiant primaire.

Les programmes utilisant cette base de données sont écrits en RPG. L'annexe ??????? en reprend les caractéristiques dont nous nous sommes servi pour analyser ces programmes. D'une façon générale, on peut dire que ce langage est très rigide syntaxiquement. Chaque partie d'instruction a sa place précise sur la ligne. C'est un langage procédural dans lequel, toutefois, il n'y a pas de paramètres : toutes les variables sont globales.

3.4 La partie abordée dans ce mémoire

La partie du problème à laquelle nous nous intéresserons dans le cadre de ce mémoire est la suivante : appliquer la méthode de rétro-ingénierie décrite à la section 2.2 afin de dériver un schéma conceptuel de la base de données belge. Les chapitres 4 et 5 décriront l'étape d'extraction des données : tout d'abord l'extraction des structures de base, composées des champs de chaque table, décrits dans les fichiers physiques, et de certains identifiants disponibles dans les fichiers logiques, et ensuite la recherche des contraintes dans les codes sources des programmes. Le chapitre 6 décrira alors l'étape de conceptualisation. A chaque étape, nous nous demanderons quel soutien l'atelier DB-MAIN a apporté et quels ajouts on pourrait y faire pour en améliorer la performance et l'utilité.

Chapitre 4 : Les schémas bruts

Nous allons, dans ce chapitre, démarrer le processus de rétro-ingénierie de la base de données belge des achats. Nous nous appuyerons pour cela sur la méthode décrite à la section 2.2. La première étape est l'extraction des structures de données, qui fait l'objet de ce chapitre et du suivant. Le premier concerne l'extraction du schéma global et le suivant présente les résultat de l'analyse des codes cources des programmes analysés. Le chapitre 6 présente la phase de conceptualisation de base et enfin, le chapitre 7 concluera le processus avec la normalisation conceptuelle.

Ce chapitre est donc consacré à la description de l'extraction des structures de données du schéma global de la base de données. Celle-ci fait en deux temps : d'une part la recherche des structures des tables et des champs et d'autre part la recherche des index sur ces tables.

Chaque table correspond à un fichier et chaque fichier correspond à une table. Toutefois, il n'est pas aisé de sélectionner les tables concernées par les achats; en effet, la base de données du département « achats » fait partie de la base de données générale utilisée par tous les départements de l'usine. Nous avons fait appel pour ce choix à la personne responsable de la maintenance de cette partie de la base de données. Elle nous a aimablement indiqué quels sont les fichiers concernés par les achats et a imprimé les descriptions de ceux-ci. Les fichiers retenus sont au nombre de 31.

Nous avons vu qu'il existait deux sortes de fichiers : des fichiers dits « physiques » et des fichiers dits « logiques ». Les fichiers physiques contiennent les enregistrements des tables et les fichiers logiques contiennent les index sur le fichier physique correspondant. Ainsi, le fichier physique contenant les en-têtes de commandes s'appelle « PURMST ». Il lui correspondent 11 fichiers logiques (« PURMST01 », « PURMST02 »,...) contenant les index sur « PURMST ». Nous allons nous intéresser d'abord aux fichiers physiques, de façon à construire un schéma global, et ensuite aux fichiers logiques, qui viendront compléter ce schéma avec des identifiants.

4.1 L'extraction du schéma global

Nous nous intéressons ici aux fichiers physiques uniquement. Ceux-ci fournissent le nom des champs, leur type (chaîne de caractères ou nombre), leur longueur, leur position dans le fichier et leur description. Ces renseignements permettent d'avoir déjà une certaine idée de ce que seront les identifiants, les contraintes référencielles,... En effet, les fichiers ont une structure assez « standard » : une commande possède un numéro, le numéro du fournisseur d'une commande référence le numéro du fournisseur dans le fichier fournisseurs,... Mais nous n'en sommes qu'au stade des suppositions et seule l'analyse des programmes de maintenance de ces fichiers pourra nous apporter les preuves des hypothèses formulées.

Le Tableau 1 nous présente la liste des 31 fichiers retenus, acompagnés de leur description. Le vocabulaire employé se réfère à la section 3.1 dans laquelle nous avons décrit sommairement le fonctionnement du département « achat » de l'usine belge.

Tableau 1 : Les fichiers physiques

Nom du fichier	Signification	Description
BUYER	Buyer	décrit les acheteurs
CALNDR	Calendar	calendrier des jours fériés, périodes de fermeture,...
CMPANY	Company	fichier des compagnies
COUNTR	Counter	compteur pour les n° de commandes
CURCOD	Currency Code	décrit les devises
DELMTH	Delivery Method	méthodes de livraison
DELTRM	Delivery Terms	termes de livraison
HDRCOM	Header Comment	commentaires des en-têtes de commandes
IPSP01	Supplier Approbation	décrit les fournisseurs approuvés (type et date d'approbation,...)
IPSP02	Standard Comments	commentaires généraux concernant l'usine
IPSP03	Payment Code	codes de paiement
IPSP06	Control File	n° des commandes référencées par un programme
IPSP08	Status code	code des statuts d'approbation des fournisseurs
LNGCD	Language Code	langues utilisées par les fournisseurs
PARTMS	Part Master	produits
PAYTRM	Payment Terms	termes de paiement
PMCREF	Part Master Cross Reference	table de mise en correspondance des numéros de produits
POPDHD	Price Deal Header	liste de prix des fournisseurs
POPDQP	Deal Quantity/Price	remises accordées par les fournisseurs à l'achat d'une certaine quantité
PRTCOM	Supplier/Part Comments	commentaires à propos d'un fournisseur et d'un produit
PURADD	Supplier Address	adresses des fournisseurs
PURCOM	Purchase Order Comments	commentaires à propos d'une commande
PURITM	Purchase Item	lignes de commandes
PURLED	Lead Time	délai de livraison
PURMST	Purchase Master	en-têtes de commandes
PURVEN	Purchase Vendor	fournisseurs
T$MA31#1	Currency Conversion Table	table de conversion des devises
UNITMS	Unit of Measure	unités de mesure
VATCTL	VAT Control	contrôle des codes de TVA
VENCOM	Vendor Comments	commentaires sur les fournisseurs
WHOUSE	Warehouse	lieu de livraison

Voici, à titre d'exemple, le contenu du fichier « BUYER » :

```
1,00      *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2,00      * Source directives for the SYNON/1 compile pre-processor
3,00      *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
4,00      * Title
5,00      *++++++++++++++
6,00      * Pre-Compile
7,00      *++++++++++++++
8,00      * Compiler
9,00      *++++++++++++++
10,00     * Post-Compile
11,00     *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
12,00     A*************************************************************************
13,00     A*                TRAVOPS/38 DATA BASE SPECIFICATIONS                    *
14,00     A*                                                                       *
15,00     A*   PHYSICAL FILE NAME:     BUYER                                       *
16,00     A*                                                                       *
17,00     A*   FULL FILE NAME:         BUYER/ANALYST CODES                         *
18,00     A*                                                                       *
19,00     A*   FILE DESIGNATOR:        Z                                           *
20,00     A*                                                                       *
21,00     A*   SPECIAL NOTES:          (NONE)                                      *
22,00     A*                                                                       *
23,00     A*   LAST CHANGE:        DATE      S.I.R.   PROGRAMMER                   *
24,00     A*                     XX/XX/XX    XXXXXX   XXXXXXXXXXXXXXXXXXXX       *
25,00     A*                                                                       *
26,00     A*************************************************************************
27,00     A*
28,00     A                                     REF(DBREFER)
29,00     A          R ZBUYER                   TEXT('BUYER/ANALYST CODE')
30,00     A*
31,00     A*=======================================================================
32,00     A*
33,00     A            ZBAC      R             REFFLD(BAC)
34,00     A            ZDESCR    R             REFFLD(REFDES)
35,00     A                                     TEXT('BUYER/ANALYST DESCRIPTION')
36,00     A                                     COLHDG('BUYER/ANALYST' +
37,00     A                                     'DESCRIPTION')
38,00     A            ZORTY     R             REFFLD(FLAG)
39,00     A                                     COLHDG('ORDER TYPE')
40,00     A            ZAPPRO    R             REFFLD(FLAG)
41,00     A                                     COLHDG('APPROBATION FLAG Y/N')
42,00     A            WFLAG1    R             REFFLD(FLAG)
43,00     A                                     COLHDG('FREE FLAG 1')
44,00     A            WFLAG2    R             REFFLD(FLAG)
45,00     A                                     COLHDG('FREE FLAG 2')
46,00     A            WFLAG3    R             REFFLD(FLAG)
47,00     A                                     COLHDG('FREE FLAG 3')
48,00     A*
49,00     A*************************************************************************
50,00     A*  END OF SPECIFICATIONS FOR - BUYER FILE                               *
51,00     A*************************************************************************
```

Les lignes 1,00 à 27,00 sont des lignes de commentaires. Elles se repèrent au symbole '*' présent après la première lettre de la ligne.

La ligne 28,00 indique le nom du fichier contenant les formats des données (ici, le fichier « DBREFER »).

La ligne 29,00 indique le nom du « record format ». Il s'agit du nom donné à la structure des données (ici, 'ZBUYER').

Les lignes 33,00 à 47,00 indiquent les noms des champs, la référence à un format dans le fichier « DBREFER » (REFFLD(xxx) pour REFerence FieLD(xxx)) et l'en-tête de colonne indiquant la signification du champ (COLHDG(xxx) pour COLumn HeaDing(xxx)).

Le premier travail effectué est l'encodage dans l'atelier DB-Main de la description de ces fichiers. Celui-ci s'est fait en recopiant les listings créés à partir des fichiers physiques et du fichier « DBREFER » et contenant une description plus lisible des fichiers. Cet encodage est long et fastidieux : il faut introduire une quantité énorme de champs en spécifiant, pour chacun d'eux, leur nom, leur type, leur longueur et une description sémantique. De plus, lorsque ce

travail a été réalisé, il était impossible de recopier des attributs déjà encodé, ce qui nous obligeait à encoder absolument tous les champs. Heureusement, certaines facilités de recopiage existent maintenant et permettraient d'accélérer quelque peu le processus.

Il est intéressant de citer également le cas du fichier « PARTMS », qui contient la description des produits achetés. Mais, comme nous l'avons dit ci-dessus, les fichiers font partie de la base de données générale de l'usine. Ainsi, ce fichier ne contient pas que des données concernant les achats, mais également des données concernant l'utilisation du produit pendant la production, d'autres destinées à la comptabilité,... Il a donc fallu faire un tri parmi les champs de ce fichier, basé sur les descriptions sémantiques de ceux-ci, qui donnent une idée assez précise de l'utilisation qui en sera faite. Lorsqu'il n'est pas possible de déceler si un champ concerne l'achat du produit ou pas, nous l'avons inclu dans la base de données, de façon à être certain d'y retrouver tous ceux qui nous intéressent. Les autres ne feront l'objet d'aucune contrainte et il est donc inutile de les encoder. De plus, nous nous rendrons compte au fur et à mesure de l'avancement du processus de rétro-ingénierie que le schéma devient de plus en plus complexe et prend de plus en plus de place, et l'ajout de champs augmente d'autant la place occupée par la table.

La liste complète des fichiers encodés est reprise à l'annexe 3. Nous allons maintenant détailler l'analyse des fichiers logiques, qui nous permettra de retrouver certaines identifiants.

4.2 L'analyse des fichiers logiques

L'analyse des fichiers logiques (contenant la description des index) nous apporte peu d'information à ce stade-ci. Seuls les index uniques sont intéressants car ils permettent de déceler des identifiants. Remarquons toutefois que rien ne permet de déduire que ces identifiants sont minimaux. De plus, l'analyse des programmes permettra également de retrouver des identifiants, qui viendront peut-être contrecarrer l'information contenue dans les fichiers logiques. Les fichiers contenant des index qui ne sont pas déclarés « UNIQUE » ne sont pas intéressant, nous n'en avons donc pas tenu compte.

Cette façon de faire nous écarte quelque-peu de la méthode de la section 2.2. En effet, selon celle-ci, il aurait fallu encoder toutes les structures d'accès, identifiantes ou pas, pour les éliminer lors de la phase de simplification du schéma, car elle n'apportent rien au niveau sémantique. Cette façon de faire est intéressante lorsqu'on utilise un « parser », qui traduit directement un texte source, écrit par exemple en SQL, en structures de données et qui ne fait pas de différences entre les différentes structures. Dans notre cas, les structures sont entrées « à la main » et l'encodage, puis l'élimination, de ces structures est en fait une perte de temps inutile...

Le Tableau 2 contient donc la liste des fichiers logiques contenant des index identifiants, ainsi que les champs qui les composent et des éventuelles conditions supplémentaires.

Tableau 2 : les fichiers logiques

Fichier logique	Fichier physique	Index unique
BUYER01	BUYER	ZBAC
CALNDR01	CALNDR	LCALYY, LCALMM, LCALDD
CALNDR06	CALNDR	LBAXYY, LBAXMM, LBAXDD, LCALYY, LCALMM, LCALDD
CMPNY01	CMPNY	ZCMPNY
COUNTR01	COUNTR	ZCMPNY, ZTYPE
CURCOD01	CURCOD	ZCURCOD
DELMTH01	DELMTH	ZMTHCD, ZLNGCD
DELMTH02	DELMTH	ZLNGCD, ZMTHCD
DELTRM01	DELTRM	ZTRMCD, ZTRMLG
DELTRML1	DELTRM	ZDELCD, ZTRMLG
IPSP01L1	IPSP01	PLAB01, PCOM01, PBAS01, PSUF01, SUPL01, ADDR01, STAT01
IPSP01L2	IPSP01	SUPL01, ADDR01, STAT01, PLAB01, PCOM01, PBAS01, PSUF01
IPSP03L1	IPSP03	CMNY03, PAF203, (ACTI03 ≠ 'D')
IPSPO8L1	IPSP08	STAT08, (ACTI08 ≠ 'D')
LNGCOD01	LNGCOD	ZBCODE
PARTMSL1	PARTMS	ACMPNY, APARTL, APARTC, APART, APARTS
PAYTRM01	PAYTRM	ZPAYCD, ZPAYLG
PMCREF01	PMCREF	CRCOMP, CRCMPR
PMCREF02	PMCREF	CRCOMP, CRBAS, CRSUFX, CRLBL, CRCOM
POPDHD01	POPDHD	HDCMNY, HDPART, HDPRTS, HDPRTL, HDPRTC, HDVEND
PRTCOM01	PRTCOM	QCCMNY, QCPART, QCPRTS, QCPRTL, QCPRTC, QCVEND, QCSEQ
PRTCOM02	PRTCOM	QCCMNY, QCPART, QCPRTS, QCPRTL, QCPRTC, QCVEND, QCSEQ, (QCSTAT ≠ 'D')
PURADD01	PURADD	G4CMNY, G4VEND, G4CODE, G4OSEQ
PURITM01	PURITM	G2CMNY, G2BAC, G2ORNO, G2LINE, (G2TYPE = 'P')
PURLED01	PURLED	G5VEND, G5METH, G5LNGC
PURMST01	PURMST	G1CMNY, G1BAC, G1ORNO, (G1PURT = 'P')
PURVEN01	PURVEN	G3CMNY, G3VND#
UNITMS01	UNITMS	ZUNMS
VATCTL01	VATCTL	ZCCODE
WHOUSE01	WHOUSE	Z1CMNY, Z1WHSE

Voici, à titre d'exemple, le contenu du fichier « IPSP08L1 » :

```
85,90      *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
86,00      * Source directives for the SYNON/1 compile pre-processor
86,10      *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
86,20      * Title
86,30     /*T: IPS - Status table pour Approbation des fournisseurs
86,40      *++++++++++++++
86,50      * Pre-Compile
86,60      *++++++++++++++
86,70      * Compiler
86,80     /*Z:    CRTLF       MAINT(*IMMED) RECOVER(*AFTIPL)
86,90     /*Z:                AUT(*ALL)
87,00      *++++++++++++++
```

```
87,10      * Post-Compile
87,20      *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
87,30     A*
87,40     A                                     UNIQUE
87,50     A          R IPSP08R                  PFILE(IPSP08)
87,60     A                                     TEXT('Statuts fournisseurs')
87,70     A          K STAT08
87,80     A          O ACTI08                   COMP(EQ 'D')
```

La ligne 87,40 indique que cet index est unique. Il définit donc un identifiant pour la table IPSP08.

La ligne 87,50 indique le fichier physique correspondant (PFILE(IPSP08)) et le nom du « record format » (IPSP08R) (voir l'analyse des fichiers physiques concernant ces termes).

La ligne 87,70 indique que le champ « STAT08 » du fichier « IPSP08 » est le premier champ (et l'unique dans ce cas-ci) de l'index. Ceci vient du 'K' se trouvant devant le nom du champ ('K' symbolise le mot « Key »).

La ligne 87,80 indique qu'il faut omettre (symbolisé par la lettre 'O') les enregistrements dont la valeur du champ « ACTI08 » est égale à 'D'. Dans d'autres cas, la lettre 'O' sera remplacée par la lettre 'S' (pour le mot « Sélection ») signifiant qu'il ne faut tenir compte que des enregistrements dont le champ spécifié vérifie la condition énoncée.

On peut donc en déduire que « STAT08 » est un identifiant des enregistrements de « IPSP08 » dont la valeur de « ACTI08 » est différente de 'D'.

Tous ces identifiants sont repris dans la description des fichiers contenue dans l'annexe 3.

A ce stade-ci, nous avons la description de la structure de chacun des fichiers repris ainsi qu'un ou des identifiants d'une bonne partie de ceux-ci.

4.3 L'atelier DB-Main

Le schéma résultant de l'analyse des fichiers physiques et logiques est celui de la version « Brut » du schéma « Belgium-Malta »[5] : il contient donc la description des différents champs et les identifiants trouvés dans les fichiers logiques.

L'encodage des descriptions des fichiers dans l'atelier s'est fait « à la main ». Chaque champ de chaque table a dû être décrit. Le rôle de DB-Main est, à ce stade, assez passif. Il l'aurait moins été si la description des structures des fichiers était disponible dans le langage SQL. En effet, le « parser » SQL nous aurait permis d'éviter d'encoder toutes ces données, ce qui prend un temps non négligeable. De plus, les facilités de recopiage des champs nous aurait quelque peu facilité la tâche, notamment dans le cas où un fichier contient une longue liste d'attributs similaires.

La recherche des identifiants dans les fichiers logiques s'est également faite sans l'aide de DB-Main. Dans ce cas-ci, l'atelier ne nous aurait pas beaucoup aidé dans notre recherche, car il suffisait de déceler la présence du mot-clé « UNIQUE » dans les fichiers logiques. Toutefois,

[5] Le nom « Belgium-Malta » vient du fait que la base de données de la Belgique et de Malte sont rigoureusement les mêmes.

dans le cas où il existe un grand nombre d'index, dont très peu sont identifiants, une recherche de patterns permettrait de gagner un peu de temps.

De plus, les descriptions des fichiers physiques reçues contenaient les noms des fichiers logiques correspondants. On pourrait imaginer un scénario dans lequel ces noms n'auraient pas été disponibles. Dans ce cas, l'atelier aurait sans doute permis, via une recherche de pattern contenant le nom d'un fichier physique, de retrouver les fichiers logiques associés à celui-ci et éventuellement effectuer à ce stade la sélection de ceux qui contenaient le mot-clé « UNIQUE ».

Les identifiants sont les seules contraintes que nous pouvons retrouver au niveau des fichiers. En effet, le SGBD utilisé ne supporte pas les autres contraintes, comme les contraintes de référence, et seule l'analyse des programmes utilisant les fichiers décrits ci-dessus permettra d'inclure dans notre schéma suffisamment de contraintes pour qu'il soit significatif au niveau du but fixé pour ce mémoire. Le chapitre suivant présente l'analyse des programmes, terminant ainsi l'étape de d'extraction des structures de données, avant que les deux chapitres suivant ne s'attardent sur l'étape de conceptualisation des structures de données.

Chapitre 5 : Les schémas enrichis

Ce chapitre présente les résultats de l'analyse des programmes de maintenance des fichiers décrits dans le chapitre précédent.

Les programmes de maintenance de ces fichiers n'ont pas été trouvés sans peine. Il a fallu parcourir une liste énorme de noms de fichiers et y sélectionner ceux qui, à première vue, étaient susceptibles de contenir des instructions de vérifications de contraintes. La liste ci-dessous contient les noms des 24 progammes retenus. Il en existe certainement d'autres, contenant des contraintes supplémentaires, mais les recherches nécessaires à leur découverte auraient été longues et fastidieuses. De plus, ces programmes contiennent déjà une palette assez large de types de contraintes et le but n'est pas non plus d'être exhaustif, mais de faire apparaître des structures intéressantes d'un point de vue méthodologique.

DE140	OP104	OP140	PO130
OP130	CHKACC	DE105	PO155
OP114	PAR001	DE150	IPS005
OP112	MRP990	IC165	PO120
OP110	T31002R	PO903	PO203
OP106	TS075	IC150	IPS014

Tous ces programmes sont écrits dans le langage RPG[6]. Leur analyse a été faite à partir des listings de ceux-ci, en simulant leur exécution. Ces programmes servant à la maintenance de fichiers, ils comportent généralement un menu composé des items suivants : « create », « delete », « modify ». Des variantes sont disponibles en fonction des fichiers correspondants.

Quand un item de menu est sélectionné, il fait apparaître un écran, dans lequel l'utilisateur introduit les valeurs indispensables à la réalisation de la tâche demandée. Ces écrans sont définis dans des fichiers séparés du programme. Ceux-ci indiquent la position de chaque mot affiché à l'écran, les zones de saisie, les variables qui contiendront les valeurs introduites,... Ils ont peu d'importance pour nous, car les vérifications des contraintes se font surtout dans le programme proprement-dit, et pas au niveau des écrans.

Quand les valeurs ont été introduites à l'écran, les vérifications des contraintes commencent. Celles-ci se font sur les variables du programme, juste avant de les enregistrer dans les fichiers. Enfin, les valeurs sont sauvegardées dans le fichier maintenu par le programme.

Nous ne détaillerons pas l'analyse complète de chaque programme, car ce serait trop long et très ennuyeux. Nous nous focaliserons plutôt sur des exemples illustrant les différents types de contraintes trouvées.

Les différents types de contraintes explicitées dans ce chapitre sont les champs obligatoires, les contraintes sur le domaine, les identifiants, les contraintes de référence, les différentes décompositions d'un enregistrement et enfin les autres contraintes, qui regroupent celles qui ne sont pas exprimables directement au niveau du schéma. De cette façon, nous passons en revue les contraintes habituellement rencontrées dans les bases de données relationnelles.

[6] Les éléments de ce langage présents dans les programmes analysés se trouvent dans l'annexe 2.

Au niveau de l'atelier, les contraintes ont été introduites manuellement. Toutefois, lorsque toutes les contraintes ont été retrouvées, nous avons recherché les patterns spécifiques au chaque type exposé afin d'évaluer la faisabilité d'une recherche semi-automatique par l'atelier en utilisant la fonctionnalité de recherche de patterns, décrite à la section 2.4.3.

Nous avons vu que les patterns sont définis via deux fichiers : le premier contient une librairie de patterns de base et le second contient le pattern spécifique. Voici le contenu de la librairie utilisée dans la suite[7] :

```
chiffre ::= range(0-9);
indic ::= chiffre chiffre;
sep_rpg ::= /g"[ ]*";
sep_ligne ::= /g"[/n/t/r ]*";
num_ligne ::= /g"[0-9,]*";
mot_rpg ::= /g"[a-zA-Z0-9_#*@'-][a-zA-Z0-9_#*@,'-]*";
```

Le pattern « indic » contient la description d'un indicateur, « sep_rpg » décrit un séparateur du langage rpg (un nombre quelconque, éventuellement nul, de caractères blancs), « sep_ligne » un séprateur de lignes, « num_ligne » un numéro de ligne et « mot_rpg » les mots utilisés par le langage.

5.1 Les champs obligatoires

Nous avions supposé à la section 3.3 lors de la présentation du SGBD utilisé que tous les champs étaient obligatoires. Lors de l'analyse des fichiers physiques et logiques, rien ne permettait de décider si un champ était facultatif ou pas. Mais, dans les programmes, on trouve des parties de code qui peuvent être considérées comme la vérification du caractère obligatoire d'un champ. Celles-ci sont en fait assez simples à repérer. Elles consistent à vérifier que le champ ne possède pas la valeur '*BLANKS'. Cette valeur est une valeur conventionnelle représentant l'absence de valeur en RPG. On peut donc en déduire que les champs faisant l'objet d'une telle vérification doivent être considéré comme obligatoire et les autres pas.

Mais examinons de plus près un exemple de vérification de ce type de contraintes. Ainsi, par exemple, dans le programme « DE150 », maintenant le fichier « BUYER » des acheteurs, on trouve la portion de code suivante :

```
815,00     C*    ERR IF BUYER ANALYST DESCRIPTION IS EQUAL TO BLANKS
816,00     C*
817,00     C           @DESC     IFEQ *BLANKS
818,00     C                     ADD  1         X
819,00     C                     MOVE 'IC_1204' EMSG,X
820,00     C                     MOVEA'1'       *IN,33
821,00     C                     ENDIF
```

Si la variable *@DESC* contient la valeur '*BLANKS', on ajoute le message 'IC_1204' au tableau « EMSG » des messages d'erreurs et on met l'indicateur 33 à 1.

A un autre endroit du programme, la variable *@DESC* est sauvegardée dans le champ *ZDESCR* du fichier « BUYER », dont le « record format » est « ZBUYER » :

```
637,00     C                     MOVE @BAC      ZBAC
638,00     C                     MOVE @DESC     ZDESCR
639,00     C                     MOVE @ORTY     ZORTY
```

[7] La description du langage de définition des patterns se trouve à l'annexe 1.

```
640,00     C                   MOVE @APPRO     ZAPPRO
640,013108 C                   MOVE @FLG1      WFLAG1
641,00     C*
642,00     C*   ----------------------------------------
643,00     C*    RECORD ADDITION
644,00     C*   ----------------------------------------
645,00     C           @OPT    IFEQ 1
646,00     C                   WRITEZBUYER
```

On peut donc en conclure que le champ *ZDESCR* du fichier « BUYER » est obligatoire.

Les champs obligatoires sont assez nombreux dans les fichiers analysés. Ceux qui ne le sont pas seront caractérisés dans l'atelier DB-MAIN par le sigle [0-1] se trouvant à côté de leur nom.

Les essais de recherche semi-automatique, grâce aux facilités de « pattern matching » de l'atelier, ont été mis en oeuvre de la façon suivante. Un pattern a été défini pour rechercher les lignes de programme testant l'égalité d'une variable avec la contante '*BLANKS'. Ce pattern est le suivant :

```
var ::= mot_rpg;
oblig ::= "C" sep_rpg @var sep_rpg "IFEQ" sep_rpg "*BLANKS";
```

Le pattern « oblig » définit les lignes de programmes qui nous intéressent. Le pattern « var » contient le nom de la variable en question. Ce pattern a été testé sur les programmes analysés « à la main ». Chaque champ obligatoire d'un fichier a été retrouvé grâce à cette recherche ciblée. Toutefois, il faut ajouter deux autres patterns : celui qui retrouve le champ dans lequel la variable est sauvegardée et celui qui retrouve le fichier auquel appartient le champ.

Pour retrouver le champ dans lequel est recopiée la valeur de la variable, nous avons utilisé le pattern suivant :

```
champ ::= mot_rpg;
moveop ::= {"MOVE"|"MOVEL"};
move ::= moveop sep_rpg @var sep_rpg @champ;
add ::= "Z-ADD" sep_rpg @var sep_rpg @champ;
```

Les patterns « move » et « add » définissent les lignes qui nous intéressent. Le premier est utilisé pour des champs alphanumériques et le second pour les champs numériques. Si on ne connaît pas le type à l'avance, il suffit de faire la recherche avec les deux patterns. Le pattern « var », qui est le même que celui de la recherche précédente, contient évidemment le nom de la variable à laquelle nous nous intéressons. Le pattern « moveop » contient les différentes possibilités d'opérateurs de recopiage de la valeur d'une variable de type alphanumérique vers une autre. La pattern « champ » contient le nom du champ dans lequel est recopiée la variable. Toutefois, ce n'est pas suffisant, car il se peut que la variable soit recopiée plusieurs fois avant que sa valeur ne soit sauvegardée dans un fichier. Il faudra alors appliquer la recherche de ce pattern plusieurs fois de suite, en plaçant dans le pattern « var » la valeur du pattern « champ » trouvée à l'étape précédente. On peut ainsi construire un graphe de noms de variables liées entre-elles par l'opérateur définit ci-dessus. Il nous reste enfin à savoir dans quel fichier la valeur se retrouve finalement. Pour cela, le pattern suivant nous sera d'un grand secours :

```
format ::= mot_rpg;
write ::= "WRITE" sep_rpg @format;
```

Le pattern « write » recherche les endroits du programme où des enregistrements sont ajoutés à un fichier. Etant donné que chaque programme traite tous les cas possibles (ajout, effacement

ou modification d'un enregistrement d'un fichier), il n'est pas nécessaire de faire la recherche des endroits où le même enregistrement pourrait être simplement modifié. De plus, les contraintes doivent nécessairement être vérifiée lors d'un ajout, et en nous cantonnant à ce cas, nous ne retrouvons pas moins de contraintes qu'en les traitant tous.

Pour se faire une idée précise de la façon dont l'atelier recherche des patterns, nous allons reprendre le programme DE150, qui maintient le fichier « BUYER » des acheteurs.

La première recherche nous indique quelles sont les variables testées. Après avoir sélectionné l'item de menu correspondant à la recherche de pattern dans un code source, nous indiquons quel pattern nous voulons rechercher. Dans ce cas, il s'agit du pattern « oblig ». La Figure 21 nous montre la sélection du pattern. On remarque la partie variable « var » de ce dernier.

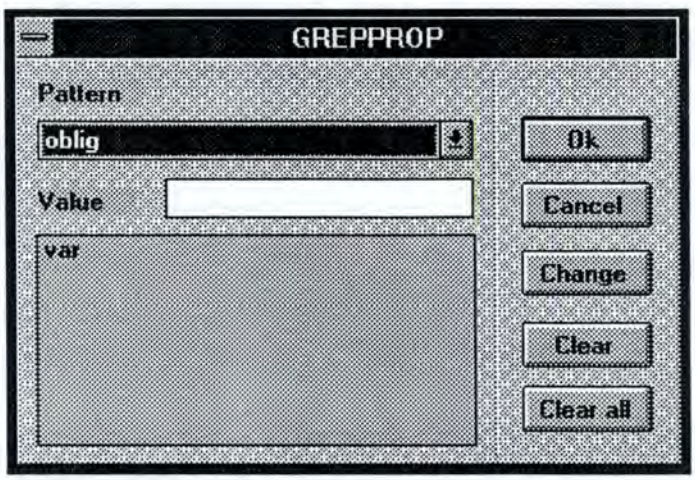


Figure 21 - Sélection du pattern "oblig"

Après avoir cliqué sur « OK », l'atelier s'arrête sur la ligne suivante :

```
694,00     C*   ERR IF BUYER ANALYST CODE NOT ENTERED
695,00     C*
696,00     C           @BAC      IFEQ *BLANKS
697,00     C                     ADD  1          X
698,00     C                     MOVE 'DE_0153' EMSG,X
699,00     C                     MOVEA'1'        *IN,32
700,00     C                     END
```

La variable *@BAC* ne peut donc prendre la valeur nulle. Après avoir relancé la recherche plusieurs fois, l'atelier nous propose la variable *@DESC*, que nous avions également trouvée :

```
815,00     C*    ERR IF BUYER ANALYST DESCRIPTION IS EQUAL TO BLANKS
816,00     C*
817,00     C           @DESC     IFEQ *BLANKS
818,00     C                     ADD  1          X
819,00     C                     MOVE 'IC_1204' EMSG,X
820,00     C                     MOVEA'1'        *IN,33
821,00     C                     ENDIF
```

Le pattern « var » contient la valeur « @DESC », indiquant le nom de la variable en cours de traitement. Nous allons maintenant rechercher le champ auquel cette variable est liée. Pour cela, nous choisissons le pattern « move » ou « add » grâce à la boîte de dialogue de la Figure 22).

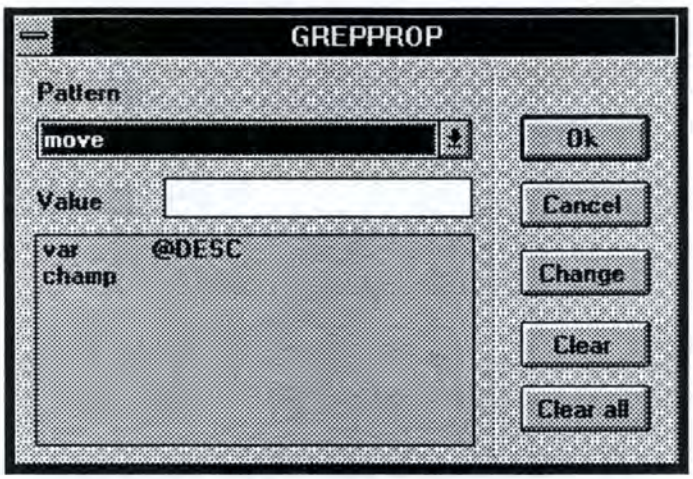


Figure 22 - Sélection du pattern "move"

On remarque sur cette figure que le pattern « var » contient toujours la même valeur, et l'atelier va donc rechercher les variables dans lesquelles le programme recopie la valeur de *@DESC* :

```
637,00     C                    MOVE @BAC       ZBAC
638,00     C                    MOVE @DESC      ZDESCR
639,00     C                    MOVE @ORTY      ZORTY
640,00     C                    MOVE @APPRO     ZAPPRO
```

Il s'agit de la variable *ZDESCR*. Nous relançons la recherche pour trouver d'éventuelles d'autres variables dans même cas et l'atelier nous donne le message de la Figure 23.

Figure 23 - Message de fin de recherche

Nous sommes à présent certains que la valeur de la variable *@DESC* ne sera recopiée que dans la variable *ZDESCR*. Il reste maintenant à retrouver les fichiers dans lesquels le programme ajoute des enregistrements. Ceci se fait grâce au pattern « write » :

```
355,00     C                    MOVEA'0'        *IN,26
356,00     C*
357,00     C*  WRITE ANY ERRORS/MESSAGES
358,00     C*
359,00     C           EMSG,1   IFNE *BLANKS
```

Cette ligne est une ligne de commentaire (reconnaissable à l'astérisque après le caractère 'C') et ne doit donc pas être prise en compte. Après avoir relancé la recherche quelques fois, l'atelier nous propose la ligne suivante :

```
645,00     C           @OPT     IFEQ 1
646,00     C                    WRITEZBUYER
```

```
647,00     C                    ADD  1         X
648,00     C                    MOVE 'DE_0150' EMSG,X
649,00     C                    ELSE
```

Or, le record format « ZBUYER » contient un champ de nom *ZDESCR* qui a reçu la valeur de la variable *@DESC*. On peut dès lors en déduire que le champ *ZDESCR* du fichier « BUYER » (qui est le fichier physique dont le nom du record format est « ZBUYER ») est un champ obligatoire.

Cette procédure a l'air plus fastidieuse que la recherche manuelle, mais il n'en est rien. Il suffit, dans le cas des champs obligatoires, de suivre la valeur des variables trouvées lors de la première recherche pour retrouver les contraintes. Dans la recherche manuelle, tout le programme doit être passé au crible. On y gagne donc au niveau du temps, sans perdre d'information, puisque toutes les variables trouvées manuellement ont également été retrouvées par l'atelier. Il faut toutefois faire remarquer que cette recherche automatique doit s'accompagner d'une vérification minutieuse. La ligne du programme contenant le test sur la variable n'est peut-être jamais exécutée lors de l'ajout d'un enregistrement. Dans ce cas, il faut retrouver le contexte dans lequel se trouve la ligne et le temps passé risque d'être aussi important que lors d'une recherche manuelle.

En résumé, on peut dire que l'avantage se marque fortement lorsque le programme à analyser comporte peu de contraintes. De plus, la recherche par l'atelier des variables testées permet de n'en oublier aucune. En outre, il est plus facile de traiter de cette façon une variable à la fois, alors que manuelllement, les manipulations de listings deviendraient vite exaspérantes... On peut également imaginer une utilisation conjointe des deux méthodes : la recherche des patterns indique de façon précise quels sont les variables intéressantes et permet d'accélérer le « décorticage » du programme en ne s'intéressant qu'aux choses qui nous intéressent directement. En recherche de pattern pure, on n'est pas sûr de ce qu'on fait, et en mode manuel pur, on fait beaucoup trop, étant donné qu'il faut relever les contraintes de toutes les variables, car on ne sait pas à l'avance si leur valeur se retrouvera dans un fichier ou pas... En prenant une combinaison des deux, on peut cibler les recherches et gagner ainsi du temps.

Nous allons maintenant passer à un type de contraintes qui ressemble à celui que nous venons d'étudier : les contraintes sur le domaine.

5.2 Les contraintes sur le domaine

Les contraintes sur le domaine d'une variable sont similaires à celles sur les champs obligatoires. Au lieu de tester si la variable possède la valeur '*BLANKS', on teste si sa valeur possède bien une des valeurs permises. Dans ce cas, le champ sera automatiquement obligatoire, à moins qu'il ne soit spécifiquement prévu que le champ puisse prendre la valeur nulle.

Par exemple, dans le programe « DE105 », maintenant le fichier « WHOUSE » des lieux de livraison, on trouve la portion de code suivante :

```
949,00     C          @CSDE     IFNE 'Y'
950,00     C          @CSDE     IFNE 'N'
951,00     C                    ADD  1         X
952,00     C                    MOVE 'IC_4702' EMSG,X
953,00     C                    MOVEA'1'       *IN,33
954,00     C                    END
955,00     C                    END
```

Si la valeur de *@CSDE* est différente de 'Y' et de 'N', on ajoute le message 'IC_4702' au tableau « EMSG » des messages d'erreur. Dans une autre partie du programme, la variable *@CSDE* est sauvegardée dans le champ *Z1CSDE* du fichier « WHOUSE », dont le « record format » est « Z1WHOUSE » :

```
  792,00     C                    MOVE @WHSE     Z1WHSE
  793,00     C                    MOVE @DESC     Z1DESC
  794,00     C                    MOVE @CSDE     Z1CSDE
  795,00     C                    MOVE @ADDR1    Z1ADD1
  796,00     C                    MOVE @ADDR2    Z1ADD2
  797,00     C                    MOVE @ADDR3    Z1ADD3
...

  807,00     C*   ---------------------------------------
  808,00     C*    RECORD ADDITION
  809,00     C*   ---------------------------------------
  810,00     C           @OPT     IFEQ 1
  811,00     C                    WRITEZ1WHOUSE
```

On peut donc en conclure que le champ *Z1CSDE* du fichier « WHOUSE » doit posséder une valeur égale à 'Y' ou à 'N'. C'est en fait une façon de représenter une valeur booléenne.

Dans l'atelier DB-MAIN, les contraintes sur les domaines des champs ne sont pas prises directement en compte. Il faudra donc les introduire « à la main » dans la partie « sémantique » de la description du champ dont il est question.

Il y a également moyen de rechercher automatiquement les champs faisant l'objet de telles contraintes. Pour cela, nous avons employé la stratégie suivante : rechercher les variables qui interviennent dans un test et suivre le trajet de leur valeur jusqu'à l'écriture dans un fichier. La deuxième partie est identique à celle de la technique des champs obligatoires.

Les contraintes sur le domaine englobent les contraintes de champ obligatoire. Toutefois, ces dernières sont retrouvées plus rapidement lorsqu'elles sont traitées séparément, ce qui nous à poussé à écrire des patterns différents dans les deux cas.

La recherche des variables faisant l'objet d'un test a été réalisée en utilisant le pattern suivant :

```
var ::= mot_rpg;
valeur ::= mot_rpg;
relop ::= {"IFEQ"|"IFNE"|"IFLT"|"IFLE"|"IFGT"|"IFGE"};
domaine ::= @var sep_rpg relop sep_rpg @valeur;
```

Le pattern « domaine » recherche les lignes de programmes qui testent si la valeur d'une variable est comparée à une autre valeur (d'une variable ou d'une constante). La pattern « var » contient le nom de la variable et le pattern « valeur » contient la valeur comparée. Le pattern « relop » contient les différents opérateurs de comparaison possibles.

La recherche automatique s'arrête évidemment sur chaque ligne comprenant une comparaison. Il faut faire le tri et ne retenir que les choses qui nous intéressent. Après avoir relancé plusieurs fois la recherche, l'atelier nous propose la ligne suivante :

```
  947,00     C*
  948,00     C*
  949,00     C           @CSDE    IFNE 'Y'
  950,00     C           @CSDE    IFNE 'N'
  951,00     C                    ADD  1         X
  952,00     C                    MOVE 'IC_4702' EMSG,X
  953,00     C                    MOVEA'1'       *IN,33
  954,00     C                    END
```

On reconnaîtra l'exemple pris lors de l'exposé des contraintes sur le domaine. A partir de ce moment, il suffit de rechercher le champ dans lequel la valeur de la variable est recopiée et le fichier auquel ce champ appartient, de la même façon que pour les champs obligatoires.

Les mêmes remarques restent valables dans ce cas-ci : la recherche purement automatique produit un résultat insatisfaisant et ce n'est que combinée à une recherche dans les listings qu'elle est utile. De plus, contrairement au cas précédent où le nombre de lignes proposées par l'atelier était restreint, la recherche produit un grand nombre de lignes de test. Pour un grand nombre d'entre-elles, la recherche d'un champ et d'un fichier se solde par un échec qu'il n'est pas aisé de prévoir. Toutefois, un certain nombre de variables peuvent être éliminées directement : les indicateurs, par exemple, qui servent grosso modo de variables booléennes indiquant la réussite ou l'échec d'une opération, ne sont jamais sauvés dans un fichier. Mais il faut le savoir à l'avance... Il ne faut retenir également que les lignes qui produisent un message d'erreur après avoir fait le test. Celles-ci ne sont heureusement pas légion et permettent de faire un premier tri intéressant. En ne retenant que ce type de lignes, nous avons relevé 6 variables, ce qui n'est pas énorme. Sur les 6, une seul n'est recopiée dans aucune autre variable. Les 5 autres génèrent un groupe de 8 variables. La recherche du fichier fournit le record format « Z1WHOUSE » qui contient 5 des 8 variables en question et il en reste donc 3, qui ne sont jamais recopiées ou qui le sont dans une variable déjà prise en compte.

Un fois de plus, la recherche de patterns doit être utilisée comme un support à la recherche de contraintes et pas comme une solution finale. Il reste nécessaire de comprendre la façon dont un programme est exécuté pour y retrouver des contraintes.

5.3 Les identifiants

Les identifiants peuvent se trouver en analysant les fichiers logiques (voir section 4.2), mais proviennent également de l'analyse des programmes de maintenance. Dans ce cas, ils font l'objet d'une combinaison de vérifications : le caractère obligatoire et, dans le cas de l'ajout d'un enregistrement, l'absence d'un enregistrement possédant déjà la valeur du champ concerné. Dans le cas d'une modification ou d'un effacement, le programme teste s'il existe un enregistrement possédant la valeur du champ concerné (dans le cas où la valeur a été modifiée pendant l'exécution du programme). Ainsi, le programme « DE140 » nous renseigne sur l'identifiant du fichier « UNITMS » :

```
683,00     C            @UNMS     IFEQ *BLANKS
684,00     C                      ADD  1         X
685,00     C                      MOVE 'DE_0163' EMSG,X
686,00     C                      MOVEA'1'       *IN,32
687,00     C                      END
688,00     C*
689,00     C            EMSG,1    CABNE*BLANKS   $PRMPT
690,00     C            *IN33     CABEQ'1'       $PRMPT
691,00     C*
692,00     C            @UNMS     CHAINUNITMS01              8082
693,00     C*
694,00     C*  ISSUE MSG RECORD IN USE ALREADY
695,00     C*
696,00     C            *IN,82    IFEQ '1'
697,00     C                      ADD  1         X
698,00     C                      MOVE 'DE_0240' EMSG,X
699,00     C                      END
700,00     C*
701,00     C*   ADD MODE     (ERR IF RECORD ALREADY ON FILE)
702,00     C*
703,00     C            @OPT      IFEQ 1
704,00     C            *IN80     IFNE '1'
```

```
  705,00     C                   ADD  1         X
  706,00     C                   MOVE 'DE_0164' EMSG,X
  707,00     C                   MOVEA'1'       *IN,32
  708,00     C                   END
  709,00     C*
  710,00     C*   CHG/DELETE MODES  (ERR IF RECORD NOT ALREADY ON FILE)
  711,00     C*
  712,00     C                   ELSE
  713,00     C           *IN80   IFEQ '1'
  714,00     C                   ADD  1         X
  715,00     C                   MOVE 'DE_0165' EMSG,X
  716,00     C                   MOVEA'1'       *IN,32
  717,00     C                   END
  718,00     C                   END
...

  625,00     C                   MOVE @UNMS     ZUNMS
  626,00     C                   MOVE @DESC     ZDESC
  627,00     C*
  628,00     C*   ---------------------------------------
  629,00     C*    RECORD ADDITION
  630,00     C*   ---------------------------------------
  631,00     C           @OPT    IFEQ 1
  632,00     C                   WRITEZUNITMS
...

  637,00     C*   ---------------------------------------
  638,00     C*    RECORD UPDATE
  639,00     C*   ---------------------------------------
  640,00     C*
  641,00     C                   UPDATZUNITMS
...
```

Les lignes 683,00 à 687,00 assurent que la variable *@UNMS* ne contient pas la valeur nulle, ce qui indique que le champ correspondant à cette variable est obligatoire.

La ligne 692,00 recherche dans le fichier « UNITMS » (le fichier physique correspondant au fichier logique « UNITMS01 ») les enregistrements dont le champ *ZUNMS* (cfr description du fichier « UNITMS01 ») possède la même valeur que celle de la variable *@UNMS*. Si de tels enregistrements existent, l'indicateur 80 sera mis à 0 et si tel n'est pas le cas, l'indicateur 80 sera mis à 1.

Dans le cas où on ajoute un enregistrement au fichier (*@OPT* = 1), les lignes 703,00 à 708,00 vérifient qu'aucun enregistrement n'a été trouvé dans la recherche de la ligne 692,00, ce qui revient à tester si l'indicateur 80 est mis à 1. Si tel n'est pas le cas, on ajoute le message 'DE_0164' au tableau « EMSG » des messages d'erreur. Ensuite, l'enregistrement est ajouté au fichier lors de l'exécution des lignes 625,00 et 632,00.

Dans le cas où on modifie ou efface un enregistrement du fichier (*@OPT* ≠ 1), les lignes 713,00 à 717,00 vérifient qu'un enregistrement a été trouvé lors de la recherche de la ligne 692,00, ce qui revient à tester si l'indicateur 80 est mis à 0. Si tel n'est pas le cas, on ajoute le message 'DE_0165' au tableau « EMSG » des messages d'erreur. Ensuite, l'enregistrement est mis à jour lors de l'exécution des lignes 625,00 et 641,00.

On peut donc dire que le champ *ZUNMS* est identifiant du fichier « UNITMS ». En effet, il est obligatoire, dans le cas d'un ajout, il ne peut exister d'enregistrement dont le champ *ZUNMS* possède la même valeur que le champ *ZUNMS* de l'enregistrement que l'on veut ajouter, et dans le cas d'une modification, on ne peut modifier que des enregistrements dont la valeur du champ *ZUNMS* est présente dans le champ d'un enregistrement du fichier.

Les identifiants sont directement pris en charge par l'atelier. Il suffit de définir un groupe reprenant les champs composant l'identifiant et de sélectionner le bouton-radio correspondant à l'identifiant primaire ou secondaire selon le cas. Il faut être vigilant, car des contraintes supplémentaires peuvent apparaître quand le fichier logique utilisé pour accéder au fichier en contient. On est alors dans un cas semblable à ceux rencontrés lors de l'analyse des fichiers logiques.

Pour retrouver un identifiant grâce à une recherche de pattern, il faut déjà avoir une bonne idée de la structure du programme. Dans ce cas-ci, il faut savoir, par exemple, que la condition « *@OPT* = 1 » équivaut à l'ajout d'un enregistrement, il faut connaître la signification de l'indicateur suivant une instruction « CHAIN »,... Nous avons utilisé les patterns définis comme suit :

```
var ::= mot_rpg;
fichier ::= mot_rpg;
indic1 ::= indic;
indic2 ::= indic;
ident1  ::= @var sep_rpg "CHAIN" sep_rpg @fichier sep_rpg @indic1 [sep_rpg
@indic2];
ligne1 ::= "@OPT" sep_rpg "IFEQ" sep_rpg "1" sep_ligne;
ligne2  ::= num_ligne sep_rpg "C" sep_rpg "*IN" @indic1 sep_rpg "IFNE" sep_rpg
"'1'";
ident2 ::= ligne1 ligne2;
```

1. Recherche d'un ligne contenant une instruction « CHAIN » (pattern « ident1 »)
2. Recherche d'un groupe de ligne composé de (pattern « ident2 ») :
   - Un test pour vois si @OPT = 1 (pattern « ligne1 »)
   - Un test sur l'indicateur à la fin de la première ligne trouvée (pattern « ligne2 »)

La recherche du pattern « ident1 » s'arrête sur la ligne suivante en donnant le résultat de la Figure 24, qui est le résultat espéré.

```
875,00     C                    MOVE LCOMP     KCOMP
  876,00     C                      MOVE @WHSE     KWHSE
  877,00     C           WHKEY      CHAINWHOUSE01             8082
  878,00     C*
  879,00     C*  ISSUE MSG RECORD IN USE ALREADY
  880,00     C*
```

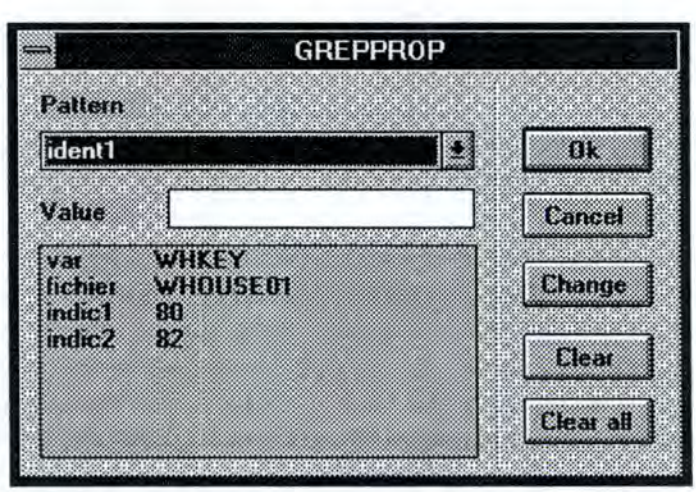


Figure 24 - résultat de la recherche du pattern "ident1"

Il nous reste à lancer la recherche du pattern « ident2 » à partir de l'endroit où se trouve la ligne trouvée, de façon à voir si cette instruction « CHAIN » sert à valider une contrainte de type identifiant. L'atelier nous propose alors la ligne suivante :

```
886,00     C*   ADD MODE     (ERR IF RECORD ALREADY ON FILE)
887,00     C*
888,00     C           @OPT       IFEQ 1
889,00     C           *IN80      IFNE '1'
890,00     C                      ADD  1          X
891,00     C                      MOVE 'IC_1210' EMSG,X
892,00     C                      MOVEA'1'       *IN,32
893,00     C                      END
894,00     C*
```

On sait maintenant que *WHKEY* est identifiant du fichier physique corrspondant au fichier logique « WHOUSE01 ». Pour savoir de quoi est composé *WHKEY*, il suffit d'en recherche toutes les occurrences, grâce au pattern suivant :

```
occur ::= @var;
```

Il suffit de placer 'WHKEY' dans le pattern « var » et de lancer la recherche. Celle-ci s'arrête à trois endroits : sur une ligne de commentaire, sur la décomposition de *WHKEY*, représentée ci-dessous, et sur la ligne contenant l'instruction « CHAIN » l'utilisant.

```
321,00     C*   WAREHOUSE KLIST
322,00     C*   ---------------
323,00     C           WHKEY      KLIST
324,00     C                      KFLD           KCOMP
325,00     C                      KFLD           KWHSE
326,00     C*    WORKCENTER KLIST
327,00     C*    --------------------
```

On trouve alors que *WHKEY* est composé de *KCOMP* et de *KWHSE*. Pour connaître les variables correspondantes, on utilise le pattern « move » en plaçant l'un des deux noms dans le pattern « champ » (et pas dans le pattern « var » comme précédemment, vu qu'on vérifie ici qu'un groupe de champ est identifiant au lieu de chercher dans quel champ une variable est sauvegardée). La recherche fournit dans les deux cas la ligne précédent l'instruction « CHAIN » :

```
874,00     C*
875,00     C                      MOVE LCOMP     KCOMP
876,00     C                      MOVE @WHSE     KWHSE
877,00     C           WHKEY      CHAINWHOUSE01             8082
878,00     C*
```

On sait donc maintenant que le couple (*LCOMP*, *@WHSE*) est identifiant de « WHOUSE ». Il suffit d'employer la méthode habituelle pour retrouver les champs correspondants.

Ces contraintes sont plus complexes que les précédentes. De plus, il faut déjà connaître la structure du programme pour définir le pattern. Si tous les programmes sont écrits de manière similaire, on peut, après avoir analysé un ou deux programmes, définir le pattern et faire la recherche automatique pour les autres. Par contre, si chaque programme est différent des autres, cette méthode ne sera sûrement pas d'un grand secours. La seule chose qui peut être utilisée est la recherche des lignes contenant l'instruction « CHAIN », qui est indispensable si on veut tester le caractère identifiant : elle permettra de se focaliser sur les variables utilisées dans cette instruction.

5.4 Les contraintes de référence

Les contraintes de référence sont les plus importantes dans un schéma relationnel. Comme ces contraintes ne sont pas supportées par le SGBD utilisé, elles sont présentes dans les programmes. Elles consistent à vérifier qu'un champ est présent dans un fichier autre que celui maintenu et que ce champ est identifiant dans ce dernier fichier.

Ainsi, dans le programe « PO155 », maintenant le fichier des fournisseurs « PURVEN », on trouve la portion de code suivante :

```
621,00     C           WHKEY     KLIST
622,00     C                     KFLD           KCOMP
623,00     C                     KFLD           @3WHSE
624,00     À*
...

3024,00    À*   VALIDATE PRIMARY WAREHOUSE
3025,00    À*
3026,00    C          @3WHSE     IFNE *BLANKS                      ÀIF 003
3027,00    C          WHKEY      CHAINWHOUSE01             80      À 003
3028,00    C          *IN80      IFEQ '1'                          ÀIF 004
3029,00    C                     ADD  1         X                  À 004
3030,00    C                     MOVE 'DE_0216' EMSG,X             À 004
3031,00    C                     MOVE '1'       *IN53              À 004
3032,00    C                     ENDIF                             ÀENDIF 004
3033,00    C                     ENDIF                             ÀENDIF 003
```

Après avoir testé si la variable *@3WHSE* n'a pas une valeur nulle (le champ correspondant sera donc a priori facultatif), on sélectionne les enregistrements du fichier « WHOUSE » (le fichier physique correspondant au fichier logique « WHOUSE01 ») sur base de l'index unique (et donc identifiant) décrit dans « WHOUSE01 » (instruction « CHAIN »). Dans ce cas-ci, il s'agit des enregistrements dont la valeur de *Z1CMNY* est égale à la valeur de la variable *KCOMP* et dont la valeur de *Z1WHSE* est égale à la valeur de la variable *@3WHSE* (cfr description du fichier « WHOUSE01 » et partie supérieure du programme). Si aucun enregistrement ne satisfait à ces contraintes, l'indicateur 80 est mis à 1. Sinon, il est mis à 0.

Si l'indicateur 80 est à 1, on ajoute le message 'DE_0216' au tableau « EMSG » des messages d'erreur car dans ce cas, il n'existe aucun enregistrement de « WHOUSE » dont le couple de champs (*Z1CMNY*, *Z1WHSE*) posséde la valeur de (*KCOMP*, *@3WHOUSE*) et la contrainte de référence n'est pas vérifiée.

Cette portion de code teste donc l'existence, dans le fichier « WHOUSE », d'un enregistrement dont les valeurs des champs *Z1CMNY* et *Z1WHSE* valent celles des variables *KCOMP* et *@3WHSE*. A un autre endroit du programme, les variables *KCOMP* et *@3WHSE* sont sauvegardées dans les champs *G3CMNY* et *G3WHSE* du fichier « PURVEN »[8], dont le « record format » est « G3PURVEN » :

```
2123,00    C                     MOVE @3CONT    G3CONT
2124,00    C                     MOVE @DISC     G3DISC
2125,00    C                     MOVE @3WHSE    G3WHSE
2126,00    C                     MOVE @3MPOC    G3MPOC
2127,00    C                     MOVE @3DELT    G3DELT
2128,00    C                     MOVE @3PAYT    G3PAYT
...
```

[8] Remarquons que c'est la variable *LCOMP* et non pas *KCOMP* qui est sauvegardée. Toutefois, avant le test sur la variable *KCOMP*, cette dernière avait été initialisée avec la valeur de *LCOMP*, ce qui revient au même que de faire le test avec *LCOMP* au lieu de *KCOMP*.

```
2148,00     C                   Z-ADDSYSDAT    G3LADT
2149,00     C                   TIME           G3LATM
2150,00     C                   MOVE Y6USER    G3USER
2151,00     C                   MOVE LCOMP     G3CMNY
2152,00     À*
2153,00     À*   ----------------------------------
2154,00     À*    RECORD ADDITION
2155,00     À*   ----------------------------------
2156,00     C           @OPT    IFEQ 1                          ÀIF 001
2157,00     C                   MOVE 'A'       G3STAT           À 001
2158,00     C                   WRITEG3PURVEN              83   À 001
2159,00     À*
```

On peut donc en déduire une contrainte de référence des champs (*G3CMNY*, *G3WHSE*) du fichier « PURVEN » vers les champs identifiants (*Z1CMNY*, *Z1WHSE*) du fichier « WHOUSE ».

L'atelier DB-MAIN prend directement en charge ces contraintes. Pour cela, il faut définir un groupe comprenant les champs *G3CMNY* et *G3WHSE* et définir une contrainte de référence vers les champs identifiants (*Z1CMNY*, *Z1WHSE*) du fichier « WHOUSE ». Il faut remarquer toutefois qu'il est obligatoire que les champs référencés soient déclarés comme identifiants dans l'atelier.

Des contraintes de référence existent également vers des champs qui ne sont pas des identifiants minimaux. En fait, ces contraintes combinent des attributs de référence avec des attributs provenant de la dénormalisation. Ainsi, dans le programme « PO130 » maintenant le fichier « PURADD », on trouve ceci :

```
 351,00     C           VENKEY  KLIST
 352,00     C                   KFLD           LCOMP
 353,00     C                   KFLD           @VEND
...

1082,00     C           VENKEY  CHAINPURVEN01              80
1083,00     C           *IN80   IFEQ '1'
1084,00     C                   ADD  1         X
1085,00     C                   MOVE 'PO_1019' EMSG,X
1086,00     C                   MOVEA'1'       *IN,31
1087,00     C                   ELSE
1088,00     C           G3STAT  IFEQ 'D'
1089,00     C                   ADD  1         X
1090,00     C                   MOVE 'PO_0052' EMSG,X
1091,00     C                   MOVEA'1'       *IN,31
1092,00     C                   END
1093,00     C                   END
...

1385,00     C           @4VOLT  IFNE G3SPLT
1386,00     C           XFRST1  ANDEQ'Y'
1387,00     C                   ADD  1         X
1388,00     C                   MOVE 'PO_2068' EMSG,X
...

 891,00     C                   MOVE LCOMP     G4CMNY
 892,00     C                   MOVE @VEND     G4VEND
 893,00     C                   MOVE @CODE     G4CODE
 894,00     C                   MOVE @4CONT    G4CONT
 895,00     C                   MOVE @4VNDN    G4VNDN
 896,00     C                   Z-ADD@4VOLT    G4VOLT
 897,00     C                   MOVE @4ADD1    G4ADD1
```

Le second groupe de lignes (1082,00 à 1093,00) nous renseigne sur la présence d'une contrainte de référence vers le fichier « PURVEN » duquel on retire les enregistrements tels que *G3STAT* = 'D' (le préfixe 'G3' est celui du fichier « PURVEN » et donc les noms commençant par 'G3' représentent des champs de ce fichier). Le troisième groupe de lignes

(1385,00 à 1388,00) impose que la valeur de la variable *@4VOLT* (et donc du champ *G4VOLT* comme on le voit à la ligne 896,00) soit identique à celle du champ *G3SPLT* du fichier « PURVEN ». Donc, la contrainte ne porte pas uniquement sur les champs *G4COMP* et *G4VEND*, mais également sur le champ *G4VOLT*, qui ne fait pas partie de l'identifiant. C'est un cas typique de dénormalisation.

Dans ce cas-ci, il faut à nouveau faire un détour en définissant un groupe reprenant les champs du fichier « PURADD » et écrire la contrainte dans la partie « sémantique » de la description de ce groupe. De plus, le fait que la contrainte de référence n'utilise que les enregistrements de « PURVEN » dont le champ *G3STAT* est différent de 'D' doit également être décrit à cet endroit, même s'il s'agit d'une contrainte de référence normale.

On peut rechercher également ce type de contraintes grâce aux patterns. Pour cela, on utilise des patterns semblables à ceux utilisés dans le cas des identifiants. Ceux que nous avons utilisés sont les suivants :

```
var ::= mot_rpg;
fichier ::= mot_rpg;
indic1 ::= indic;
indic2 ::= indic;
ref1 ::= @var sep_rpg "CHAIN" sep_rpg @fichier sep_rpg @indic1 [sep_rpg @indic2];
ref2 ::= "*IN" @indic1 sep_rpg "IFEQ" sep_rpg "'1'";
```

Le pattern « ref1 » recherche la ligne contenant l'instruction « CHAIN » recherchant les enregistrements d'un fichier sur base d'un index composé des champs référençant le fichier. Le pattern « var » contient l'index, le pattern « fichier » contient le fichier logique référencé et les pattern « indic1 » et « indic2 » contiennent le numéro du premier et du second indicateur utilisé. Quand on a trouvé une telle ligne, il faut vérifier qu'il y a une erreur quand on ne trouve pas d'enregistrement dans le fichier, c'est-à-dire qand l'indicateur « indic1 » est à '1'. Ce test est effectué par le pattern « ref2 ».

Pour retrouver les différents champs concernés par la contrainte, on employera la même méthode que dans les cas précédents.

Cette façon de faire marche bien dans ce cas-ci parce que l'indicateur « indic1 » est toujours le même pour les contraintes de référence et ne sert que dans ce cas. En effet, si le même indicateur était utilisé à d'autres fins, la recherche du pattern « ref2 » risquerait de proposer une ligne n'ayant rien à voir avec une contrainte de référence. Pour ne pas tomber dans ce piège, il convient d'être attentif à la chose suivante : il ne peut y avoir de changement de la valeur de « indic1 » entre la ligne trouvée grâce au pattern « ref1 » et celle trouvée par le pattern « ref2 ».

La recherche de la combinaison « ref1 » - « ref2 » a produit toutes les contraintes de référence à trouver : on l'utilisera donc de la même façon que précédemment, c'est-à-dire comme nettoyage préalable des variables à examiner de façon fouillée.

5.5 Les différentes décompositions

Il arrive que des programmes utilisent des structures différentes pour le même fichier. C'est le cas des programmes PO155 et PO120 pour le fichier T$MA31#1.

programme PO155 :

```
452,00     I            DS
453,00     I                                    1   3 T$ZTAB
454,00     I                                    4   6 T$KEY
455,00     I                                    1  17 CKEYTA
```

programme PO120 :

```
108,70     I            DS
108,80     I                                    1  17 CKEYTA
108,90     I                                    1   3 T$ZTAB
109,00     I                                    4   5 T$CUR
109,10     I                                    6   70T$YY
109,20     I                                    8   90T$MM
```

Le champ « CKEYTA » est sauvegardé dans chacun des programmes dans le fichier T$MA31#1.

Ce champ est décomposé d'une manière différente par les deux programmes. Le programme PO155 utilise en fait les 6 premières positions du champ, tandis que le programme PO120 utilise les 9 premières, tout en le décomposant d'une manière différente. Il s'agit d'un cas de vues externes différentes d'un même fichier. On retiendra donc ces différentes vues de façon à les intégrer par la suite.

Dans l'atelier, ce genre de contraintes pose problème, car un attribut décomposable ne possède qu'une seule structure. Si une autre existe, comme dans ce cas-ci, il faudra l'expliciter dans la partie « sémantique » de la description du champ CKEYTA.

De telles décompositions sont aisées à retrouver grâce aux patterns : le symbole « DS » indique la présence d'une structure de données et il suffit d'employer le pattern suivant pour retrouver les différentes structures de données utilisées dans le programme.

```
var ::= mot_rpg;
pos1 ::= chiffre [chiffre] [chiffre];
pos2 ::= chiffre [chiffre] [chiffre];
decomp1 ::= num_ligne sep_rpg "I" sep_rpg @pos1 sep_rpg @pos2 sep_rpg @var
sep_ligne;
decomp ::= "DS" sep_ligne decomp1;
```

Le pattern « decomp » recherche les lignes de début de définition d'une structure de données. Le pattern « decomp1 » recherche les lignes de décomposition proprement dites. Il comprend trois variables : « pos1 », qui donne la position initiale, « pos2 », qui contient la position finale et « var » qui contient le nom du champ se trouvant entre les positions « pos1 » et « pos2 ».

Après avoir retrouvé les différentes décompositions, il reste à retrouver quelles structures font partie d'un fichier. Ceci se fait à nouveau de la même façon que pour les contraintes précédentes.

5.6 Les autres contraintes

Enfin, il reste les contraintes non exprimables de façon directe sur un schéma. Celles-ci seront explicitées dans la partie « sémantique » de la description d'un groupe reprenant les attributs à propos desquels « on veut dire quelque-chose ».

Ainsi, par exemple, on trouve dans le programme IC165 maintenant le fichier « CALNDR », on trouve :

```
  789,00     C          @BAXY     IFEQ 'Y'
  790,00     C          @BBAX     ANDNE'Y'
  791,00     C                    ADD  1          X
  792,00     C                    MOVE 'IC_4539' EMSG,X
  793,00     C                    MOVE 1         XERR
  794,00     C                    MOVEA'1'       *IN,35
  795,00     C                    MOVEA'1'       *IN,24
  796,00     C                    END
...
  829,00     C                    MOVE @NPROD    LPROD
  830,00     C                    MOVE @BBAX     LBAX
  831,00     C                    MOVE @PLAN     LPLAN
  832,00     C                    MOVE @COM      LCOMNT
  833,00     C                    MOVE @BAXY     LBAXY
  834,00     C                    MOVE @BAXMM    LBAXMM
  835,00     C                    MOVE @BAXYY    LBAXYY
```

Les lignes 789,00 à 796,00 ajoutent le message 'IC_4539' à la liste des messages d'erreur si la variable *@BAXY* = 'Y' et si la variable *@BBAX* ≠ 'Y'. La deuxième partie de code fait le lien entre les variables et les champs du fichier. On peut donc dire que si le champ *LBAXY* = 'Y', le champ *LBAX* vaudra également 'Y'.

Des contraintes beaucoup plus complexes ont été retrouvées dans ces programmes. Elles suivent globalement les mêmes règles et sont décrites en détail dans la description complète du schéma.

Au niveau de DB-MAIN, ces contraintes se retrouvent dans la partie « sémantique » d'un groupe contenant les champs concernés par la contrainte. Elles sont décrites en pseudo-langage.

De plus, il est impossible de définir des patterns de recherche pour de telles contraintes. Tout au plus peut-on remonter la chaîne des variables, à partir du « record format » décrivant les champs utilisés jusqu'aux variables testées, mais ça représente la quasi-totalité des variables et ça n'apporte pas beaucoup de résultat comparé au temps que ça prend. Une autre approche, un peu plus optimale, serait de partir de toutes les variables faisant l'objet d'un test et de redescendre jusqu'aux champs des fichiers. Dans ce cas, on utiliserait le pattern défini pour les contraintes sur le domaine. Le nombre de variables est alors plus restreint, surtout si on se limite aux variables faisant l'objet d'un test suivi de l'ajout d'un message d'erreur en cas de non-conformité.

Nous venons de passer en revue les différents types de contraintes qui se trouvent dans les programmes analysés. Ceux-ci ne contiennent évidemmment pas toutes les contraintes portant sur cette base de données, mais l'exhaustivité dans ce cas représente un travail beaucoup trop long vis-à-vis du temps disponible. Le but n'est pas de construire le schéma parfait, mais de mettre à l'épreuve la méthode de rétro-ingénierie décrite à la section 2.2 et l'atelier DB-MAIN.

Dans ce cadre, le nombre de programmes analysés et de contraintes retrouvées suffit amplement.

5.7 L'intégration des vues externes

Les seules vues externes présentes dans les programmes analysés sont celles du fichier « CURRENCY CONVERSION TABLE » contenant les valeurs de conversion des différentes devises.

En effet, celui-ci contient deux attributs décomposés de façons différentes par différents programmes. Le premier attribut, « CKEYTA », possède deux décompositions :

T$ZTAB (1-3)
T$CUR (4-5)
T$YY (6-7)
T$MM (8-9)
? (10-17)

T$ZTAB (1-3)
T$KEY (4-6)
? (7-17)

La solution utilisée a été de partitionner le type d'entités en deux sous-types : « KEY1 » et « KEY2 » correspondant respectivement à la première et la seconde décomposition de l'attribut « CKEYTA ». Le second attribut, « DATATA », possède quant à lui, trois décompositions différentes :

? (1-10)
DESC (11-50)

T$REF (1-20)
? (21-50)

? (1-22)
T$CONT (23-31)
T$CONS (32-40)
T$CONB (41-49)
? (50-50)

La même technique a été utilisée, en regroupant toutefois les deux dernières décompositions, vu que la partie inconnue de la dernière englobe le premier champ de la seconde et une partie de sa partie inconnue. Leur regroupement engendre donc la décomposition suivante :

T$REF (1-20)
? (21-22)
T$CONT (23-31)
T$CONS (32-40)
T$CONB (41-49)
? (50-50)

Après avoir effectué le double partitionnement du type d'entités « CURENCY CONVERSION TABLE », on trouve la partie de schéma de la Figure 25.

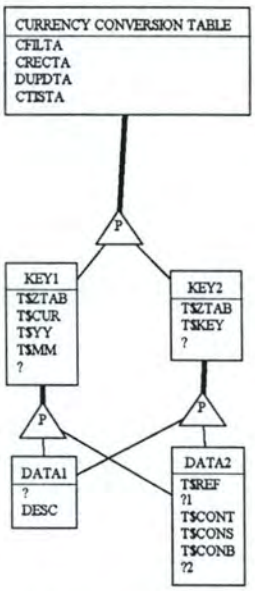


Figure 25 - Partitionnement de "CURRENCY CONVERSION TABLE"

Cette décomposition est étonnante par le fait que les différents découpages du fichier ne coïncident pas. Il faut alors passer par une intégration détournée, comme nous l'avons fait.

Toutes ces contraines ont été intégrées au schéma global, engendrant la version « Enriched » du schéma « Belgium-Malta », qui se trouve dans l'annexe 4.

Chapitre 6 : Le schéma conceptuel de base

La dérivation du schéma conceptuel de base consiste à transformer le schéma enrichi décrit dans le chapitre précédent de façon à générer un schéma conceptuel. Ceci se fait en trois étapes : la préparation du schéma, la détraduction et la désoptimisation. Ces deux dernières étapes se font généralement en parallèle.

6.1 La préparation du schéma

La préparation du schéma consiste à en éliminer les structures « inutiles » à un niveau conceptuel telles les structures d'accès, certaines structures redondantes,...

Dans notre cas, les structures d'accès sont les index : on n'en a retenu que ce qui concerne les identifiants, éliminant de ce fait les renseignements qui ne nous intéressent pas.

Il existe des champs techniques qui ne participent pas à la sémantique de la base de données : ce sont les champs contenant le dernier utilisateur ayant accédé à la table, la date et l'heure de la dernière modification, parfois également le programme ayant effectué la modification et/ou le terminal auquel était connecté l'utilisateur en question. On les retrouve chaque fois en fin de table. Tous ces champs ont été ajoutés pour des raisons techniques et forment en quelque sorte le schéma physique. Nous les avons donc éliminés. Par exemple, la table « STATUS CODE » correspondant au fichier « IPSP08 » contient les champs *PGMN08* (nom du programme), *USID08* (identificateur du dernier utilisateur), *WSTN08* (identificateur du terminal), *MDAT08* (date de la dernière mise à jour) et *MTIM08* (heure de la dernière mise à jour).

Un traitement important également effectué à ce moment est la standardisation des noms : les fichiers de départ contenaient des noms assez courts. En effet, leur longueur est limitée à 6 caractères par le système. De plus, chaque fichier possède un préfixe ou un suffixe propre, généralement de 1 ou 2 caractères, ce qui réduit d'autant le nombre de caractères disponibles pour le nom proprement dit.

Les nouveaux noms s'inspirent à la fois des anciens et de la description sémantique des champs. En effet, à chacun de ceux-ci est associé une description, ce qui permet d'y puiser l'inspiration nécessaire à ce renommage. Cette phase est très importante. En effet, les contraintes de référence apparaissent de façon beaucoup plus claire lorsque l'attribut de référence et l'attribut référencé portent le même nom. De plus, on perçoit plus facilement la sémantique des différentes tables en examinant les champs qui la composent lorsque ceux-ci portent des noms significatifs plutôt qu'une abréviation sur 6 caractères. Comment deviner, par exemple, que « QBPCD » signifie « Print CoDe » sans le savoir à l'avance ?

Nous avons également éliminé des identifiants qui n'étaient pas minimaux. Ceux-ci proviennent des fichiers logiques et/ou de l'analyse des programmes dans lesquels il arrive qu'un groupe d'attributs soit déclaré comme identifiant, alors qu'un sous-groupe de celui-ci l'est également. Dans ce cas, nous avons conservé l'identifiant minimal au détriment des autres. Certains identifiants sont composés des mêmes attributs qu'un autre et la seule différence est qu'ils ne sont pas dans le même ordre. Dans ce cas, nous n'en avons gardé qu'un, choisissant de

préférence celui qui est déclaré comme clé unique dans un fichier logique. Ainsi, par exemple, la table « SUPPLIER APPROBATION » correspondant au fichier « IPSP01 » contient trois identifiants, dont un seul est minimal.

Il arrive également qu'un attribut ou un groupe d'attributs ne soit identifiant que d'une partie des enregistrements de la table. C'est le cas lorsque l'index unique contient une condition. Ainsi, par exemple, le fichier d'index « CALNDR02 » contient un index unique (et donc identifiant) dans lequel apparaît la condition 'Non production date flag = 'Y''. Dans ce cas, le type d'entités « CALENDAR » correspondant au fichier logique est partitionné en deux sous--types : « NON PROD DATE CAL », dans lequel l'attribut *Non production date flag* possède la valeur 'Y' et « PROD DATE CAL », dans lequel ce même attribut possède une valeur différente de 'Y'. La Figure nous montre le résultat de ce partitionnement. On retrouve ce même phénomène au niveau des types d'entités « STATUS CODE », « PURCHASE MASTER » et « PURCHASE ITEM ». Toutefois, les deux derniers échappent à ce traitement car les contraintes les concernant sont déjà tellement complexes qu'un partitionnement à ce niveau-ci aurait pour effet de doubler le nombre de sous-types de ceux-ci. Nous verrons dans les chapitres suivants que ce nombre est très élevé pour ces deux types d'entités... La condition est donc conservée dans la description sémantique de l'identifiant.

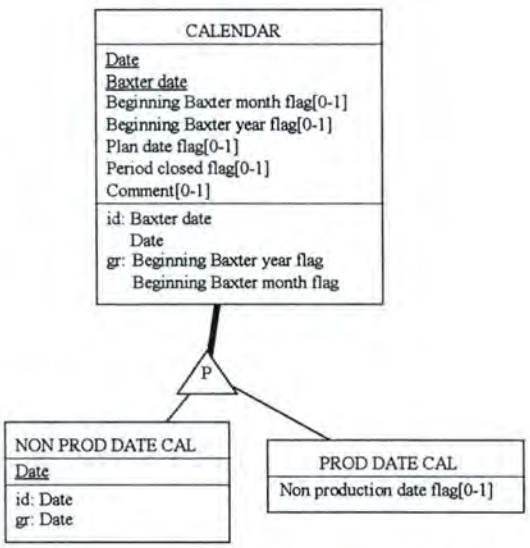


Figure 26 - Partitionnement du type d'entités "CALENDAR"

Toutes ces transformations ont été regroupées au sein de la version « Simplified » du schéma « Belgium-Malta », qui se trouve dans l'annexe 5. Nous allons maintenant entrer dans la partie concernant la détraduction et la désoptimisation.

6.2 La détraduction et la désoptimisation du schéma

Ces deux étapes s'effectuant en parallèle, il est difficile d'en parler séparément. Nous allons donc décrire les différentes transformations appliquées au schéma, sans spécifier s'il s'agit d'une transformation de détraduction ou de désoptimisation.

Tout d'abord, les contraintes de référence peuvent être transformées directement en types d'associations fonctionnels (voir transformation T - 9). C'est la première chose que nous ayons

faite, générant ainsi 15 types d'associations. Il arrive que certaines contraintes de référence contiennent une condition sur un champ d'un des types d'entités concernés par celle-ci. Dans ce cas, le type d'entités sur lequel porte la condition est partitionné en deux sous-types : le premier, qui satisfait la condition en question, et le deuxième, qui satisfait la négation de celle-ci.

C'est le cas, par exemple, de la contrainte de référence liant le champ *Status code* de la table « SUPPLIER APPROBATION » contenant les approbations des fournisseurs pour certains produits à l'identifiant de la table « STATUS CODE » contenant les codes d'approbation, à condition que la valeur du champ *Activity code* soit différente de 'D'. Dans ce cas, la table « STATUS CODE » est partitionnée en deux sous-types : le premier pour lequel le champ *Activity code* possède la valeur 'D', appelé dans ce cas « D-STATUS CODE », et le second pour lequel ce même champ possède une valeur différente de 'D', appelé « NON D-STATUS CODE ». La contrainte de référence peut à ce moment être transformée en type d'associations reliant les types d'entités « SUPPLIER APPROBATION » et « NON-D STATUS CODE ». Le résultat du traitement de cette contrainte se trouve à la Figure 27.

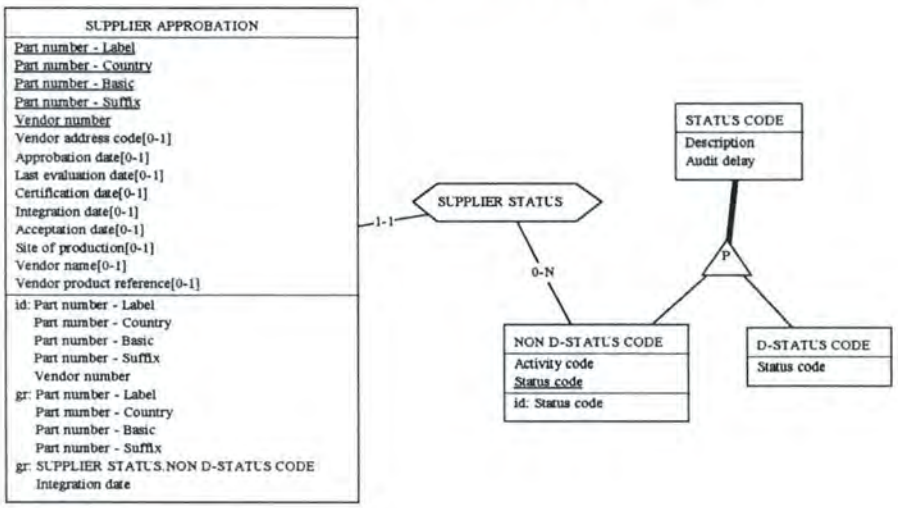


Figure 27 - Contrainte de référence sous condition

Pour le traitement des contraintes plus complexes, nous avons préféré diviser le travail en nous concentrant à chaque étape sur une partie différente du schéma. Ainsi, nous avons commencé par traiter la partie concernant les en-têtes et les lignes de commandes (avec les petites tables qui gravitent autour de celles-ci), ensuite la partie concernant la table des fournisseurs et enfin la partie concernant les termes de livraison et de paiement ainsi que les modes de livraison.

6.2.1 Les en-têtes et les lignes de commandes

Cette partie est de loin la plus complexe dans ce schéma. En effet, il suffit de comparer le nombre de groupes des tables « PURCHASE MASTER » (en-têtes de commandes) et « PURCHASE ITEM » (lignes de commandes) avec celui des autres tables pour s'en rendre compte.

Il est apparu assez rapidement que les commandes se divisaient en trois grands types : « Codified », « MRO » et « Non-stock ». Ces trois types correspondent respectivement à la valeur 'C', 'M' ou 'N' du champ *Order type* de la table « BUYER ANALYST », ce champ

étant présent également dans les tables « PURCHASE MASTER » et « PURCHASE ITEM » par dénormalisation. Selon la valeur de ce champ, les différentes commandes possèdent des propriétés différentes, ce qui nous a poussé à créer trois sous-types de chacune de ces tables, après avoir éliminé la dénormalisation relative au champ *Order type* grâce à la transformation T - 11.

Le type d'entités « PURCHASE ITEM » est partitionné en trois sous-types : « PI_CODIFIED » (correspondant à la valeur 'C' du champ *Order type*), « PI_MRO » (correspondant à la valeur 'M') et « PI_NONST » (correspondant à la valeur 'N'). Le même partitionnement est réalisé au niveau des types d'entités « PURCHASE MASTER » et « BUYER ANALYST». La Figure 28 représente cette partie de schéma. Les types d'entités ont été affichés sans leurs attributs, de façon à ne pas surcharger le dessin. Même si cette figure ne le suggère pas, les différents sous-types ont des comportements différents. Selon le cas, ils jouent un rôle spécifique dans l'un ou l'autre type d'associations.

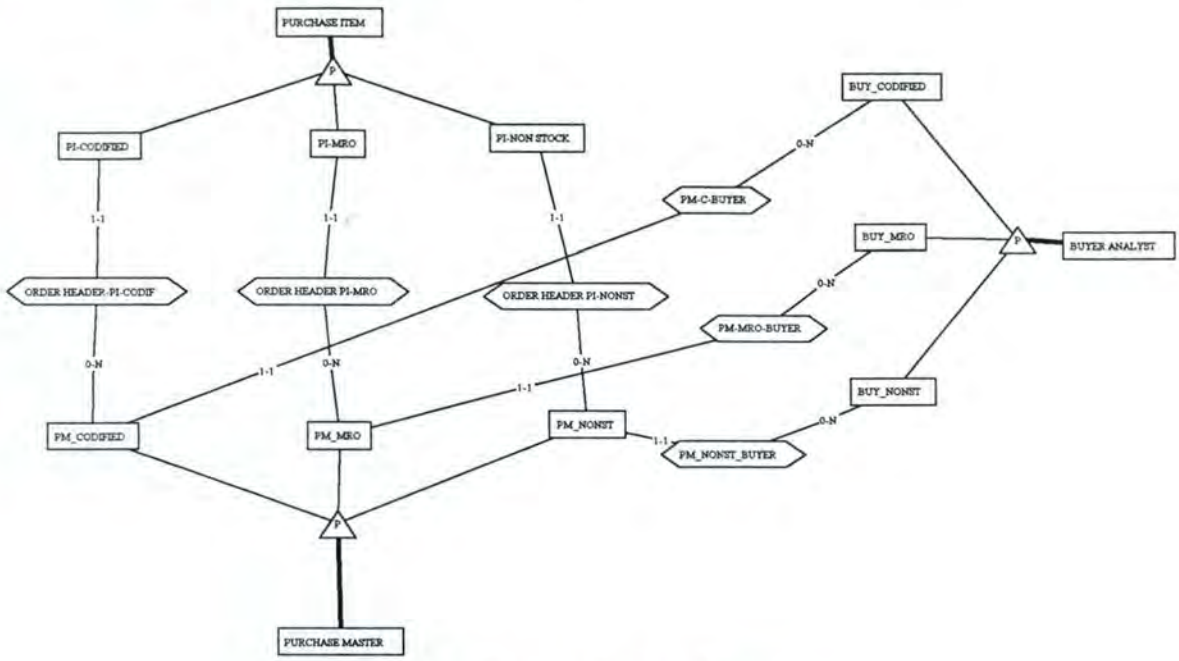


Figure 28 - Un premier partitionnement

On remarquera que cette transformation n'est pas spécifique à un processus précis, détraduction ou désoptimisation, mais résulte plutôt du traitement des contraintes non supportées par le SGBD.

Maintenant que les trois types d'entités auxquels nous nous intéressons sont partitionnés en trois sous-types chacun, nous pouvons distribuer les contraintes de façon plus ciblée vers l'un ou l'autre de ceux-ci. On pourra ainsi traiter les contraintes de façon personnalisée.

Au niveau des en-têtes de commande, les contraintes restent générales et il n'y a pas vraiment lieu de distinguer les différents sous-types.

6.2.2.1 Contraintes de « PURCHASE MASTER »

```
Si Vendor number <> '9999999'
Alors
   (Company code, Vendor number, Delivery method code, Currency code,
   Language code) -> VENDOR IDENTIFICATION.(Company code, Vendor number,
   Delivery method code, Currency code, Language code)
Sinon
   Delivery method code <> NULL
   Currency code <> NULL
   Vendor address code = NULL
```

On remarque qu'une distinction s'impose entre les entités dont la valeur de l'attribut *Vendor number* est égale à '9999999' et celles dont la valeur est différente de '9999999'. Dans le premier cas, l'attribut *Vendor address code* prend automatiquement la valeur nulle. Dans le second cas, on ne sait rien dire. Il est utile de faire remarquer que la valeur '9999999' de l'attribut *Vendor number* signifie que le bon de commande concerné est envoyé à un fournisseur occasionnel, qui ne fait donc pas partie de la base de données.

```
Warehouse code = NULL <=> Bill - to customer number <> NULL ou
                          Ship - to customer number <> 0
```

Cette contrainte reste telle quelle, vu qu'on ne peut la traduire directement dans le modèle utilisé.

```
Si Communic. method = 'X'
Alors
   Si Vendor address code <> NULL
   Alors
      VENDOR ADDRESS.(Telex number) <> NULL (VENDOR ADDRESS référencé)
   Sinon
      VENDOR IDENTIFICATION.(Telex number) <> NULL (VENDOR IDENTIFICATION
      référencé quand Vendor number <> '9999999')

Si Communic. method = 'F'
Alors
   Si Vendor address code <> NULL
   Alors
      VENDOR ADDRESS.(Fax number) <> NULL (VENDOR ADDRESS référencé)
   Sinon
      VENDOR IDENTIFICATION.(Fax number) <> NULL (VENDOR IDENTIFICATION
      référencé quand Vendor number <> '9999999')
```

Dans la partie concernant l'attribut *Communic. method*, le cas où l'attribut *Vendor address code* prend la valeur nulle est sujet à discussion : soit le type d'entités « VENDOR IDENTIFICATION » est toujours référencé (et dans ce cas, la deuxième possibilité de la première contrainte n'est jamais vérifiée), soit lorsque le type d'entités « VENDOR IDENTIFICATION » n'est pas référencé, l'attribut *Communic. method* ne prend ni la valeur 'X', ni la valeur 'F'. La sémantique du schéma suggère la seconde solution. En effet, l'attribut *Communic. method* représente la méthode employée pour communiquer avec le fournisseur. Or, lorsque le type d'entités « VENDOR IDENTIFICATION » n'est pas référencé, il n'y a pas lieu de retenir le moyen de communication, surtout que ce dernier est lié à un numéro (de fax ou de telex) qui est un attribut d'un des types d'entités. On supposera donc que c'est cette solution qui est la bonne, bien que nous n'ayons aucun moyen de le prouver.

Nous allons maintenant nous intéresser aux contraintes du type d'entités « PURCHASE ITEM ». Une partie de celles-ci concerne l'ensemble des sous-types déjà construits, une autre le sous-type « PI-CODIFIED », une autre le sous-type « MRO », une autre le sous-type « NON-STOCK » et une dernière les sous-types « MRO » et « NON-STOCK ».

6.2.2.2 Contraintes de « PURCHASE ITEM »

Contraintes globales

```
Si Part code - compressed <> NULL
Alors
   (OH.Company code, OH.Vendor number, Part code - basic,
   Part code - suffix, Part code - label, Part code - country) ->
   DEAL QTY/PRICE.(Company code, Vendor number, Part code - basic,
   Part code - suffix , Part code - label, Part code - country)

Rem : OH = ORDER HEADER = l'union des différents types d'associations
existant entre « PURCHASE ITEM » et « PURCHASE MASTER »
```

Cette contrainte référentielle est valable uniquement dans le cas où l'attribut *Part code - compressed* ne possède pas la valeur nulle. Elle suggère donc un partitionnement supplémentaire avec un sous-type correspondant au cas où cet attribut possède la valeur nulle et un sous-type dans le cas contraire. Une fois cette contrainte traduite sous forme de type d'associations, il faudra indiquer que l'attribut *Company code* du type d'entités « DEAL QTY/PRICE » possède la même valeur que l'attribut *Company code* du type d'entités « PURCHASE MASTER » accédé via le type d'associations « ORDER HEADER », ou plutôt le type d'associations reliant le sous-type de « PURCHASE ITEM » utilisé au sous-type de « PURCHASE MASTER » correspondant[1].

Mais avant de partitionner aveuglément les trois sous-types, il convient d'analyser les autres contraintes. Voici tout d'abord celles concernant le premier de ceux-ci : PI-CODIFIED.

Contraintes de « PI-CODIFIED »

```
Part code - compressed <> NULL ou Non stock Part <> NULL

Si Non stock Part = NULL
Alors


    Si (OH.Company code, Part code - Basic, Part code - Suffix,
    Part code - label, Part code - country, OH.Vendor number) ->
    PRICE DEAL HEADER.(OH.Company code, ..., Vendor Nnumber)
    Alors
       Contract number = PRICE DEAL HEADER.Contract number
    Sinon
       Contract number = NULL
```

[1] En effet, « ORDER HEADER » est un nom générique qui représente l'union des types d'associations reliant un sous-type de « PURCHASE MASTER » à un sous-type de « PURCHASE ITEM ». Ainsi, sur la **Erreur! Source du renvoi introuvable.**, ORDER HEADER = OH-PI-CODIF ∪ OH-PI-MRO ∪ OH-PI-NONST.

```
Stocking uom -> PART MASTER.(Unit of measure)
Buying uom -> PART MASTER.(Unit of measure)
(PART MASTER identifié par Part code - Compressed)
{Tot. bal due - buying uom} = {Order qty - buy. uom} - {Tot. qty rec.
 in buy. uom}
```

Ces contraintes suggèrent de partitionner PI-CODIFIED en deux sous-types : le premier correspondant aux enregistrements dont la valeur de « Non stock Part » est nulle et le second correspondant aux autres enregistrements. Le premier sous-type est directement concerné par le corps de la condition. Deux cas sont possibles pour celui-ci : soit le type d'entités « PRICE-DEAL HEADER » est référencé, soit pas. Dans le premier cas, l'attribut *Contract number* possède la même valeur que ce même attribut du type d'entités « PRICE DEAL HEADER ». C'est un cas typique de dénormalisation et la transformation ??????? nous invite à supprimer cet attribut. Dans le second cas, ce même attribut possède la valeur nulle. D'une manière générale, on peut donc l'éliminer, par suppression de la dénormalisation ou parce qu'il ne possède pas de valeur.

Le sous-type correspondant à la valeur nulle de l'attribut *Non stock Part* fait également l'objet d'une autre série de contraintes permettant de simplifier ce type d'entités : les attributs *Stocking uom* et *Buying uom* ont la même valeur que l'attribut *Unit of measure* du type d'entités « PART MASTER » identifié par l'attribut *Part code - compressed*. Il s'agit d'un cas de dénormalisation, qu'on peut éliminer en supprimant ces deux attributs. Une dernière contrainte indique une dépendance fonctionnelle, qui a été éliminée en supprimant l'attribut *Tot. bal due - buying uom*. En effet, cet attribut représente ce qui reste à recevoir du fournisseur et est calculé en soustrayant la valeur reçue (*Tot. qty rec. in buy. uom*) de la valeur commandée (*Order qty - buy. uom*).

La Figure 1 reprend la décomposition finale du type d'entités « PI-CODIFIED » en y incluant ses sous-types et les différents types d'associations dans lesquels ce type d'entités ou un de ses sous-types joue un rôle.

On remarque la division du type d'entités : d'une part le sous-type tel que l'attribut *Non stock Part* possède une valeur non nulle, et d'autre part le sous-type tel que l'attribut *Part code - compressed* possède une valeur non-nulle. Le T, présent dans le triangle représentant la relation de sous-typage, indique que la totalité des enregistrements de « PI-CODIFIED » sont répartis dans ses sous-types, ce qui est exprimé dans la première partie des contraintes relatives à celui-ci. Le type d'associations « PART PI-C-CMPR » provient du fait que le groupe d'attributs (« ORDER HEADER PI-C-CMPR.PURCHASE MASTER.Company code », « Part code - compressed ») référence le type d'entités « PART MASTER » (de façon à pouvoir retrouver les deux unités de mesure). Ceci est implicitement contenu dans la contrainte relative aux deux attributs *Buying uom* et *Stocking uom*. Malgré que le type d'entités « PART MASTER » ne soit pas explicitement identifié par le couple (*Company code*, *Part code - compressed*), on peut le deviner. En effet, « PART MASTER » est identifié par (*Company code*, *Part code - label*, *Part code - country*, *Part code - basic*, *Part code - suffix*), ces quatre attributs étant la décomposition de *Part code - compressed*. Aucune contrainte ne le spécifie dans le cas de « PI-CODIFIED », mais c'est le cas pour les deux autres sous-types de « PURCHASE ITEM » et le fait a été confirmé de vive voix par la personne chargée de la maintenance de ces fichiers. De plus, une analyse des données a permis d'appuyer cette hypothèse, vu qu'il n'existe aucun enregistrement possédant la même valeur pour le couple (*Company code*, *Part code - compressed*).

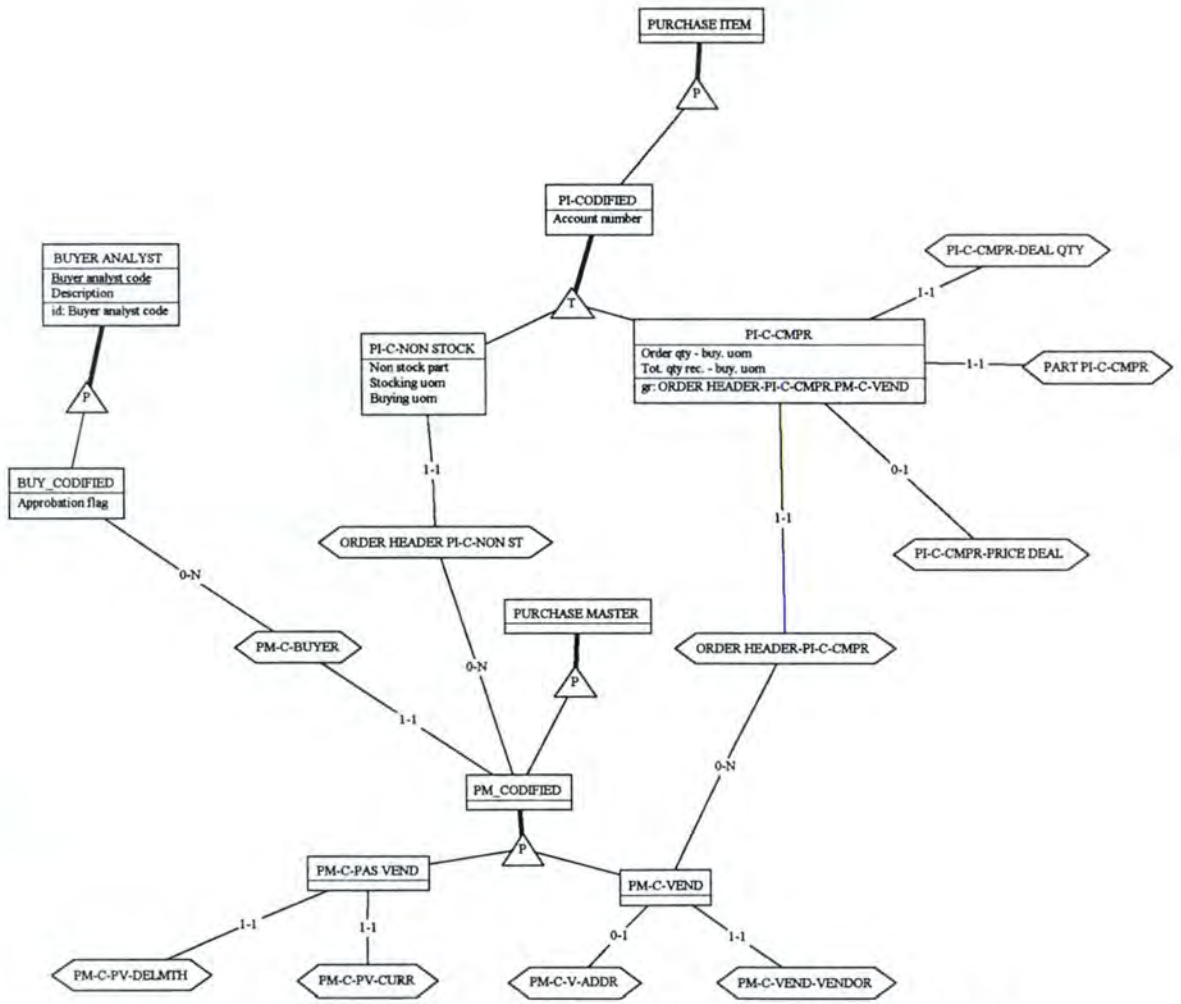


Figure 1 - Partitionnement du type d'entités "PI-CODIFIED"

Les deux autres sous-types de « PURCHASE ITEM », en l'occurrence « PI-MRO » et « PI-NON STOCK » possèdent des contraintes propres et des contraintes communes. Ces dernières seront dupliquées au sein de chaque sous-type. Voici donc les contraintes relatives à « PI-MRO ».

Contraintes de « PI-MRO »

```
MRO family, MRO group, MRO sequence, MRO request number <> NULL

Part code - Compressed = f(MRO family, MRO group, MRO sequence)

Account number = 0
```

Le fait que l'attribut *Account number* soit égal à 0 nous permet de le faire redescendre au niveau des sous-types. Il apparaîtra dans PI-CODIFIED, pas dans « PI-MRO » (puisqu'il y vaut 0) et dans certains des sous-types de « PI-NON STOCK » (voir à ce propos l'analyse des contraintes de ce type d'entités).

```
Si Non stock part <> NULL
Alors
  Contract number = NULL
  {Order qty - buy. uom} <> NULL
```

```
{Order qty in stock . uom} = {Order qty - buy. uom}
{Tot. balance due in stck. uom} =
        {Order qty in stock. uom} - {Tot. qty rec. in stock. uom}
{Tot. bal. due - buy. uom} =
        {Order qty - buy. uom} - {Tot. qty rec. in buy. uom}
Stocking uom <> NULL et -> UNIT OF MEASURE.Unit of measure
Buying uom <> NULL et -> UNIT OF MEASURE.Unit of measure
VAT code -> VAT CONTROL.VAT code
```

Ce groupe de contraintes dépend de la valeur de l'attribut *Non stock part*, ce qui suggère de partitionner « PI-MRO » en deux sous-types : le premier pour lequel la valeur de *Non stock part* sera nulle et le second pour lequel elle ne le sera pas. Ces deux sous-types sont appelés respectivement « PI-MRO PAS NON ST » et « PI-MRO-NON ST ». Ce dernier est directement concerné par la contrainte : l'attribut *Contract number* y contient la valeur nulle, l'attribut *Order qty in stock . uom* contient la même valeur que *Order qty - buy. uom*, et peut donc être éliminé, de même que les attributs *Tot. balance due in stck. uom* et *Tot. bal. due - buy. uom* pour les mêmes raisons que dans le cas de « PI-C-CMPR ». Enfin, les trois dernières lignes sont des contraintes de référence ordinaires qui sont directement transformées en type d'associations fonctionnels.

Dans le cas où l'attribut *Non stock part* contient une valeur nulle, on retrouve tous les attributs. Nous allons maintenant examiner les contraintes suivantes.

```
Si Part code - compressed <> NULL
Alors
  (Part code - label, Part code - country, Part code - basic,
  Part code - suffix) = f(Part code - compressed)


  (OH.Company code, Part code - label, Part code - country,
  Part code - basic, Part code - suffix) -> PART MASTER.(Company code,
  Part code - label, Part code - country, Part code - basic,
  Part code - suffix) tq PART MASTER.Activity code = 'A' et
                                  Buying to stock. uom conv. <> 0

  Si OH.BUYER ANALYST OF THE ORDER.Approbation flag = 'Y'
  Alors
     (Part code - label, Part code - country, Part code - basic,
     Part code - suffix, OH.Vendor number, OH.Vendor address code) ->
     SUPPLIER APPROBATION.( Part code - label, Part code - country,
     Part code - basic, Part code - suffix, Vendor number,
     Vendor address code)

  Si OH.Vendor number <> '9999999' et Non-stock part = NULL
  Alors
     OH.Company code, Part code - basic, Part code - suffix ,
     Part code - label, Part code - country, OH.Vendor number) ->
     PRICE DEAL HEADER.(PRICE DEAL SUPPLIER.Company code, Part code - basic,
     Part code - suffix , Part code - label, Part code - country,
     PRICE DEAL SUPPLIER.Vendor number)
```

La condition 'Part code - compressed <> NULL' est toujours vérifiée, puisque la valeur de cet attribut est fonction de celles de *MRO family*, *MRO group*, *MRO sequence* et *MRO request number*, qui sont toujours différentes de la valeur nulle. La fonction consiste en fait à agréger les valeurs des quatre attributs pour en former une seule.

La contrainte suivante est une contrainte de référence avec une condition. Elle sera donc transformée en type d'associations vers un sous-type du type d'entités « PART MASTER ». En effet, ce dernier doit être partitionné de façon à ce qu'un de ses sous-types satisfasse la condition 'Activity code = 'A' et Buying to stock. uom conv. <> 0'. Les deux sous-types sont

« A PART » et « OTHER PART ». En effet, l'attribut *Buying to stock. uom conv.* ne prend jamais la valeur 0 et il suffit donc de ne tenir compte que de l'attribut *Activity code*.

Les lignes suivantes nous suggèrent de partitionner les deux sous-types de « PI-MRO » en deux : un premier sous-type tel que la valeur de « OH.BUYER ANALYST OF THE ORDER.Approbation flag » = 'Y' et un second pour le cas contraire. La contrainte de référence valable dans le premier cas est directement transformée en type d'associations fonctionnel.

Il reste enfin à considérer, pour le type d'entités « PI-MRO PAS NON ST », les deux cas suivants : celui où l'attribut « OH.Vendor number » prend la valeur '9999999' et celui où il ne la prend pas. Ceci se matérialise par deux sous-types, dont un des deux jouera un rôle dans le type d'associations résultant de la transformation des attributs de référence cités dans la contrainte.

Le résultat de ces transformations nous donne la partie de schéma de la Figure 2.

Le dernier sous-type de « PURCHASE ITEM» est le type d'entités « PI-NON STOCK ». Les contraintes le concernant sont en grande partie les mêmes que celles de « PI-MRO ». La différence se situe au niveau des attributs *Non stock part* et *Part code - compressed*. En effet, le premier possède toujours une valeur non nulle pour le type d'entités « PI-NON STOCK », alors que ce n'est pas le cas pour « PI-MRO » et inversément. Grossièrement, le type d'entités « PI NON STOCK » se partitionnera de la même façon que « PI MRO », en inversant les rôles des deux attributs mentionnés ci-dessus. Mais examinons quand-même les contraintes de « PI NON STOCK ».

```
Non stock part <> NULL
```

Comme prévu, la valeur de l'attribut *Non stock part* n'est pas nulle.

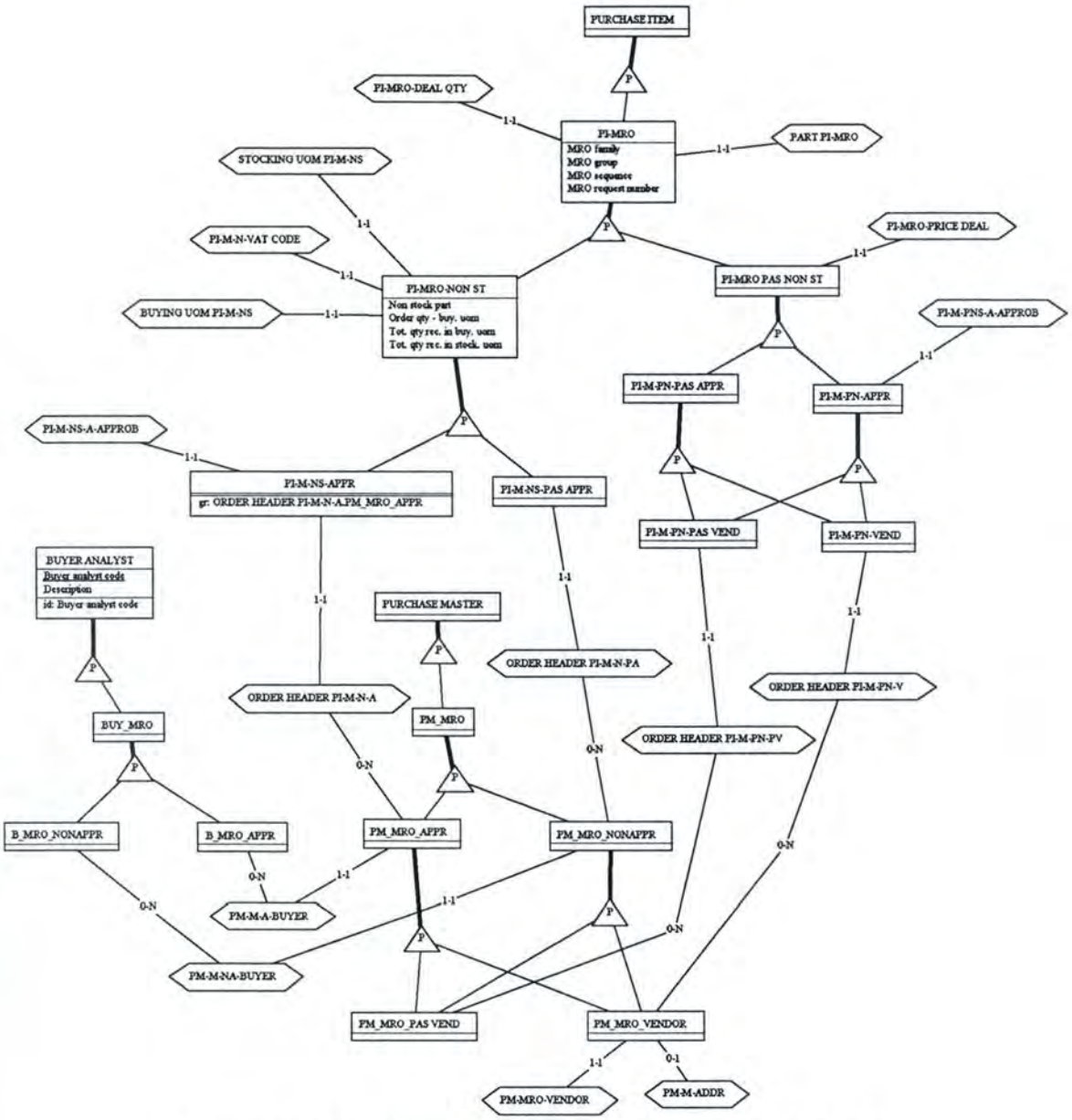


Figure 2 - Partitionnement du type d'entités "PI-MRO"

```
Si Non stock part <> NULL
Alors
  Contract number = NULL
  {Order qty - buy. uom} <> NULL
  {Order qty in stock . uom} = {Order qty - buy. uom}
  {Tot. balance due in stck. uom} =
           {Order qty in stock. uom} - {Tot. qty rec. in stock. uom}
  {Tot. bal. due - buy. uom} =
           {Order qty - buy. uom} - {Tot. qty rec. in buy. uom}
  Stocking uom <> NULL et -> UNIT OF MEASURE.Unit of measure
  Buying uom <> NULL et -> UNIT OF MEASURE.Unit of measure
  VAT code -> VAT CONTROL.VAT code
```

La condition 'Non stock part <> NULL' est évidemment toujours vérifiée, et on peut transposer ce qui a été dit pour le type d'entités « PI-MRO-NON ST » directement au niveau de « PI NON STOCK » (élimination de la dénormalisation et transformation des contraintes de référence en types d'associations fonctionnels).

```
Si Part code - compressed <> NULL
Alors
  (Part code - label, Part code - country, Part code - basic,
  Part code - suffix) = f(Part code - compressed)

  (OH.Company code, Part code - label, Part code - country,
  Part code - basic, Part code - suffix) -> PART MASTER.(Company code,
  Part code - label, Part code - country, Part code - basic,
  Part code - suffix) tq PART MASTER.Activity code = 'A' et
                                    Buying to stock. uom conv. <> 0

  Si OH.BUYER ANALYST OF THE ORDER.Approbation flag = 'Y'
  Alors
     (Part code - label, Part code - country, Part code - basic,
     Part code - suffix, OH.Vendor number, OH.Vendor address code) ->
     SUPPLIER APPROBATION.( Part code - label, Part code - country,
     Part code - basic, Part code - suffix, Vendor number,
     Vendor address code)

  Si OH.Vendor number <> '9999999' et Non-stock part = NULL
  Alors
     OH.Company code, Part code - basic, Part code - suffix , Part code -
     label, Part code - country, OH.Vendor number) -> PRICE DEAL
     HEADER.(PRICE DEAL SUPPLIER.Company code, Part code - basic, Part code
     - suffix , Part code - label, Part code - country, PRICE DEAL
     SUPPLIER.Vendor number)
```

Il faudra partitionner « PI NON STOCK » en deux sous-types : le premier, « PI-N-NON CMPR », tel que la valeur de *Part code - compressed* soit nulle et le second, « PI-N-CMPR », tel que cette même valeur soit non nulle. On remarque immédiatement l'analogie avec « PI MRO » où le même rôle était joué par l'attribut *Non stock part*. On peut alors répéter les mêmes raisonnements, ce qui nous amènera à partitionner le type d'entités « PI-N-CMPR » en deux sous-types, « PI-N-C-APPR », dans lequel la valeur de « OH.BUYER ANALYST OF THE ORDER.Approbation flag » soit égale à 'Y' et « PI-N-C-NON APPR » dans lequel la valeur de cet attribut soit différente de 'Y'. Enfin, le dernier groupe de contraintes n'est pas valable, puisque l'attribut *Non stock part* possède toujours une valeur non nulle.

```
Si OH.CPA Flag = 'Y'
Alors
  Part code - compressed = NULL ou Account number = 0

Si Part code - compressed = NULL
Alors
  Account number <> 0
```

Cette contrainte ne peut pas être traduite directement dans le modèle que nous utilisons. Elle restera donc telle quelle. La deuxième partie nous indique que les deux attributs ne peuvent pas avoir leur valeur nulle simultanément. La traduction de toutes ces contraintes donne lieu à la partie de schéma contenue dans la Figure 3.

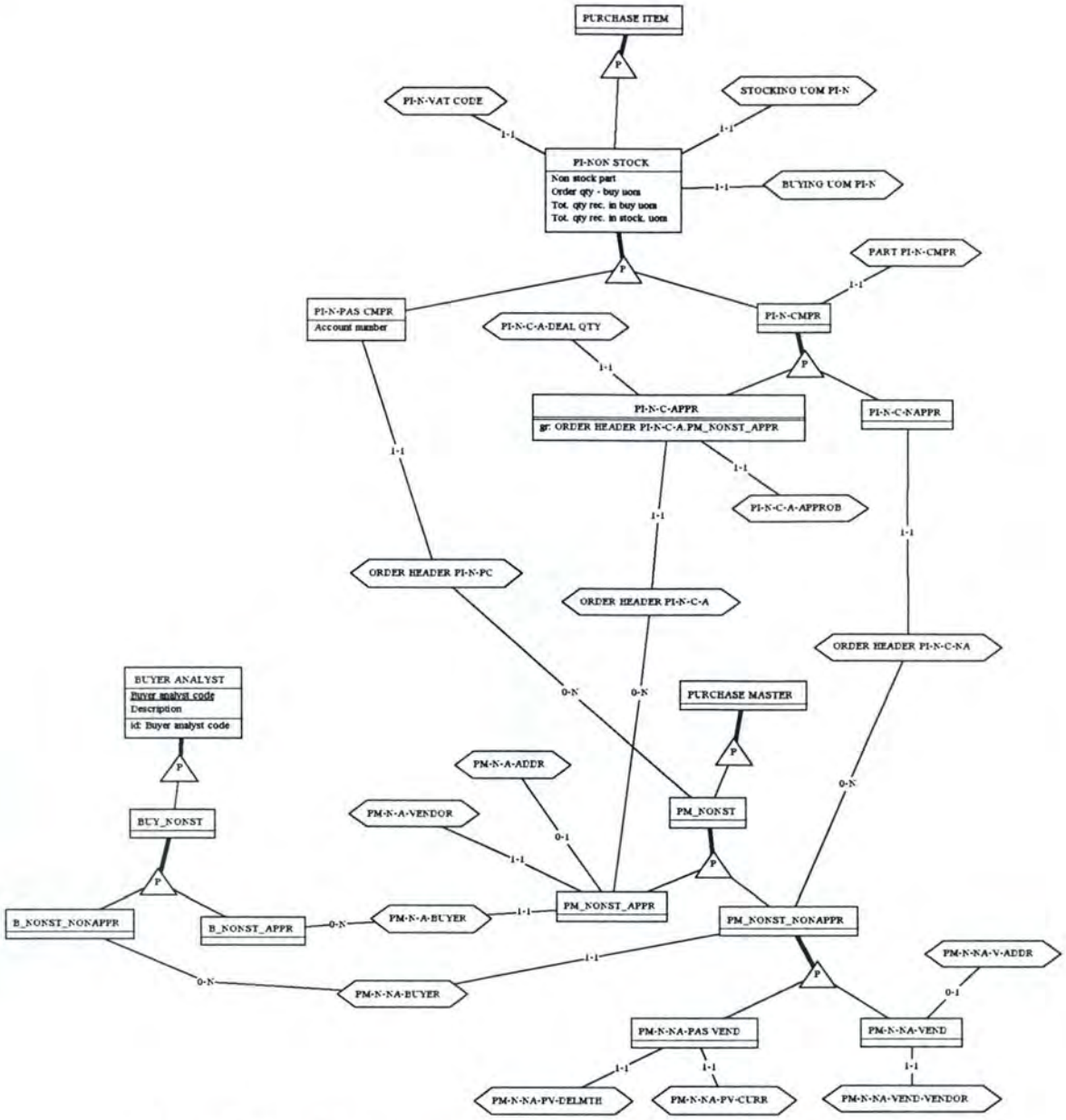


Figure 3 - Partitionnement du type d'entités "PI-NON STOCK"

Enfin, la Figure 4 nous montre le partitionnement final du type d'entités « PURCHASE ITEM ». Elle est extraite de la version « Conceptual-1 » du schéma « Belgium-Malta », qui reprend tous les résultats de l'analyse des contraintes effectuée jusqu'à présent. Ce schéma se trouve à l'annexe 6.

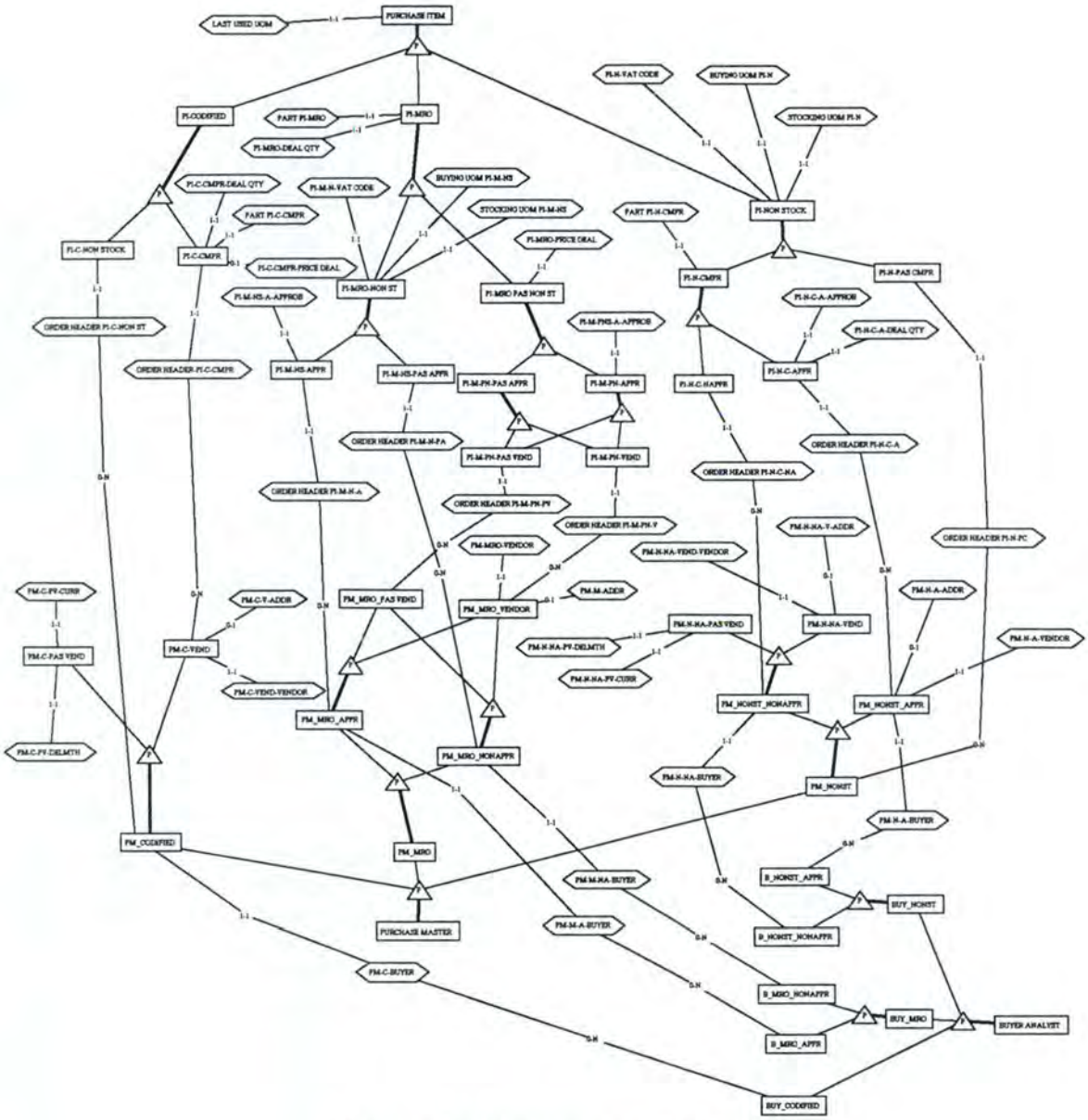


Figure 4 - Partitionnement final

6.2.2 Les fournisseurs

Les contraintes concernant la table « VENDOR IDENTIFICATION » des fournisseurs sont moins complexes que celles relatives aux commandes. Toutefois, on remarque un grand nombre de contraintes de référence et de groupes au sein de ce type d'entités.

Les contraintes de référence sont généralement composées de deux attributs, dont un est commun à plusieurs groupes. Par exemple, on retrouve une contrainte de référence du groupe d'attributs (*Delivery terms code*, *Language code*), une autre composée de (*Payment terms code*, *Language code*) et une troisième comprenant (*Delivery method code*, *Language code*).

On peut donc traduire ces contraintes en types d'associations auxquels est ajouté une contrainte d'égalité de l'attribut *Language code*. Le second cas concerne l'attribut *Company code* et est traité de la même façon.

Les groupes d'attributs définissant des contraintes qui ne sont pas directement supportées par le SGBD sont assez nombreux. Toutefois, la quasi-totalité d'entre-eux expriment des relations entre les attributs de « VENDOR IDENTIFICATION ». Seul le dernier groupe d'attributs contient une contrainte un peu particulière :

```
Exists DELMETHLEADTIME.(Company code, Vendor number, Language code) ->
VENDOR IDENTIFICATION.(Company code, Vendor number, Language code) and
DELMETHLEADTIME.(Delivery method code, Language code) ->
  DELIVERY METHOD.(Delivery method code, Language code)
```

La table « DELMETHLEADTIME » contient les délais de livraison. Ils dépendent en fait du fournisseur et de la méthode de livraison (Delivery method).

La contrainte nous apprend que, pour chaque fournisseur, il existe au moins un enregistrement dans la table « DELMETHLEADTIME » concernant ce fournisseur et dont la méthode de livraison, reprise dans la table « DELIVERY METHOD », est exprimée dans la langue du fournisseur.

Cette contrainte se traduit de la façon suivante.

Un type d'association « one-to-many », « LT-VENDOR », relie le type d'entités « DELMETHLEADTIME » au type d'entités « VENDOR IDENTIFICATION », avec la cardinalité de ce dernier égale à 1-N, de façon à exprimer l'existence d'un délai de livraison associé à ce fournisseur. Aucune contrainte ne spécifie que le couple (*Company code*, *Vendor number*) référence la table des fournisseurs. Toutefois, il serait stupide de créer un enregistrement contenant un délai de livraison concernant un fournisseur inconnu... Nous supposerons donc l'existence d'une telle contrainte, de manière à éviter le partitionnement des délais de livraison en deux : ceux qui référencent un fournisseur connu et ceux qui concernent un fournisseur fantôme...

De même pour le couple (*Delivery method*, *Language code*) de la table des délais de livraison : il serait également stupide de donner un délai de livraison pour une méthode inconnue... On créera donc un type d'associations fonctionnel « LT-DELMTH » entre la table « DELMETHLEADTIME » et « DELIVERY METHOD ».

De plus, il existe un type d'associations « DEF DELMTH », pour DEFault DELivery MeTHod, entre la table « VENDOR IDENTIFICATION » et la table « DELIVERY METHOD » résultant de la traduction d'une contrainte de référence.

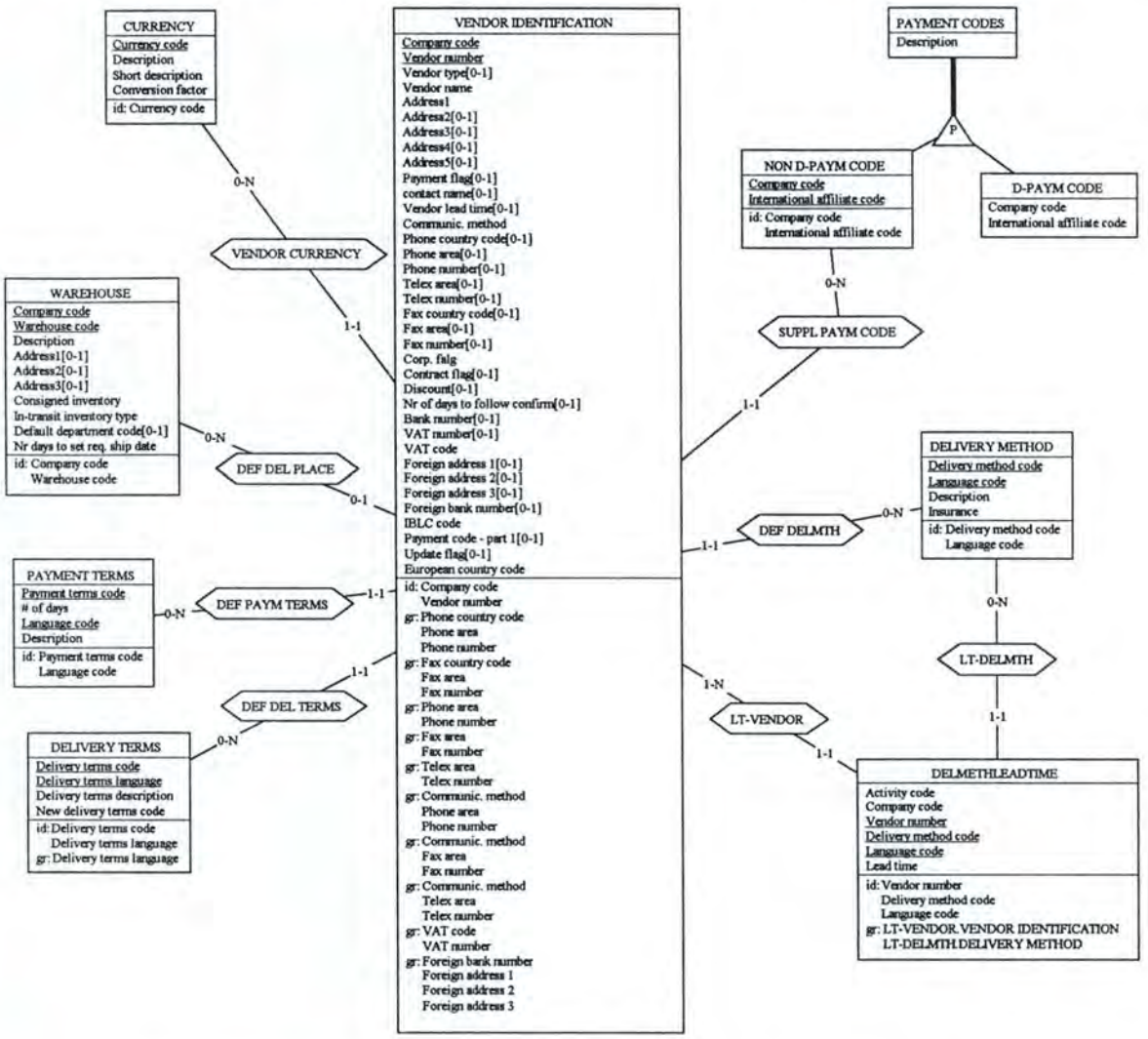
Dans ce contexte, la contrainte peut être exprimée de la façon suivante :

DEF DELMTH.DELIVERY METHOD.Langage code in
(LT-DELMTH o LT-VENDOR).Language code

Cette contrainte exprime que la langue du fournisseur (retrouvée grâce à la méthode de livraison par défaut de celui-ci) appartient à l'ensemble des langues utilisées dans les délais de livraison associés à ce fournisseur. Nous verrons plus loin que la langue associée à un mode de livraison est reprise dans le type d'entités « LANGUAGE CODE » et qu'un type d'associations fonctionnel « DELMTH_DELMTHLANG » relie ces deux types d'entités. Dès lors, la contrainte sera encore plus directe :

DELMTH_DELMTHLANG o DEF DELMTH in
DELMTH_DELMTHLANG o LT-DELMTH o LT-VENDOR

La Figure 5 nous montre le résultat final de ces transformations. Celle-ci est un extrait de la version « Conceptual-2 » du schéma « Belgium-Malta » qui se trouve à l'annexe 7. Ce dernier contient l'ensemble des structures qui ont été traitées jusqu'à présent. Il nous reste ensuite à examiner les contraintes des termes de livraison et de paiement et des modes de livraison.



DEF DELMTH.DELIVERY METHOD.Language code in (LT-DELMTH o LT-VENDOR).Language code

SUPPL PAYM CODE.PAYMENT CODE.Company code = DEF DEL PLACE.WAREHOUSE.Company code = Company code

DEF PAYM TERMS.PAYMENT TERMS.LANGUAGE CODE = DEF DEL TERMS.DELIVERY TERMS.LANGUAGE CODE = DEF DELMTH.DELIVERY METHOD.LANGUAGE CODE

Figure 5 - traitement du type d'entités "VENDOR IDENTIFICATION"

6.2.3 Les termes de paiement et de livraison et les modes de livraison

Ces trois types d'entités sont généralement référencés conjointement, ce qui implique systématiquement l'ajout de contraintes relatives à l'égalité de la valeur de l'attribut commun *Language code*. Le but ici est de transformer ces types d'entités de façon à simplifier ces contraintes au maximum.

Dans un premier temps, nous avons transformé les noms de façon à tenir compte du fait que la langue utilisée était prise en compte. Ainsi, chacun des trois types d'entités ont reçu le suffixe « -LANG ». Un type d'entités, que nous avons appelé « E » référence les trois tables. Si un cas se présentait où il n'y avait que deux de ces types d'entités qui étaient référencés, le raisonnement serait analogue. Cette situation de départ est reprise à la Figure 6.

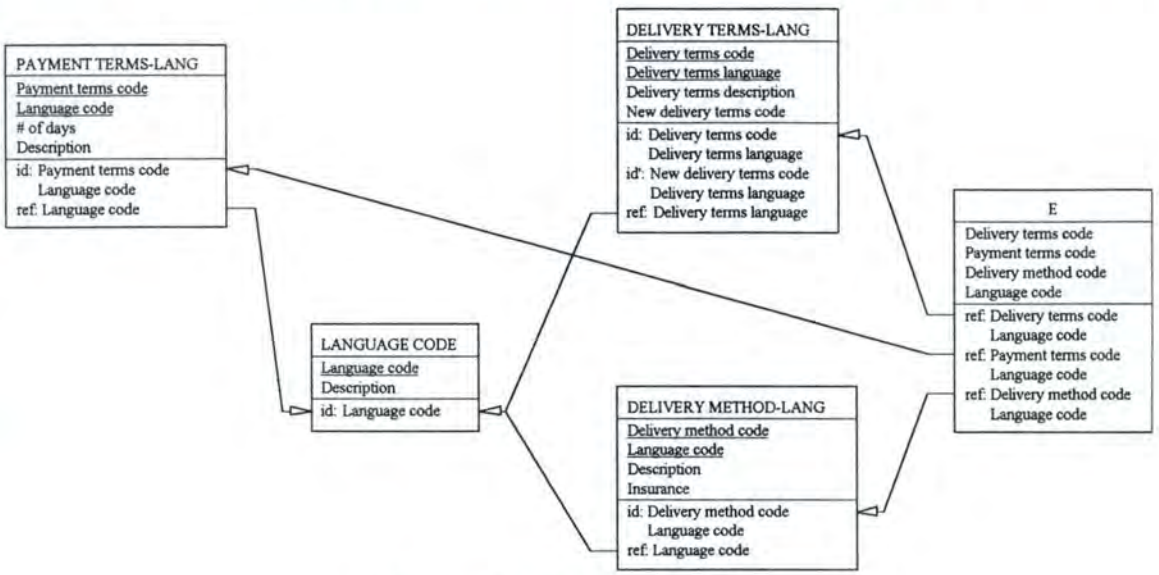
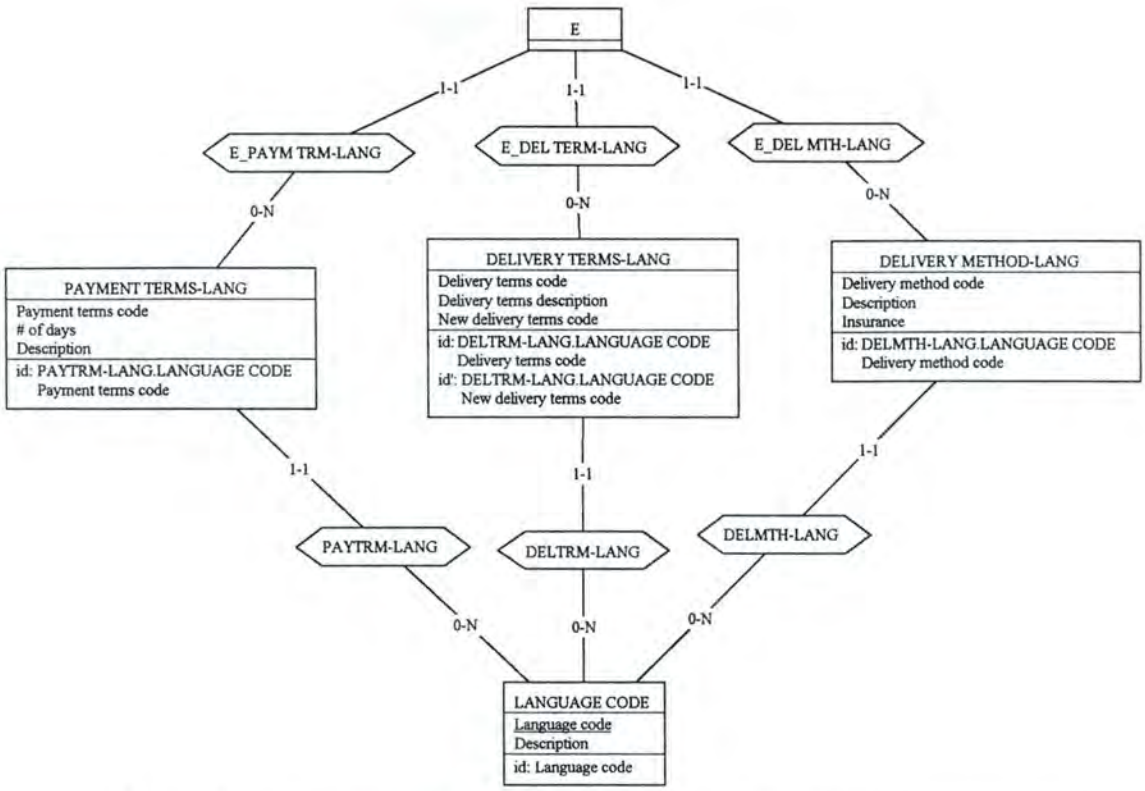


Figure 6 - Situation de départ

La première transformation effectuée est la transformation des contraintes de référence en types d'associations fonctionnels. Il faut commencer par celles du type d'entités « E », vu qu'elles contiennent l'attribut *Language code* faisant l'objet d'une autre contrainte de référence au sein des types d'entités référencés par « E ». Toutefois, étant donné qu'un attribut est commun à ces contraintes, il faut ajouter une contrainte indiquant l'égalité entre les attributs *Language code* retrouvés en suivant les trois chemins possibles. A ce moment, nous avons le schéma repris à la Figure 7.

Les contraintes d'égalité ne sont pas toujours faciles à retrouver, surtout quand ces structures font partie d'un schéma beaucoup plus grand et plus complexe. De plus, les trois types d'entités principaux contiennent une information double : celle concernant les termes de paiement, les termes de livraison ou les méthodes de livraison et celle concernant la langue utilisée. Prenons l'exemple des termes de paiement. Un bon concepteur trouverait sémantiquement plus correct de séparer la partie concernant la langue de celle concernant les termes de paiement eux-mêmes, d'autant plus que la langue est reprise dans un type d'entités à part entière. Cette idée nous suggère évidemment d'appliquer la transformation ??????? afin de représenter l'attribut *Payment terms code* par ses valeurs et de créer ainsi un type d'entités représentant les termes de paiement, indépendamment de toute considération linguistique.

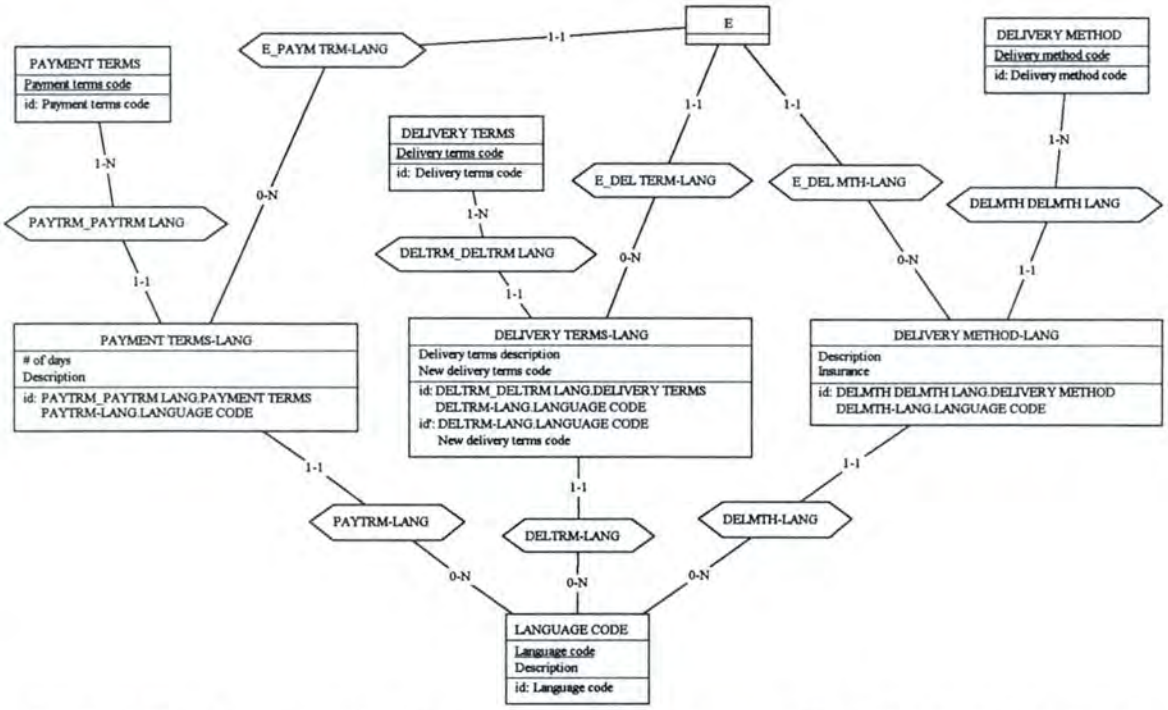


E.E_PAYM TRM-LANG.PAYMENT TERMS-LANG.PAYTRM-LANG.LANGUAGE CODE.Language code =
E.E_DEL TRM-LANG.DELIVERY TERMS-LANG.DELTRM-LANG.LANGUAGE CODE.Language code =
E.E_DEL MTH-LANG.DELIVERY METHOD-LANG.DELMTH-LANG.LANGUAGE CODE.Language code

Figure 7 - Transformation des contraintes de référence

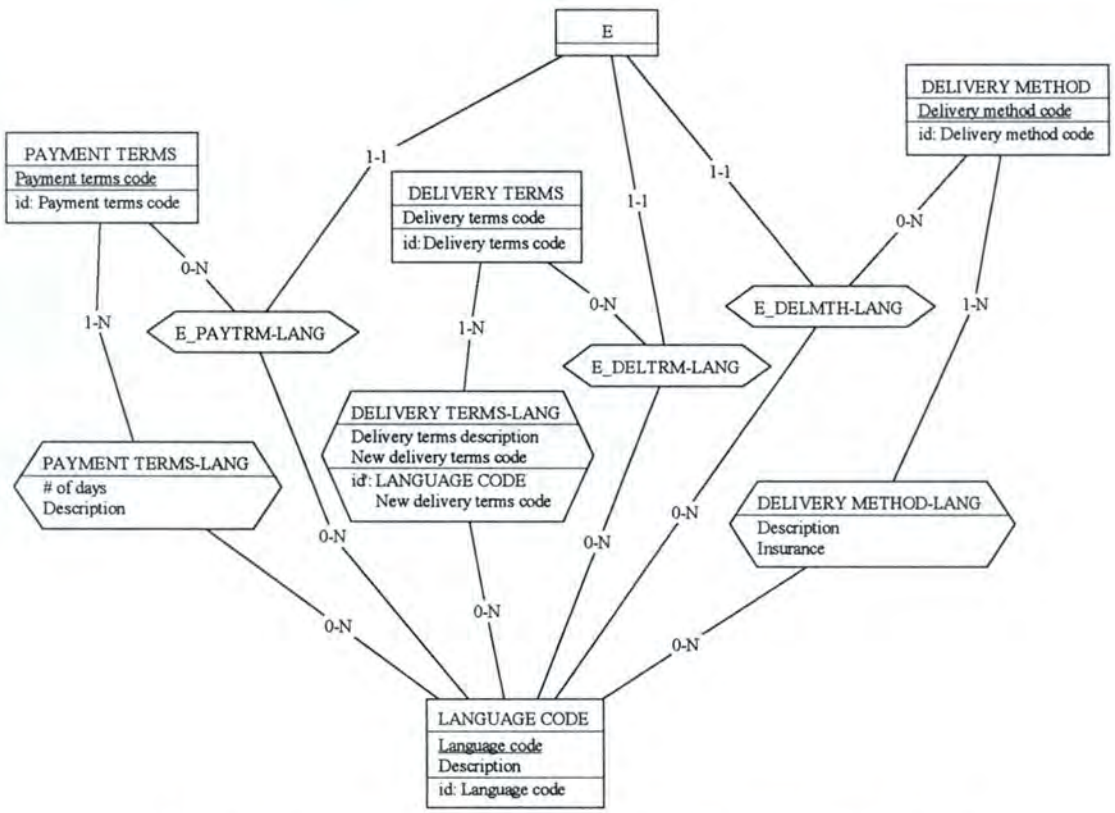
Après avoir appliqué cette transformation aux trois types d'entités concernés, on aboutit au schéma de la Figure 8.

Dans le même ordre d'idées, on pourrait trouver illogique de regrouper dans un type d'entités la rencontre entre un concept et un autre. En effet, en prenant à nouveau l'exemple des termes de paiement, on trouve dans le type d'entités « PAYMENT TERMS-LANG » des concepts relatifs à la fois aux termes de paiement proprement dits et à la langue utilisée. On pourrait arguer que ce genre de construction devrait plutôt se retrouver dans un type d'associations. Dans ce cas, la contrainte de référence pourrait se traduire par un type d'associations ternaire, comme on le voit dans la Figure 9.



PAYTRM-LANG o E_PAYTRM-LANG = DELTRM-LANG o E_DEL TERM-LANG = DELMTH-LANG o E_DEL MTH-LANG

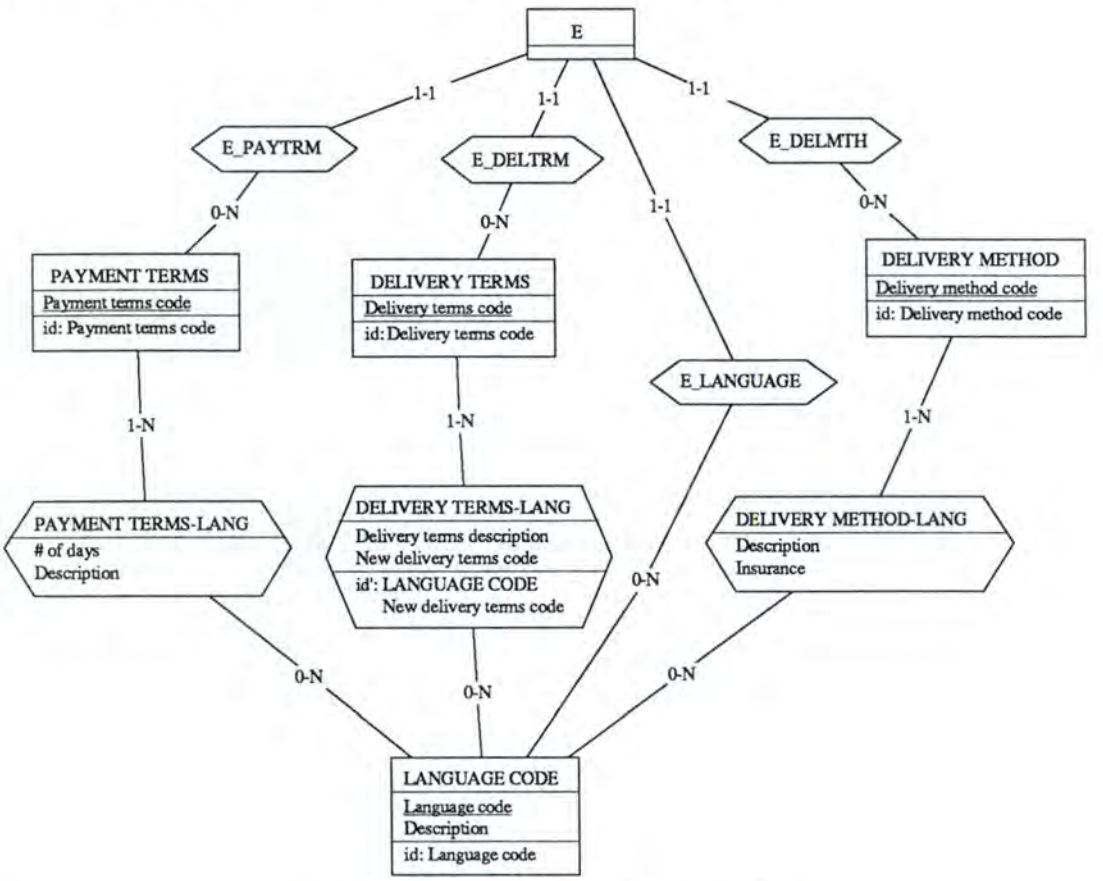
Figure 8 - Représentation des attributs par leurs valeurs



E_PAYTRM-LANG[PAYMENT TERMS, LANGUAGE CODE] ⊆ PAYMENT TERMS-LANG
E_DELTRM-LANG[DELIVERY TERMS, LANGUAGE CODE] ⊆ DELIVERY TERMS-LANG
E_DELMTH-LANG[DELIVERY METHOD, LANGUAGE CODE] ⊆ DELIVERY METHOD-LANG
E_PAYTRM-LANG[E, LANGUAGE CODE] = E_DELTRM-LANG[E, LANGUAGE CODE] = E_DELMTH-LANG[E, LANGUAGE CODE]

Figure 9 : Transformation des contraintes en types d'associations ternaires

La dernière ligne des contraintes du schéma de cette figure pourrait nous suggérer une dernière possibilité : transformer le type d'associations ternaire en deux types d'associations binaires. Le premier relie le type d'entités « E » au type d'entités « PAYMENT TERMS », « DELIVERY TERMS » ou « DELIVERY METHOD » et le deuxième relie « E » au type d'entités « LANGUAGE CODE ». La dernière ligne des contraintes de la Figure nous permet alors de simplifier le schéma en ne gardant qu'un seul des types d'associations entre « E » et « LANGUAGE CODE ». Cette situation est présentée à la Figure 38.



E_LANGUAGE ⊆ DELIVERY METHOD-LANG o E_DELMTH
E_LANGUAGE ⊆ DELIVERY TERMS-LANG o E_DELTRM
E_LANGUAGE ⊆ PAYMENT TERMS-LANG o E_PAYTRM

Figure 38 : Transformation des contraintes en types d'associations binaires

Au niveau du cas étudié, nous avons choisi la représentation de la Figure 35. Elle a l'avantage d'exprimer la contrainte d'égalité du code langage de la façon la plus concise et la plus simple. On aurait pu néanmoins choisir une autre représentation sans nuire à l'expressivité du schéma. Ce dernier est en fait la version « Conceptual-3 » du schéma « Belgium-Malta » que l'on trouve à l'annexe 8.

Le schéma que nous avons contruit dans ce chapitre est le schéma conceptuel de base : à partir d'un schéma conforme au SGBD utilisé et possédant encore des structures servant à l'optimisation des performances de ce dernier, nous avons construit un schéma plus expressif (quoique...) et indépendant de quelque SGBD que ce soit. Cette phase est importante et nécessite, dans ce cas-ci, une faculté d'abstraction. En effet, le traitement des contraintes les plus complexes, que nous avons détaillées ci-dessus, ne s'est pas fait sans mal : ce n'est qu'après quelques essais nécessitant chaque fois de recommencer le travail qu'une traduction

satisfaisante a été atteinte. Elle n'est sans doute pas encore la meilleure, quoique le concept de « meilleur schéma » varie d'une personne et d'une conception des choses à l'autre.

Il ne faut pas non plus oublier le travail effectué au niveau du graphisme : le placement d'un nombre aussi élevé de types d'entités et de types d'associations n'est pas chose facile et demande beaucoup de temps.

Chapitre 7 : Le schéma normalisé

La dernière étape restant à effectuer est la normalisation conceptuelle. Le but est de transformer le schéma conceptuel de base obtenu à la fin de l'étape de conceptualisation de base en un schéma conceptuel normalisé, qui est le schéma final auquel nous prétendons.

Le schéma conceptuel de base obtenu à la fin du chapitre 6 étant normalisé, il ne nécessite donc pas de transformation supplémentaire. Nous pouvons dès lors admirer le résultat de notre travail, qui est la version « Conceptual-3 » du schéma « Belgium-Malta » qui se trouve à l'annexe 8. Celui-ci est assez complexe et il est difficile de s'y retrouver. Pour en faciliter la lecture et la compréhension, nous allons présenter des vues différentes de ce schéma.

7.1 Vue générale

La Figure 39 nous montre le schéma simplifié. Les différents sous-types ont été agglomérés, ainsi que certains types d'entités gravitant autour d'un ou plusieurs autres, mais n'ayant pas d'importance en soi. De même, lorsque plusieurs types d'associations relient deux types d'entités, ils sont fusionnés. Le but ici est de montrer les liens entre les objets, sans entrer dans les détails.

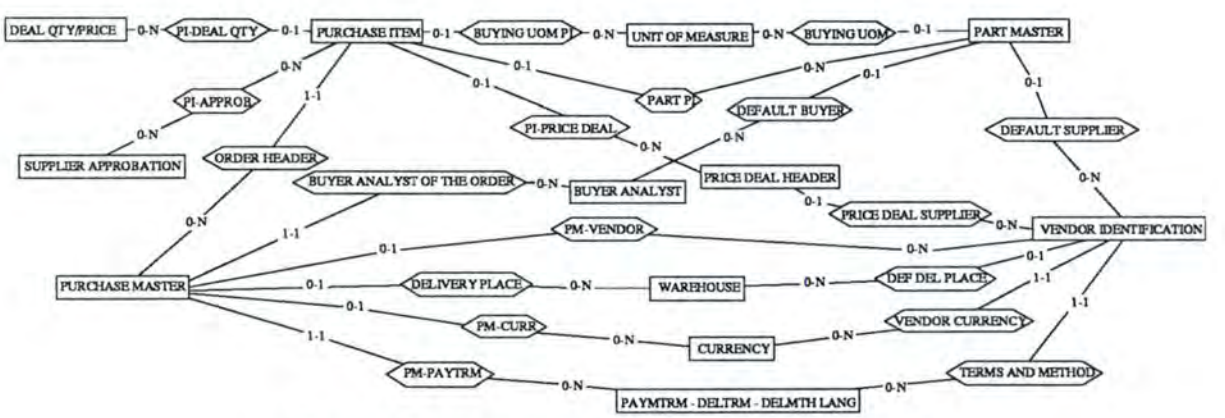


Figure 39 - Vue générale du schéma conceptuel

On remarque immédiatement les pôles principaux du schéma : le type d'entités « PURCHASE ITEM », en haut à gauche, représentant les lignes de commande, le type d'entités « PURCHASE MASTER », en bas à gauche, représentant les en-têtes de commande, le type d'entités « PART MASTER », en haut à droite, représentant les produits et le type d'entités « VENDOR IDENTIFICATION », en bas à droite, représentant les fournisseurs. Le schéma reflète donc a priori bien le concept d'achat dans une usine telle que Baxter, qui est la rencontre entre un fournisseur et des produits via une commande. Toutefois, ce schéma-ci est bien plus complexe que le schéma pédagogique connu de tous...

Nous allons maintenant nous intéresser à ces différents pôles afin de voir de quelle manière ils interviennent chacun dans le schéma. Pour cela, nous allons, pour chacun des quatre types

d'entités cités ci-dessus, extraire le sous-schéma composé du type d'entités en question, accompagné de tous les types d'associations dans lesquels il joue un rôle.

7.2 Le type d'entités « PURCHASE ITEM »

Le premier concept que nous allons examiner est celui des lignes de commande : le type d'entités « PURCHASE ITEM ». Le sous-schéma correspondant est repris à la Figure 40.

Nous avons caché certains attributs n'apportant rien à la compréhension. On remarque le partitionnement multiple du type d'entités, comme nous l'avions construit dans la section 6.2.1, et sur lequel nous ne reviendrons donc pas, ainsi que les différents types d'associations qui relient ses sous-types à ceux du type d'entités « PURCHASE MASTER ». Six autres types d'entités jouent un rôle vis-à-vis des lignes de commande. Tout d'abord, « DEAL QTY/PRICE », qui contient les prix des produits en fonction de la quantité commandée. Ensuite, « PART MASTER », qui contient les produits, « SUPPLIER APPROBATION », qui indique le mode d'approbation décerné au fournisseur de la commande, « VAT CONTROL », qui contient les codes de TVA valides, « UNIT OF MEASURE », qui indique les unités de mesure employées dans la commande et enfin « PRICE DEAL HEADER » qui contient le prix demandé par un fournisseur pour un produit.

7.3 Le type d'entités « PURCHASE MASTER »

Le concept suivant est celui des en-têtes de commande : le type d'entités « PURCHASE MASTER ». Le sous-schéma correspondant est celui repris à la Figure 41.

A nouveau, on retrouve le partitionnement multiple et les différents types d'associations vers les sous-types de « PURCHASE ITEM ». De plus, le type d'entités « BUYER ANALYST » est également partitionné en sous-types qui participent à des types d'associations vers les sous-types de « PURCHASE MASTER ». Nous avons donc trois types d'entités partitionnés l'un à côté de l'autre, ce qui complexifie évidemment le schéma de façon notoire. En plus de ceux-ci, nous retrouvons les fournisseurs (« VENDOR IDENTIFICATION »), le type d'entités « CURRENCY », qui nous indique la devise dans laquelle est calculé le montant de la commande, les trois types d'entités « DELIVERY METHOD LANG », « DELIVERY TERMS LANG » et « DELIVERY METHOD LANG » dont nous avons également déjà parlé. Enfin, on découvre 5 types d'entités qui gravitent autour des en-têtes de commande. Trois de ceux-ci contiennent des commentaires, un autre indique le lieu de livraison de la commande et le dernier contient l'adresse du fournisseur à laquelle celle-ci est envoyée.

La grande complexité de ce schéma provient surtout du fait que certains sous-types jouent un rôle au sein de types d'associations vers le même type d'entitiés, multipliant de ce fait le nombre de types d'associations vers ce dernier. On peut le remarquer pour les types d'entités « BUYER ANALYST », « VENDOR IDENTIFICATION », « CURRENCY », « DELIVERY METHOD LANG » et « VENDOR ADDRESS ».

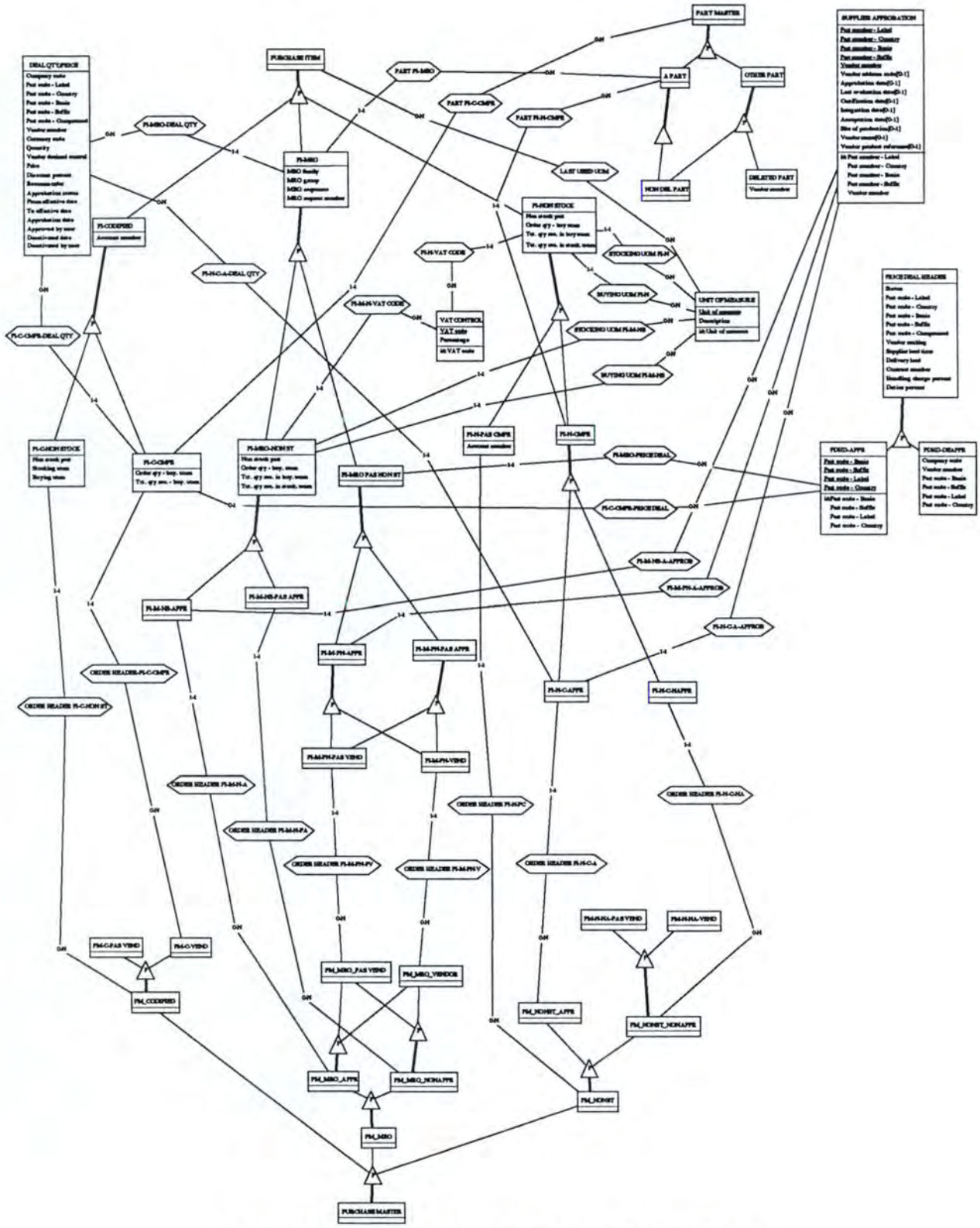


Figure 40 - Le type d'entités "PURCHASE ITEM"

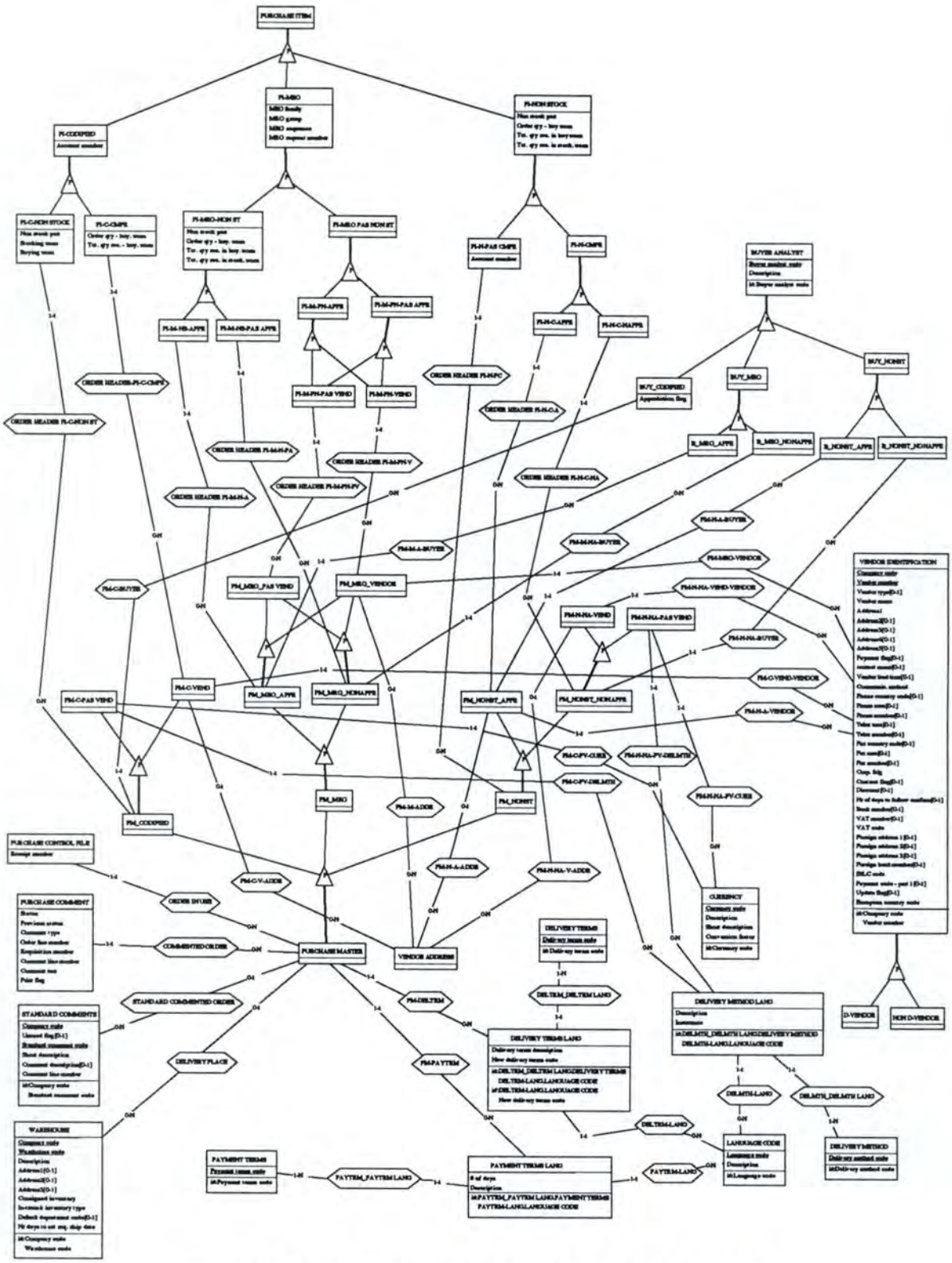


Figure 41 - Le type d'entités "PURCHASE MASTER"

7.4 Le type d'entités « PART MASTER »

Le troisième pôle du schéma général est le type d'entités « PART MASTER » contenant les produits. Nous représenterons les liens qu'il possède avec les lignes de commande de façon aggrégée, afin de ne plus reproduire l'ensemble des sous-types de celles-ci. La Figure 42 nous montre la partie de schéma relative à ce type d'entités.

Le type d'entités « PART MASTER » se comporte de façon moins extravagante que les deux précédents, malgré un partitionnement sur deux niveaux. Il est relié à trois sous-types de « PURCHASE ITEM », représenté par le type d'associations « PARTPI » sur le schéma, aux fournisseurs ainsi qu'à leur addresse, pour connaître le fournisseur habituel du produit, à l'acheteur qui commande habituellement le produit et à l'unité de mesure dans laquelle il est comptabilisé. Enfin, nous y avons ajouté le type d'entités « PART CROSS REFERENCE » qui permet de faire le lien entre les deux représentation du code d'un produit.

7.5 Le type d'entités « VENDOR IDENTIFICATION »

Il nous reste le type d'entités « VENDOR IDENTIFICATION » contenant la liste des fournisseurs de Baxter. Le sous-schéma lui correspondant se trouve dans la Figure 43.

Le type d'entités « VENDOR IDENTIFICATION » est entouré d'un nombre important de satellites... On retrouve tout d'abord des figures connues : « PART MASTER » et « PURCHASE MASTER », qui sont respectivement les produits et les commandes se rapportant au fournisseur concerné. Dans le même cas, on retrouve les types d'entités suivants : « PDHD-APPR », qui est un sous-type de « PRICE DEAL HEADER » contenant le prix proposé par un fournisseur pour un produit, « VENDOR COMMENT », qui contient des commentaires à propos d'un ou plusieurs fournisseurs, « DELMETHLEADTIME », qui contient les délais de livraison concernant un fournisseur pour un mode de livraison précis et enfin « VENDOR ADDRESS », qui contient l'adresse d'un fournisseur et dont le type d'associations « ADDRESS VENDOR » devrait, selon toute logique, être du type « one-to-one ». Mais la contrainte le spécifiant n'est pas reprise dans les programmes analysés.

Ensuite, nous avons les types d'entités qui apportent de l'information directement (c'est-à-dire via un type d'associations fonctionnel du fournisseur vers ceux-ci). Il s'agit des types d'entités « WAREHOUSE », qui contient le leiu de livraison par défaut du fournisseur, « NON D-PAYM CODE », qui est un sous-type de « PAYMENT CODE » contenant des codes relatifs au paiements effectués à ce fournisseur, « CURRENCY » contenant la devise du fournisseur et enfin les trois types d'entités « DELIVERY METHOD LANG », « DELIVERY TERMS LANG » et « PAYMENT TERMS LANG » contenant la méthode de livraison et les termes de paiement et de livraison habituels du fournisseur.

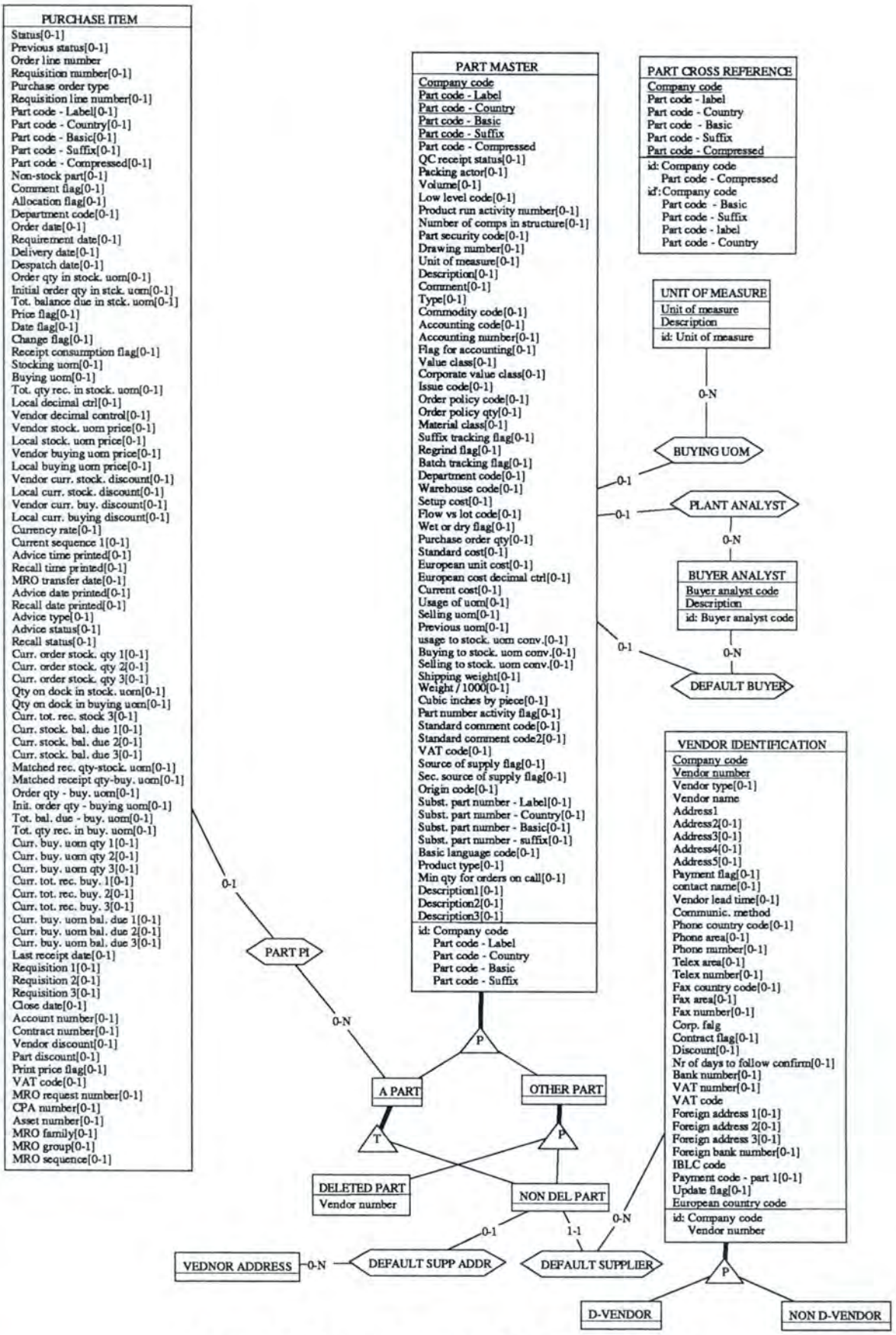


Figure 42 - Le type d'entités "PART MASTER"

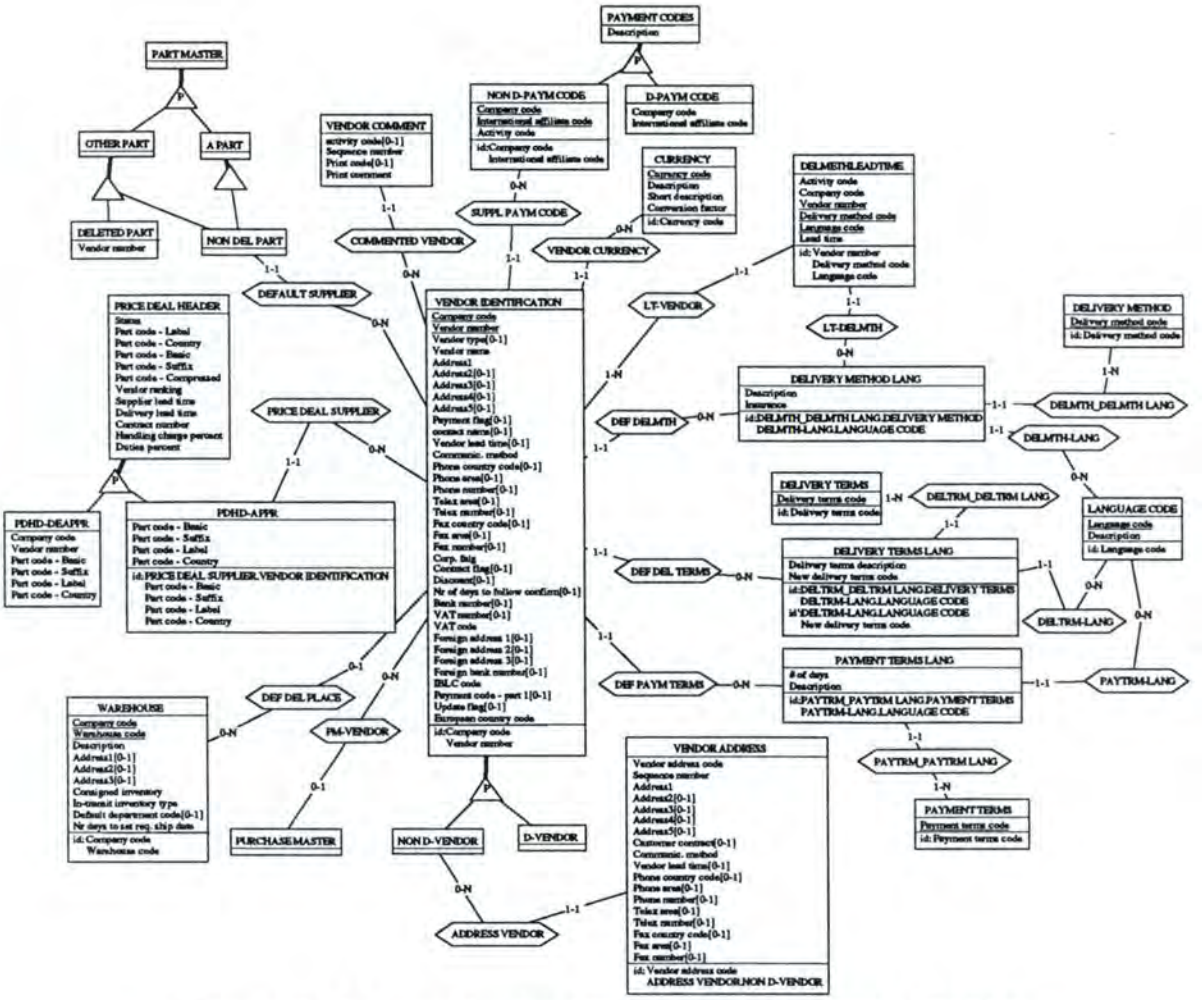


Figure 43 - Le type d'entités "VENDOR IDENTIFICATION"

Quelques types d'entités ne sont pas représentés sur ces quatre schémas. En fait, ils ne jouent de rôle dans aucun type d'associations, mais contiennent des renseignements annexes ayant peu d'influence sur le comportement global du schéma.

Il est difficile de proposer d'autres vues dérivées du schéma général, vu sa complexité et son étendue. Avec un peu d'habitude, on arrive quand-même à s'y retrouver et les vues dérivées de celui-ci qui viennent d'être proposées forment un support à l'éclaircissement.

Maintenant que nous avons terminé l'analyse de cette base de données et que nous en avons tiré un schéma conceptuel, il nous reste à prendre un peu de recul vis-à-vis de ce travail et de nous demander dans quelle mesure la méthode de rétro-ingénierie et l'atelier DB-MAIN nous ont été utiles. Cette analyse critique est faite dans le chapitre suivant, avant que le dernier chapitre ne conclue ce travail.

Chapitre 8 : Analyse critique de la méthode et de l'outil employés

Après un bref rappel de la méthode de rétro-ingénierie employée, nous allons parcourir chacune de ses étapes et nous demander dans quelle mesure cette méthode est appropriée pour ce genre de travail. En effet, par définition, une méthode est théorique et est généralement appliquée à une petite étude de cas pour en expliquer le fonctionnement. Nous avons eu l'occasion de la tester sur une base de données plus conséquente et on peut se demander quelles conclusions en tirer. Pour les appuyer, nous fournirons quelques chiffres quantifiant la base de données et l'application de la méthode (nombre de types d'entités et autres, nombre de lignes de code analysées, temps passé pour telle ou telle étape de la méthode,...).

Tout au long de ce parcours, nous nous poserons les mêmes questions pour l'atelier DB-MAIN, qui a servi de support logiciel à ce travail. Des choix ont été faits lors de son implémentation et certains ne sont peut-être pas judicieux, tandis que d'autres forment un véritable support à la rétro-ingénierie. Enfin, certains aspects ne sont pas encore implémentés et mériteraient de l'être. Ce sont ces questions que nous aborderons à propos de l'atelier.

8.1 Rappel de la méthode

Nous allons faire un bref rappel de la méthode utilisée, afin de baser notre raisonnement sur des idées précises.

La rétro-ingénierie est composée de deux grandes étapes : l'extraction des structures de données, qui fournit un schéma logique, et la conceptualisation des structures de données, qui a pour but de transformer le schéma logique en un schéma conceptuel.

L'extraction des structures de données comporte principalement l'extraction du schéma global et l'analyse des programmes. Lorsque les programmes utilisent chacun une partie du schéma et qu'il n'est pas possible de procéder à une extraction globale, il est nécessaire de faire une intégration des différentes vues des programmes.

La conceptualisation des structures de données est un processus utilisant les transformations de schéma. Après la simplification du schéma logique, la détraduction et la désoptimisation sont effectuées en parallèle pour fournir un schéma conceptuel de base. Il ne reste qu'à le normaliser pour obtenir un schéma conceptuel normalisé, qui est le but principal à atteindre dans cette méthode.

8.2 Généralités

Nous allons quantifier dans cette section un certain nombre de choses, de façon à se faire une idée plus ou moins précise de la base de données utilisée.

Au départ, nous avons sélectionné, avec l'aide de la personne responsable de la maintenance de cette base de données, un total de 31 fichiers. Ces fichiers ont engendré de façon immédiate 31 types d'entités contenant entre 2 et 100 champs, avec une moyenne de 17.5 champs. On ne dénombre évidemment pas de types d'associations à ce stade-ci, car le schéma est sous forme relationelle. Après avoir effectué toutes les transformations, le schéma général contient 89 types d'entités et 74 types d'associations. Le nombre de types d'entités a donc été multiplié par 3 et on dénombre presqu'un type d'associations par type d'entités, ce qui fait en moyenne presque deux rôles par type d'entités. Le nombre total d'attributs a diminué, à cause des attributs de référence qui ont disparu, mais il n'est pas intéressant de les dénombrer, vu qu'un nombre non négligeable de types d'entités sont des sous-types d'un autre et ne contiennent pas ou presque pas d'attributs. A noter également que l'augmentation du nombre de types d'entités est due principalement aux partitionnements de « PURCHASE MASTER » et « PURCHASE ITEM ». A partir de deux fichiers au départ, on a construit 32 types d'entités et 10 types d'associations les reliant. Pour mémoire, le schéma final occupe un peu plus de 0.8 m^2.

Une caractéristique importante de ce schéma est le nombre de sous-types. Les 31 fichiers au départ ont engendré 58 sous-types, dont la palme revient, nous les avons déjà cités, aux types d'entités « PURCHASE MASTER » et « PURCHASE ITEM ».

Il existe 5 fichiers pour lesquels on n'a trouvé aucun lien avec un autre, auxquels on ajoute les deux types d'entités « COMPANY » et « COUNTER », qui sont reliés entre-eux, mais pas au reste du schéma. Si on avait analysé un plus grand nombre de programmes, on aurait certainement découvert d'autres liens. Les noms des attributs et la sémantique du schéma laissent d'ailleurs supposer d'autres contraintes de référence et autres.

Nous allons maintenant reprendre les différentes étapes de la méthode et les commenter de manière critique, positive ou négative. Le même travail sera fait en parallèle concernant l'atelier DB-MAIN.

8.3 Critique de la méthode et de l'atelier

Cette critique suit une organisation chronologique : les étapes de la méthode sont prises dans leur ordre d'application, et la critique de l'atelier s'insère aux endroits où il est utilisé. Tout au long du parcours, nous insèrerons des commentaires relatifs au temps passé à chaque étape. Ceci permettra d'évaluer cette méthode sous un angle supplémentaire.

8.3.1 L'extraction du schéma brut

La première remarque à formuler est que dans l'étude de cas que nous venons de réaliser, nous n'avons pas beaucoup parlé de différentes vues en fonction des programmes. Le schéma global était disponible tel quel et les programmes suivaient la décomposition de celui-ci. Seul, le

fichier « CURRENCY CONVERSION TABLE » échappe à cette règle et fait l'objet de décompositions différentes.

La méthode suppose généralement qu'un texte source, composé par exemple d'instructions SQL, est disponible. Ce fut notre cas, puisque les fichiers sont décrits sous la forme d'une énumération des champs et des index. On est loin des instructions SQL, mais on peut considérer que le texte descriptif rentre dans le concept de « code source ». Ce texte est d'ailleurs plus parlant que le langage SQL : chaque table est décrite dans un fichier et les champs sont énumérés. Les index sont décrits chacun dans un fichier séparé, ne mélangeant pas les structures d'accès avec le schéma proprement dit.

Au niveau de l'atelier, on peut imaginer l'implémentation d'un parser, tel qu'il existe déjà pour le langage SQL, qui analyserait une liste de fichiers comme ceux que nous avons utilisés et créerait automatiquement les tables et les contraintes d'intégrités contenues dans ceux-ci. Il est possible d'ajouter des fonctionalités de ce type via une fonctionnalité de DB-MAIN que nous n'avons pas encore citée : « Voyager ». Celle-ci est apparue dans l'atelier lorsque ce mémoire était pratiquement terminé et n'a donc pas pu être prise en compte.

Une autre amélioration est possible : lorsqu'une table comporte plusieurs champs en commun, il devrait être possible de les sélectionner ensemble et de les recopier dans la table cible. Cette situation fait bien sûr penser aux contraintes de référence. En effet, l'emploi d'une facilité comme celle-ci pourrait déjà faire naître l'idée qu'une contrainte existe à cet endroit, alors que l'encodage systématique des champs les uns à la suite des autres ne le fait pas.

Le support idéal que peut apporter l'atelier à ce niveau-ci est bien sûr le parser de texte source. Il n'existait pas dans notre cas, mais est présent pour les modèles de données les plus courants.

Les descriptions des fichiers physiques comprenaient 31 fichiers de texte. Ceux-ci comportaient un peu plus de lignes que le nombre de champs du fichier. Ils n'étaient donc pas très longs et leur analyse était relativement aisée. Les fichier logiques étaient également faciles à analyser : après avoir éliminés ceux qui ne comportaient pas le mot-clé « UNIQUE », il suffisait de déclarer comme identifiant les attributs énoncés dans le fichier.

Le temps passé à cette étape est relativement élevé : Le choix des fichiers, l'impression et l'encodage dans l'atelier ont pris plus ou moins 5 jours. Il est évident que l'utilisation d'un « parser » aurait réduit fortement ce nombre. Mais dans notre cas, les descriptions des champs ont dû être entrées à la main et cela prend beaucoup de temps.

A ces 5 jours, on peut en ajouter 2 passés dans le département « Achat » de l'entreprise Baxter pour en comprendre le fonctionnement. Ca ne fait pas partie de l'extraction des schémas, mais ce temps fait partie du temps passé pour procéder à la rétro-ingénierie de la base de données.

8.3.2 L'analyse des programmes

L'analyse des programmes a été une étape importante dans notre étude de cas. Elle a pris beaucoup de temps, pour ne retrouver qu'une partie des contraintes...

L'atelier n'est pas d'un grand secours pour cette étape. Les contraintes sont introduites à la main : on doit créer un groupe d'attributs et /ou de rôles et exprimer la contrainte via un

mécanisme présent dans l'atelier ou pas (dans ce cas, la contrainte est exprimée dans la description sémantique du groupe). On peut en outre déplorer la non-existence de représentation des dépendances fonctionelles et multivaluées.

De plus, il n'existe pas de mécanisme pour retrouver des contraintes redondantes ou contradictoires. Nous avons eu plusieurs cas d'identifiants non minimaux et qui pourraient être repérés automatiquement sans trop de problème par l'atelier. On pourrait également imaginer d'autres types de contraintes que l'atelier pourrait supporter, alors qu'il ne le fait pas (encore) : le fait que la valeur d'un champ soit supérieure à une autre, les contraintes de référence sous conditions, que nous avons souvent rencontrées,...

Au niveau de l'analyse des programmes, nous avons déjà évalué quelque-peu la facilité de « pattern matching » de l'atelier. Elle permet de retrouver des patterns dans des programmes, mais agit à un niveau local : l'atelier trouve l'endroit où on « parle » de la variable, mais on ne sait pas d'où vient la valeur ni dans quel champ de quel fichier elle se retrouvera. De plus, l'utilisation de cet outil pour rechercher des contraintes un peu plus complexes comme des identifiants ou des contraintes de référence nécessitent de comprendre déjà à la fois la syntaxe et la sémantique du programme. Dès lors, on peut se demander si cette facilité est compétitive face à une analyse manuelle des listings des programmes... Comme nous l'avons déjà dit, il nous semble que les deux doivent cohabiter et que le « pattern matching » doit servir à cibler les recherches lors de l'analyse manuelle.

On pourrait imaginer également une fonctionnalité de l'atelier qui suggèrerait les contraintes, de référence par exemple, en se basant sur le format des champs et éventuellement leur noms.

Les 24 programmes totalisent ensemble environ 37.000 lignes de code. On peut faire remarquer à ce sujet qu'un des programmes monopolise plus de 13.000 lignes à lui tout seul, ce qui accroît d'autant la difficulté de l'analyse. Personne ne sera surpris d'apprendre qu'il s'agit du programme de maintenance des fichiers « PURCHASE MASTER » et « PURCHASE ITEM »... Ces programmes ont été écrits pour la plupart entre 1980 et 1985. Certains datent des années 90' et même de 1994. Les listings ne sont donc pas tellement poussiéreux...

L'analyse des programmes est l'étape qui demande le plus de temps. Il se décompose dans ce cas-ci en plusieurs tranches. Il a fallu tout d'abord 2 jours pour se familiariser avec le système informatique et apprendre les rudiments du langage RPG nécessaires à l'analyse de programmes. Ensuite, 2 jours ont été nécessaires pour la recherche et l'impression des programmes. Il fallait examiner une liste énorme de programmes, et décider, sur base d'un commentaire d'une demi-ligne, si celui-ci concernait un fichier de la base de données ou pas. Tous les programmes intéressants n'ont pas été trouvés, et on peut se demander quel serait le temps nécessaire pour en connaître la liste... Le parcours des 25 listings à la main a pris environ 45 jours. Cette analyse s'est faite sans aucune aide de l'atelier, car la fonction de « pattern matching » n'était pas encore implémentée à ce moment. A nouveau, quel serait le temps nécessaire pour retrouver toutes les contraintes ??? De plus, le retranscription des contraintes dans l'atelier a pris 4 jours : certaines contraintes complexes sont difficiles à traduire sous une forme lisible et nécessitent plusieurs essais et erreurs. Il faut savoir également que la version de l'atelier de l'époque était instable et qu'une sauvegarde était nécessaire après chaque opération élémentaire.

Pour ces deux étapes (l'extraction du schéma brut et l'analyse des programmes), nous avons suivi la méthode décrite, en éliminant les parties qui ne nous concernent pas (principalement l'extraction et l'intégration des vues). Elle nous paraît satisfaisante et suit la logique naturelle.

En effet, par exemple, on ne va pas rechercher les contraintes avant d'avoir extrait les structures de données globales...

8.3.3 La simplification du schéma

Cette étape, consistant à éliminer les structures redondantes ou inutiles, est indispensable avant la conceptualisation : un schéma brut contient des structures d'accès et autres qu'il convient d'éliminer, car elles ne font pas partie de la sémantique de la base de données.

L'atelier, comme nous l'avons dit ci-dessus, ne procure pas de moyen de détection de ces structures redondantes. Par exemple, des identifiants non minimaux apparaissaient régulièrement dans le schéma, et il ne serait pas tellement compliqué d'implémenter une fonctionnalité de détection de ce genre de structures redondantes. Il existe déjà une transformation qui élimine les clés d'accès, ce qui est un premier pas.

Cette étape est courte et n'a pris que quelques heures, qu'on peut évaluer approximativement à une demi-journée.

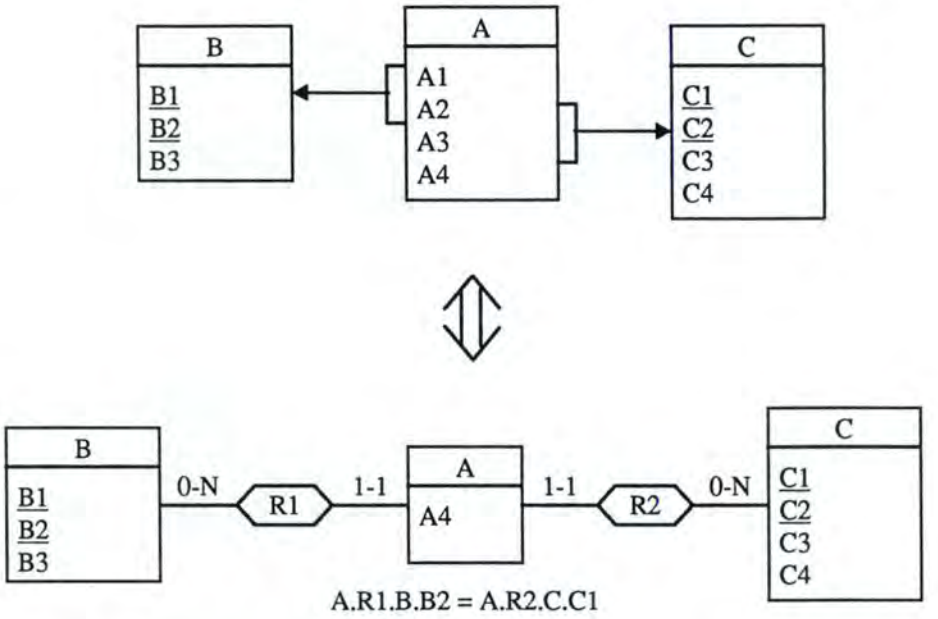
8.3.4 La détraduction et la désoptimisation

Ces deux étapes étant exécutées en parallèle, nous les traitons de la même façon.

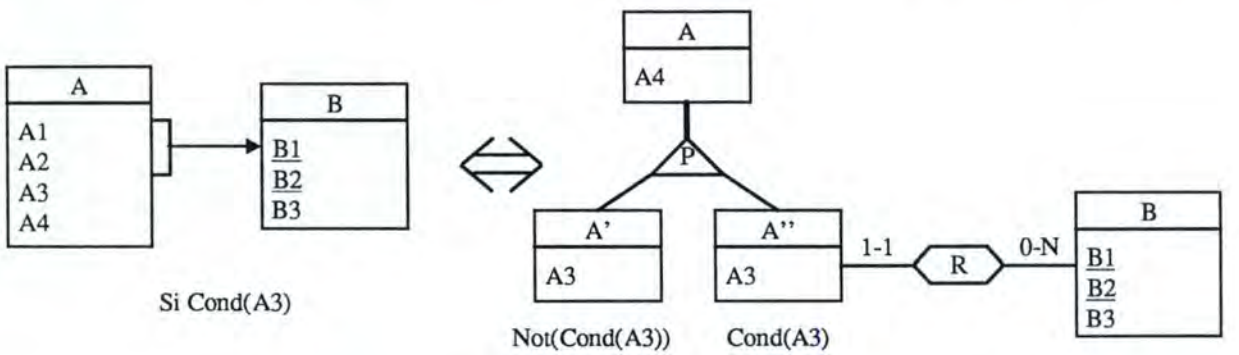
On entre ici dans le domaine des transformations de schéma. Celles-ci sont utilisées, dixit la méthode, afin de conceptualiser le schéma. Or, dans notre cas, bon nombre de contraintes ont dû être traitées sans faire appel à ces transformations. Ce sont surtout les contraintes que l'on retrouve dans les parties sémantiques des descriptions. Souvent, le traitement de celles-ci engendrait un ou des sous-type(s). Dans certains cas, les transformations apportées au schéma étaient assez systématiques et pourraient faire l'objet d'un complément à la liste des transformations utilisées dans la méthode.

C'est le cas, tout d'abord, de plusieurs groupes d'attributs de référence possédant un attribut commun. Dans ce cas, chacun des groupes est transformé en type d'associations et une contrainte supplémentaire exprime l'égalité de l'attribut commun (voir transformation T - 20). Cette transformation est utilisée à maintes reprises lors de la détraduction et de la désoptimisation du schéma simplifié. Nous l'avons appliqué systématiquement lorsque le cas se présentait et le résultat était satisfaisant.

Une autre concerne les contraintes de référence avec une condition sur un champ d'un des types d'entités concernés. Nous avons alors systématiquement généré deux sous-types de celui-ci : le premier satisfaisant la condition et le second satisfaisant sa négation. De la sorte, la contrainte de référence était « descendue » au niveau du bon sous-type, sans condition supplémentaire cette fois, et pouvait être transformée en type d'associations de façon tout-à-fait classique. Nous retrouvons ces concepts dans la transformation T - 21.



T - 20 : Transformation de groupes d'attributs de référence avec un attribut commun



T - 21 : Transformation d'une contrainte de référence sous condition

Au niveau de l'atelier, plusieurs améliorations peuvent voir le jour. Tout d'abord, certaines transformations de la méthode ne sont pas applicable. L'élimination de la dénormalisation (transformation T - 10) en est un exemple typique. Ceci est dû en partie au fait que les dépendances fonctionnelles ne sont pas représentées non plus. Si c'était le cas, cette transformation ne poserait pas plus de problème qu'une autre. La deuxième transformation concernant la dénormalisation (transformation T - 11) n'est pas non plus présente dans l'atelier, à cause certainement de l'impossibilité de représenter la contrainte d'égalité entre attributs de deux types d'entités différents reliés par un type d'associations. Cette transformation est pourtant utilisée à maintes reprises : la dénormalisation est une technique employée abondamment pour éviter les jointures ou le parcours d'un chemin d'accès trop long. Il semble donc important qu'une telle transformation soit présente dans l'atelier.

Le problème pourrait être contourné via la possibilité de définir des contraintes de référence vers des identifiants non-minimaux : le groupe d'attributs référencés devrait comporter au moins un identifiant et lors de la transformation en type d'associations, les attributs ne faisant pas partie de l'identifiant seraient éliminés du type d'entités de départ, au même titre que les autres. Toutefois l'application de cette transformation devra se faire avec toute la vigilance

requise. En effet, les attriubts concernés par cette transformation font peut-être également partie d'une autre contrainte, qu'il convient de traiter également.

La possibilité de définir des égalités d'attributs de types d'entités différents via un type d'associations, voire le parcours d'un chemin composé de types d'associations fonctionnels, serait intéressant. En effet, dans l'étude de cas, on retrouve l'attribut *Company code* dans un grand nombre de types d'entité et il fait souvent partie de l'identifiant. A ce moment, il a donc été nécessaire d'exprimer l'égalité d'un tel attribut avec celui d'un autre type d'entités, en suivant parfois des chemins détournés engendrés par le jeu des multiples contraintes de référence. De plus, l'implémentation de l'expression directe de ce type de contrainte contribuerait à la possibilité d'implémenter également la transformation T - 20 que nous avons décrite ci-dessus.

La transformation T - 21 pourrait également faire partie de l'arsenal de l'atelier. Le problème majeur se situe au niveau de la contrainte. Celle-ci peut prendre des formes assez variées et est donc difficile à gérer de façon automatique.

Enfin, lorsqu'on utilise un nombre assez important de sous-types, il arrive qu'un type d'associations soit réparti parmi plusieurs sous-types. Il serait intéressant de permettre à l'utilisateur d'accéder au type d'entités qui se trouve à l'extrémité de tous ces types d'associations. Le cas le présente dans ce mémoire lorsqu'il faut exprimer l'identifiant de « PURCHASE ITEM » : celui-ci est composé de l'en-tête de commande correspondante et du numéro de ligne. Or, l'en-tête de commande est accessible via l'un ou l'autre type d'associations selon le sous-type auquel appartient la ligne de commande. Dès lors, il serait intéressant de pouvoir accéder à l'en-tête de commande via l'union de tous ces types d'associations, comme le suggère la Figure 44. On pourrait discuter sur l'écriture de la contrainte et proposer « id(A) = A1, (R'.B' $\cup$ R''.B'') ».

Toutefois, on peut se demander si cette situation est suffisamment fréquente pour exiger le développement d'une telle représentation, qui n'est certainement pas facile à implémenter...

Cette étape est également très importante au niveau du temps passé. La détraduction et la désoptimisation ont pris ensemble environ 20 jours. Ici également, tout le temps n'a pas été passé à appliquer les transformations : chaque fois que le schéma s'enrichissait d'un nouveau type d'entités (leur nombre a triplé entre le schéma brut et le schéma conceptuel), il fallait lui trouver une place, chaque fois que naissait un type d'associations (c'est -à-dire à 74 reprises), il fallait l'insérer, ainsi que ses rôles, dans un schéma déjà touffu. Le placement des objets a donc pris beaucoup de temps, d'autant plus que chaque déplacement d'objet provoque le réaffichage du schéma, qui prend généralement entre 2 et 3 secondes[10]. Quand on a multiplié par le nombre de déplacements effectués, on arrive à quelques heures.

[10] Ces chiffres sont à remettre dans leur contexte. Le processeur utilisé était un 80486 SX fonctionnant à 33MHz. La carte vidéo, qui joue également un rôle à ce niveau, comportait 500K de mémoire locale. Avec un matériel moderne, ce réaffichage est presque imperceptible à l'oeil nu.

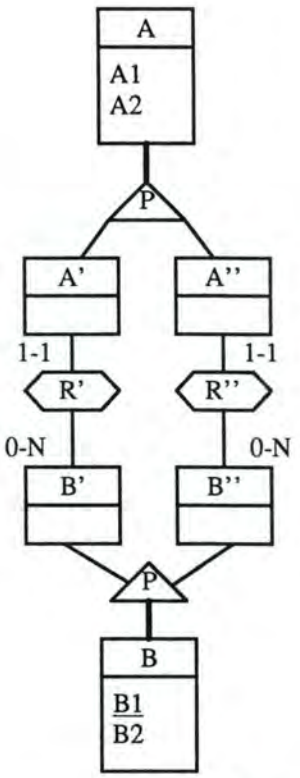


Figure 44 - Expression d'un identifiant comportant plusieurs types d'associations

8.3.5 La normalisation conceptuelle

Dans notre cas, l'étape de normalisation conceptuelle est réduite à néant : en fait, le schéma est tellement complexe qu'on ne voit pas comment produire des types d'associations « many-to-many » ou ternaires... De plus, les différentes dépendances fonctionnelles ont été éliminées lors du traitement des contraintes et donc le schéma est normalisé.

On pourrait se dire que la normalisation d'un schéma est quelque chose de connu et de maîtrisé et que, par conséquent, les transformations relatives à cette étape devraient être implémentées intégralement dans l'atelier. Il n'en est rien !!! En ce qui concerne les attributs décomposables ou multivalué, les types d'associations N-aires ou « many-to-many » et les relations IS-A, tout est prévu. Mais quand il s'agit de s'attarder aux dépendances fonctionelles entre attributs, on ne trouve rien... Ce point a déjà été soulevé au moment de la conceptualisation et il est inutile d'y revenir plus longuement.

8.4 En bref...

Nous reprenons dans le Tableau 3 certains chiffres cités au long de ce chapitre. La première colonne reprend les différentes étapes de la méthode. Pour chacune d'elle, on donne le temps passé pour la réaliser et deux appréciations de l'atelier. La première représente l'aide que l'atelier nous a procuré lors de cette étude de cas et la seconde représente l'aide qu'on peut

espérer de l'atelier si celui-ci était amélioré en fonction des différentes suggestions formulées dans ce chapitre. Ces deux évaluations suivent la règle suivante :

- 0 = l'atelier n'aide en rien lors de cette étape
- 1 = l'atelier apporte une aide lors de cette étape
- 2 = l'atelier apporte une aide réellement appréciable lors de cette étape

Tableau 3 : Quelques chiffres...

Etape concernée	Temps passé	Aide de l'atelier	Aide future de l'atelier
Extraction du schéma brut	7 jours	0	2
Analyse des programmes	53 jours	0	1
Simplification du schéma	½ jour	0	1
Détraduction et désoptimisation	20 jours	1	2
Normalisation conceptuelle	/	/	2

Finalement, on remarque que l'atelier n'est pas encore optimal. L'extraction des schémas bruts s'est faite à la main. Toutefois, si un « parser » RPG existait (et ce ne serait pas compliqué d'en ajouter un dans l'atelier), cette aide deviendrait substantielle et permettrait de gagner un temps précieux. Au niveau de l'analyse des programmes, on se rend facilement compte que l'atelier ne pourra jamais, du moins dans un futur relativement proche, fournir une aide importante : les algorithmes d'analyse de programmes ne sont pas encore suffisamment développés pour retrouver les contraintes des champs d'un ou plusieurs fichiers dans un programme. Les facilités de « pattern matching », que nous n'avons pas utilisées vu qu'elles n'ont été disponibles que lorsque l'analyse des programmes était terminée, fournissent toutefois un support. L'amélioration de l'aide à la simplification du schéma supposerait des facilités de détection de structures redondantes, qui n'existent pas encore au sein de l'atelier. Lors de la détraduction et de la désoptimisation, les transformations de schéma offrent une aide, mais certaines améliorations sont encore possibles, notamment concernant les transformations T - 20 et T - 21. Enfin, la normalisation conceptuelle étant réduite à néant, il n'est pas possible d'évaluer l'atelier concernant ce point. Toutefois, l'atelier pourrait être d'un grand secours si les dépendances fonctionnelles y étaient implémentées. Dans ce cas, cette étape pourrait, de façon extrême, se réduire à un « clic » sur un bouton...

La méthode de rétro-ingénierie décrite à la section 2.2 semble satisfaisante pour des études de cas telles que celle-ci. Elle s'appuye sur des raisonnement naturels et rigoureux, ce qui ne veut pas toujours dire qu'il suffit de se laisser guider sans prendre d'initiative... Quelquefois, il est important de suivre un chemin un peu détourné, comme nous l'avons fait lorsque nous avons utilisé certaines transformations un peu inhabituelles.

L'atelier DB-MAIN supporte plus ou moins bien cette méthode : il a été construit dans ce but. Toutefois, certaines transformations intéressantes ne sont pas reprises, de même que le concept de dépendance fonctionnelle, qui est pourtant abondamment utilisé.

Au niveau du temps passé pour appliquer la méthode, on remarque les deux grands pôles : l'analyse des programmes et la conceptualisation. Le premier demande du temps parce qu'il faut décortiquer un programme, ce qui est rarement chose facile. Le second est long principalement à cause des aspects graphiques : agencer 89 types d'entités et 74 types d'associations n'est pas facile et prend du temps...

Chapitre 9 : Conclusion

Nous venons de réunir, dans ce mémoire, deux mondes séparés : tout d'abord le monde universitaire, qui produit de nombreuses méthodes, méthodologies et autres enseignements, et ensuite le monde des entreprises, qui recherche la performance et le moindre coût.

Le monde universitaire est représenté par ses produits : une méthode de rétro-ingénierie et un atelier logiciel la supportant. La méthode est basée principalement sur une méthode de conception des bases de données. Cette dernière consiste à construire une base de données, à partir des besoins d'un utilisateur. Ces besoins sont tout d'abord travaillés pour en tirer un schéma conceptuel. Celui-ci subit alors des transformations pour devenir un schéma conforme au modèle d'un SGBD. La méthode de rétro-ingénierie suit le processus inverse. Elle part des produits disponibles, comportant le schéma codé, les vues externes, les programmes,... pour en dériver un schéma conceptuel.

Les transformations de schéma jouent un rôle important dans ces deux méthodes, ce pourquoi nous leur avons consacré quelques pages. Enfin, l'atelier logiciel DB-MAIN a pour but de faciliter au maximum la tâche de ceux qui veulent utiliser ces méthodes. Il propose une interface de manipulation d'objets relatifs aux bases de données et permet de leur appliquer diverses transformations de schéma.

Le monde des entreprises nous fournit un cas réel, de taille appréciable, qui n'a rien de méthodologique ou de pédagogique. Celui-ci est truffé de pièges et c'est à la méthode de faire ses preuves pour les déjouer et à l'atelier de faire en sorte que la résolution de ce problème ne soit pas trop coûteuse. Ceci est d'autant plus vrai que cette base de données comporte des caractéristiques peu communes à celles que nous avons l'habitude de rencontrer : le SGBD ne supporte qu'une partie du modèle relationnel, les programmes sont écrits dans le langage RPG, qui n'est pas des plus lisibles,...

La réunion de ces deux mondes a démontré l'efficacité de la méthode de rétro-ingénierie. Toutefois, nous attirons l'attention sur le fait qu'un seul cas a été traité dans ce mémoire. Le manque de temps ne nous permet pas d'en traiter plusieurs, bien qu'il serait intéressant de le faire. Pour tester la méthode de manière efficace, il serait souhaitable d'effectuer le même travail que celui réalisé dans ce mémoire en faisant varier certains paramètres, tels que l'environnement, le SGBD, l'âge de la base de données, sa taille,... C'est en multipliant les angles d'approche que la méthode sera améliorée. Le discours est identique au niveau de l'atelier DB-MAIN : certaines améliorations ont été suggérées dans le dernier chapitre de ce mémoire, mais il y en a certainement d'autres à faire et de nouvelles études de cas pourraient servir également de moteur au perfectionnement de ce logiciel.

Le principal enseignement à tirer de ce mémoire est le fait qu'une méthode, aussi parfaite semble-t-elle, reste une méthode particulière et il existe toujours un cas pour lequel elle n'est pas totalement adaptée. De la même façon, un logiciel possède toujours des lacunes avec lesquelles il faut composer. Celui-ci doit, en outre, évoluer en fonction des changements qui interviennent dans le monde des bases de données.

Bibliographie

ANDERSSON, 1994

Andersson, M., « Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering », in « Proc. of the 13th Int. Conf. on Entity Relationship Approach », Manchester, Springer-Verlag, 1994

BODART, 1989

Bodart, F., Pigneur, Y., « Conception assistée des systèmes d'information - Méthode, Modèles, Outils », Masson, 1989

CHIANG, 1994

Chiang, R., Barron, T., Storey, V., « Reverse Engineering of Relational Databases : Extraction of an EER Model from a relational database », « Journal of Data and Knowledge Engineering », Vol. 2, N° 2 (March 1994), pp 107-142, 1994

DAVIS, 1985

Davis, K., Adarsh, K., « A Methodology for Translating a Conventional File System into an Entity-Relationship Model », in « Proc. of Entity Relationship Approach », October, 1985

ENGLEBERT, 1995

Englebert, V., Henrard, J., Roland, D., Hainaut, J-L, « DB-MAIN : un atelier d'ingénierie de bases de données », Actes des « Journées Bases de Données Avancées », 20/8 - 1/9, 1995

FONKAM, 1992

Fonkam, M., Gray, W., « An approach to Eliciting the Semantics of Relational Databases », in « Proc. of the 4th Int. Conf. on Advance Information Systems Engineering - CAiSE 92 », pp 463-480, May, LNCS, Springer-Verlag, 1992

HAINAUT, 1993a

Hainaut, J-L, Chandelon, M., Tonneau, C., Joris, M., « Contribution to a Theory of Database Reverse Engineering », in « Proc. of the IEEE Work. Conf. on Reverse Engineering », Baltimore, May, 1993

HAINAUT, 1993b

Hainaut, J-L, « Database Reverse Engineerging - A Systematic Approach », Institut d'Informatique, FUNDP, 1993

HAINAUT, 1994

Hainaut, J-L, « The DB-MAIN Database Engineering CASE Tool - Functions Overview », Institut d'Informatique, FUNDP, 1994

HAINAUT, 1995a

Hainaut, J-L, « DB-MAIN Tutorial - Volume 1 : Introduction to database design, Project Manual » (220 p.), « DB-MAIN Research Report », Institut d'Informatique, FUNDP, April, 1995

HAINAUT, 1995b

Hainaut, J-L, Englebert, V., Henrard, J., Hick, J-M, Roland, D., « Requirements for Information Systems Reverse Engineering Support », in « Proc. of the 2nd IEEE Work. Conf. on Reverse Engineering », Toronto, July 1995, IEEE Computer Society Press, 1995

HAINAUT, 1995c

Hainaut, J-L, « De la rétro-ingénierie à la ré-ingénierie d'applications », XIIIème congrès Inforsid, Grenoble, 1995

JORIS, 1992

Joris, M., Van Hoe, R., Hainaut, J-L, Chandelon, M., Tonneau, C., Bodart, F. et al., « PHENIX : methods and tools for database reverse engineering », in « Proc. 5th Int. Conf. on Software Engineering and Applications », Toulouse, 7-11 December, 1992

NILSSON, 1985

Nilsson, E., « The Translation of COBOL Data Structure to an Entity-Rel-type Conceptual Schema », in « Proc. of Entity-Relationship Approach », October, 1985

PETIT, 1994

Petit, J-M, Kouloumdjian, J., Bouliaut, J-F, Toumani, F., « Using Queries to Improve Database Reverse Engineering », in « Proc. of the 13th Int. Conf. on ER Approach », Manchester, Springer-Verlag, 1994

PREMERLANI, 1993

Premerlani, W., Blaha, M., « An Approach for Reverse Engineering of Relational Databases », in « Proc. of the IEEE Work. Conf. on Reverse Engineering », Baltimore, May, 1993

ROCK, 1990

Rock-Evans, R., « Reverse Engineering : Markets, Methods and Tools », OVUM report, 1990

Annexe 1 : Le langage de définition des patterns

Annexe 1 : Le langage de définition des patterns

Nous allons décrire ce langage dans le formalisme BNF. Cette description est incomplète. Elle est présente dans un but d'aide à la compréhension et non pas d'exhaustivité. Ce langage est lui-même une forme dérivée du BNF, comportant des parties variables.

Remarques :

1. Les caractères en gras doivent apparaître tels quels dans la définition du pattern
2. [x] représente le caractère facultatif de x
3. x^4 est équivalent à [x] [x] [x] [x] (ceci est valable pour n'importe quel exposant)
4. x^* est équivalent à x^∞

<nom de pattern> ::= [A-Za-z0-9] [A-Za-z0-9]^{29}
<segment terminal> ::= " une chaine de caractères[1]"
<expression régulière> ::= **/g** " une expression régulière "
<segment de choix> ::= **{** | ... | **}**
<groupe de segments> ::= **(*****)**
<segment répétitif> ::= *
<segment facultatif> ::= **[****]**
<intervalle> ::= **range(**c1-c2[2]**)**
<variable> ::= **@** <nom de pattern>
 ::= <nom de pattern> | <segment terminal> | <expression régulière> | <segment de choix> | <groupe de segments> | <segment répétitif> | <segment facultatif> | <intervalle> | <variable>
<pattern> ::= <nom de pattern> **::=** ***;**

Un pattern est donc une suite de segments, chacun d'eux pouvant prendre une forme particulière. Voici quelques exemples de définitions de patterns.

from ::= "FROM";

FROM

mot ::= /g"[A-Za-z0-9] [A-Za-z0-9@#$*% _-]*";

*Albert, FFr54$$*4F*

lire ::= "READ" @mot;

*READ Albert, READ FFr54$$*4F*

---

[1] On peut insérer des caractères spéciaux dans une chaîne de caractères. Ceux-ci sont : /t (tabulation), /n (nouvelle ligne) et /r (remplissage de ligne).
[2] c1 et c2 sont deux caractères.

Annexe 2 : Le langage RPG

Annexe 2 : Le langage RPG

Nous ne décrirons pas le langage RPG de façon exhaustive, mais nous reprendrons les structures les plus utiles rencontrées au cours de l'analyse des programmes. Cette version simplifiée est présentée dans le formalisme BNF. Des commentaires sont insérés tout au long de sa description. Ceux-ci servent également de description sémantique rudimentaire, qui permet de se faire un idée de l'utilité de chaque construction.

Remarques :

1. b représente le caractère blanc
2. [x] représente le caractère facultatif de x
3. x^4 est équivalent à [x] [x] [x] [x] (ceci est valable pour n'importe quel exposant)
4. x^* est équivalent à x^∞

<chiffre> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<lettre> ::= A | B | ... | Y | Z
<caractère spécial> ::= * | @ | #
<lettre ou caractère spécial> ::= <lettre> | <caractère spécial>
<caractère> ::= <lettre ou caractère spécial> | <chiffre>
<ident_6> ::= <lettre ou caractère spécial> [<caractère>]^5
<ident_8> ::= <lettre ou caractère spécial> [<caractère>]^7
<ident_10> ::= <lettre ou caractère spécial> [<caractère>]^9
<record format> ::= <ident_8>
<nom de programme> ::= <ident_6>
<en-tête de programme> ::= H [<nom de programme>]

Le H se place à la 6ème colonne et le <nom de programme> entre la 75ème et la 80ème colonne.

<nom de fichier> ::= <ident_8>
<type de fichier> ::= I | O | U | C

I = Input (fichier d'entrée), O = Output (fichier de sortie), U = Update (fichier de mise à jour), C = I/O (fichier d'entrée-sortie).

<séquence> ::= b | A | D

b et A représentent l'ordre ascendant pour les enregistrements et D l'ordre descendant.

<format> ::= F | E

F indique que le fichier est décrit dans le programme et E indique qu'il est décrit de façon externe.

<type d'adressage> ::= b | K

b signifie que le fichier est un fichier « relative record number » et K signifie que l'accès se fait par clé (dans le cas d'un fichier décrit de façon externe (<format> = E)).

<localisation> ::= DISK | WORKSTN | PRINTER

DISK indique un fichier se trouvant sur disque, WORKSTN indique un fichier contenant la description d'un écran et PRINTER indique un fichier d'impression (dans ce cas, <type de fichier> = O obligatoirement).

<ancien record format> ::= <record format>
<nouveau record format> ::= <record format>

<renommage> ::= <ancien record format> KRENAME <nouveau record format>

<ancien record format> se trouve entre la 19ème et la 26ème colonne, KRENAME

commence à la 53ème colonne et <nouveau record format> se trouve entre la 60ème et la 67ème colonne.

<spécification de fichier> ::= F <nom de fichier> <type de fichier> <séquence> <format> <type d'adressage> <localisation> [<renommage>]

Le F se place à la 6ème colonne, le <nom de fichier> entre la 7ème et la 14ème colonne, le <type de fichier> à la 15ème colonne, la <séquence> à la 18ème colonne, le <format> à la 19ème colonne, le <type d'adressage> à la 31ème colonne et la <localisation> entre la 40ème et la 46ème colonne. Quant à l'éventuel <renommage>, il se trouve sur la ligne suivante.

<spécification des fichiers> ::= <spécification de fichier>*

Description des fichiers utilisés par le programme.

<nom de l'extension> ::= <ident_6>

<longueur> ::= <chiffre> [<chiffre>]3

Nombre maximal d'éléments dans le tableau.

<longueur des éléments> ::= 1 | 2 | ... | 255 | 256

<longueur décimale> ::= <chiffre>

Nombre de chiffres après la virgule dans le cas de données numériques.

<spécification d'extension> ::= E <nom de l'extension> <longueur> <longueur des éléments> [<longueur décimale>]

Le E se place à la 6ème colonne, le <nom de l'extension> entre la 27ème et la 32ème colonne, la <longueur> entre la 36ème et la 39ème colonne, la <longueur des éléments> entre la 40ème et la 42ème colonne et la <longueur décimale> à la 44ème colonne.

<spécification des extensions> ::= <spécification d'extension>*

Description des extensions (tableaux) utilisés par le programme.

<position initiale> ::= <chiffre> [<chiffre>]3

<position finale> ::= <chiffre> [<chiffre>]3

<nom> ::= <ident_6>

<spécification interne de stucture> ::= I DS (I <position initiale> <position finale> <nom>)*

Le premier I se trouve à la 6ème colonne et DS aux colonnes 19 et 20 d'une ligne, le reste se trouve sur les lignes suivantes. Chaque quadruplet se trouve sur une ligne séparée. Le I se trouve à la 6ème colonne, la <position initiale> entre la 44ème et la 47ème colonne, la <position finale> entre la 48ème et la 51ème colonne et le <nom> entre la 53ème et 58ème colonne.

<nom externe du record format> ::= <record format>

<identification externe de la structure> ::= I <nom externe du record format>

Le I est placé à la 6ème colonne et le <nom externe du record format> entre la 7ème et la 14ème colonne.

<nom externe du champ> ::= <ident_10>

<nom interne du champ> ::= <ident_6>

<identification de champ> ::= I <nom externe du champ> <nom interne du champ>

Le I se trouve à la 6ème colonne, le <nom externe du champ> entre la 21ème et la 30ème colonne et le <nom interne du champ> entre la 53ème et la 58ème colonne.

<identification des champs> ::= <identification de champ>*

<spécification externe de structure> ::= <identification externe de structure> <identification des champs>

<spécification de structure> ::= <spécification interne de structure> | <spécification externe de structure>

<spécification des structures> ::= <spécification de structure>*

Description des structures de données utilisées par le programme.

<nom de clé> ::= <ident_6>

<clé> ::= C <nom de clé> KLIST

Le C se trouve à la 6ème colonne, le <nom de clé> entre la 18ème et la 23ème colonne et KLIST entre la 28ème et la 32ème colonne.

<composante> ::= <ident_6>

<composante de la clé> ::= C KFLD <composante>

Le C se trouve à la 6ème colonne, KFLD entre la 28ème et la 31ème colonne et la <composante> entre la 43ème et la 48ème colonne.

<définition de clé> ::= <clé> [<composante de la clé]*

La <clé> et les <composante de la clé> se trouvent sur des lignes successives.

<nom de paramètre> ::= <ident_6>

<paramètre> ::= C <nom de paramètre> PLIST

Le C se trouve à la 6ème colonne, le <nom de paramètre> entre la 18ème et la 23ème colonne et PLIST entre la 28ème et la 32ème colonne.

<composante de paramètre> ::= C PARM <composante>

Le C se trouve à la 6ème colonne, PARM entre la 28ème et la 31ème colonne et la <composante> entre la 43ème et la 48ème colonne.

<définition de paramètre> ::= <paramètre> [<composante de paramètre>]*

Le <paramètre> et les <composante de paramètre> se trouvent sur des lignes successives.

<en-tête de sous-routine> ::= C <nom de sous-routine> BEGSR

Le C se trouve à la 6ème colonne, le <nom de sous-routine> entre la 18ème et la 23ème colonne et BEGSR entre la 28ème et la 32ème colonne.

<corps de sous-routine> ::= <corps>

Exceptionnellement, le <corps> est défini plus loin

<fin de sous-routine> ::= C ENDSR

Le C se trouve à la 6ème colonne et ENDSR entre la 28ème et la 32ème colonne.

<sous-routine> ::= <en-tête de sous-routine> <corps de sous-routine> <fin de sous-routine>

Les trois parties se trouvent sur des lignes successives

<nom d'étiquette> ::= <ident_6>

<étiquette> ::= C <nom d'étiquette> TAG

Le C se trouve à la 6ème colonne, le <nom d'étiquette> entre la 18ème et la 23ème colonne et TAG entre la 28ème et la 30ème colonne.

<contenu d'un indicateur> ::= *IN <chiffre> <chiffre>

<facteur> ::= <ident_10> | <contenu d'un indicateur>

<premier facteur> ::= <facteur>

<code opératoire> ::= ADD | CHAIN | DIV | EXFMT | LOKUP | MOVE | MOVEA | MOVEL | MULT | READ | READC | READE | READP | REDPE | SETGT | SETLL | SETOF | SETON | SORTA | SQRT | SUB | UPDAT | WRITE | Z-ADD | Z-SUB

ADD = Addition, CHAIN = Recherche dans un fichier, DIV = Division, EXFMT = Affichage d'un écran formatté et lecture de celui-ci, LOKUP = Recherche d'un élément dans un tableau, MOVE = Tranfert, MOVEA (MOVE Array) = Transfert d'un tableau, MOVEL (MOVE Left) = Transfert en insérant à partir de la gauche dans le résultat, MULT = Multiplication, READ = Lecture dans un fichier, READC (READ next Changed record) = Lecture du prochain enregistrement qui a changé, READE (READ Equal key) = Lecture d'un enregistrement dont la clé est égale à celle spécifiée, READP (READ Prior) = Lecture de l'enregistrement précédent, REDPE (REaD Prior or Equal) = Lecture de l'enregistrement précédent ou égal, SETGT (SET Greater Than) = Positionne le pointeur courant d'un fichier sur l'enregistrement immédiatement supérieur à la clé spécifiée, SETLL (SET Lower Limit) = Positionne le pointeur courant d'un fichier sur le plus petit enregistrement, SETOF = Met un indicateur à 0, SETON = Met un indicateur à 1, SORTA (SORT Array) = Trie un

tableau, SQRT (SQuare RooT) = Extraction de la racine carré, SUB = Soustraction, UPDAT = Mise à jour d'un enregistrement d'un fichier, WRITE = Ecriture dans un fichier, Z-ADD (Zero and ADD) = Met à 0 et ajoute, Z-SUB (Zero and SUB) = Met à 0 et soustrait.

<constante> ::= '<ident_8>'

<nombre> ::= <chiffre> [<chiffre>]^{9}

<second facteur> ::= <facteur> | <constante> | <nombre>

<résultat> ::= <facteur>

<indicateur de résultat> ::= <chiffre> <chiffre>

<opération> ::= C [<premier facteur>] <code opératoire> [<second facteur>] [<résultat>] [<indicateur de résultat>]

C se trouve à la 6ème colonne, le <premier facteur> entre la 18ème et la 27ème colonne, le <code opératoire> entre la 28ème et la 32ème colonne, <le second facteur> entre la 33ème et la 42ème colonne, le <résultat> entre la 43ème et la 48ème colonne et l'<indicateur de résultat> entre la 54ème et la 59ème colonne.

<code conditionnel> ::= IFEQ | IFNE | IFGT | IFGE | IFLT | IFLE

IFEQ = IF EQual, IFNE = IF Not Equal, IFGT = IF Greater Than, IFGE = IF Greater or Equal, IFLT = IF Lower Than, ILLE = IF Lower or Equal

<condition> ::= C <premier facteur> <code conditionnel> <second facteur>

C se trouve à la 6ème colonne, le <premier facteur> entre la 18ème et la 27ème colonne, le <code conditionnel> entre la 28ème et la 32ème colonne et <le second facteur> entre la 33ème et la 42ème colonne.

<instruction1> ::= <instruction>

<instruction2> ::= <instruction>

Exceptionnellement, l'<instruction> est définie plus loin

<sinon> ::= C ELSE

Le C se trouve à la 6ème colonne et le ELSE entre la 28ème et la 31ème colonne.

<fin de condition> ::= C END | C ENDIF

Le C se trouve à la 6ème colonne et le END (resp. ENDIF) entre la 28ème et la 30ème (resp. 32ème) colonne.

<instruction conditionnelle> ::= <condition> <instruction1>* [<sinon> <instruction2>*] <fin de condition>

Chaque élément se trouve sur une ligne différente.

<code répétitif> ::= DO | DOUEQ | DOUNE | DOUGT | DOUGE | DOULT | DOULE | DOWEQ | DOWNE | DOWGT | DOWGE | DOWLT | DOWLE

DOUxx = DO Until xx, DOWxx = DO While xx (la signification des valeurs de xx se trouve dans la définition du <code conditionnel>).

<répétition> ::= C <premier facteur> <code répétitif> <second facteur>

Le C se trouve à la 6ème colonne, le <premier facteur> entre la 18ème et la 27ème colonne, le <code répétitif> entre la 28ème et la 32ème colonne et le <second facteur> entre la 33ème et la 42ème colonne.

<fin de répétition> ::= C END | C ENDDO

Le C se trouve à la 6ème colonne et le END (resp. ENDDO) entre la 28ème et la 30ème (resp 32ème) colonne.

<instruction répétitive> ::= <répétition> <instruction>* <fin de répétition>

Chaque élément se trouve sur une ligne différente.

<instruction de saut> ::= C GOTO <nom d'étiquette>

C se trouve à la 6ème colonne, le GOTO entre la 28ème et la 31ème colonne et le <nom d'étiquette> entre la 33ème et la 38ème colonne.

<appel de sous-routine> ::= C EXSR <nom de sous-routine>

C se trouve à la 6ème colonne, le EXSR entre la 28ème et la 31ème colonne et le <nom de sous-routine> entre la 33ème et la 38ème colonne.

<appel> ::= C CALL <nom de programme>

C se trouve à la 6ème colonne, le CALL entre la 28ème et la 31ème colonne et le <nom de programme> entre la 33ème et la 38ème colonne.

<argument> ::= C PARM [<second facteur>] <composante>

C se trouve à la 6ème colonne, le PARM entre la 28ème et la 31ème colonne, le <second facteur> entre la 33ème et la 38ème colonne et la <composante> entre la 43ème et la 48ème colonne.

<appel de programme> ::= <appel> <argument>*

Chaque élément se trouve sur une ligne différente.

<instruction> ::= <opération> | <instruction conditionnelle> | <instruction répétitive> | instruction de saut> | <appel de sous-routine> | <appel de programme>

<ligne de calcul> ::= <définition de clé> | <définition de paramètres> | <sous-routine> | <étiquette> | <instruction>

<corps> ::= <ligne de calcul>*

Chaque élément se trouve sur une ligne différente.

<programme RPG> ::= <en-tête de programme> <spécification des fichiers> <spécification des extensions> <spécification des structures> <corps>

Annexe 3 : le schéma brut

Schema Belgium-Malta/Brut-(text) : physical schema with identifiers from unique index (textual view)

BUYER ANALYST / BUYER description of a buyer/analyst

ZBAC char (2)	buyer analyst code
ZDESCR char (25)	buyer analyst description
ZORTY char (1)	order type
ZAPPRO char (1)	approbation flag (Y/N)

id: ZBAC from unique index

CALENDAR / CALNDR calendar used to verify if the delivery date is valid

LCALDT date (10)	calendar date
LBAXDT date (10)	baxter date
LPROD char (1)	non prod day flag (Y/N)
LBAX char (1)	begin baxter mth flag (Y/N)
LBAXY char (1)	begin baxter year flag (Y/N)
LPLAN char (1)	plan date flag (M,W,B,BLK)
LPCLOS char (1)	period close flag (Y/N)
LCOMNT char (10)	date comment field
LLMDTE date (10)	last maintenance date
LLMTME numeric (6)	last maintenance time field length : 6
LLMUSR char (10)	last maintenance user

id: LCALDT *from unique index* *(if LPROD = 'Y')*

id': LBAXDT, LCALDT *from unique index*

COMPANY / CMPANY company description file

ZCMPNY char (2)	company code
ZNAME char (35)	company name
ZADD1 char (35)	company address line 1
ZADD2 char (35)	company address line 2
ZADD3 char (35)	company address line 3
ZSOLD char (3)	sold-to affiliate code
ZGLAC numeric (14)	general ledger accounting no. field length : 14

ZLADT date (10) last activity date
ZLATM numeric (6) last activity time field length : 6
ZUSER char (10) workstation user I.D.
id: ZCMPNY from unique index

COUNTER / COUNTR counter for the order nr
ZCMPNY char (2) company code
ZTYPE char (2) 'N' = ship note, 'O' = order, 'P' = Pick list, 'D' = ?
ZBEGIN numeric (6) beginning number field length : 6
ZNEXT numeric (6) next available number field length : 6
ZEND numeric (6) ending number field length : 6
ZLAST numeric (6) last period ending number field length : 6
ZDESC char (25) description of ?
ZLADT date (10) last activity date
ZLATM numeric (6) last activity time field length : 6
ZUSER char (10) workstation user ID
id: ZCMPNY, ZTYPE from unique index

CURRENCY / CURCOD checks the validity of a currency
ZCURCOD char (2) currency code code
ZDESC1 char (25) currency code description
ZDESC2 char (10) currency code short descr.
ZCONVF numeric (10,5) conversion factor field length : 6; value <> 0
ZCDATE date (10) date of last file maint
ZCTIME numeric (6) time of last file maint field length : 6
ZCUSER char (10) workstation user I.D.
id: ZCURCOD from unique index

CURRENCY CONVERSION TABLE / T$MA31 currency conversion table
CFILTA char (2) file code = TA
CRECTA char (1) register code
CKEYTA char (17) key file table
DATATA char (50) data file table. Contains the conversion values

DUPDTA date (10)	date of last update
CTISTA char (2)	flag TIS

DEAL QTY/PRICE / POPDQP purchase order price deal qty/price breaks

QPCMNY char (2)	company code
QPPRTL char (2)	part number - label
QPPRTC char (2)	part number - country
QPPART char (10)	part number - basic code
QPPRTS char (2)	part number - suffix
QPCMPR char (15)	compressed part
QPVEND char (7)	vendor number
QPCURR char (2)	currency code
QPQTY numeric (9)	generic qty field field length : 5
QPDECV numeric (1)	vendor decimal control field length : 1
QPPRIC numeric (11)	po vendor price field length : 6
QPDISP numeric (4,2)	discount percent field length : 3
QPAST char (1)	recomm. order
QPSTAT char (1)	A = approved, U = unapproved, D = deact
QPFREF date (10)	from effective date
QPTOEF date (10)	to effective date
QPAPDT date (10)	approved by date
QPAPUS char (10)	approved by user
QPDEDT date (10)	date deactivated
QPDEUS char (10)	deactivated by user
QPDATE date (10)	last maint date
QPTIME numeric (6)	last maint time field length : 6
QPUSER char (10)	last maint user

DELIVERY METHOD / DELMTH description of the delivery methods

ZMTHCD char (2)	delivery method
ZLNGCD char (2)	language code
ZDESC char (10)	description
ZINSU char (1)	insurance required in {'Y', 'N'}

ZLADT date (10) last activity date
ZLATM numeric (6) last activity time field length : 6
ZUSER char (10) workstation user I.D.
id: ZMTHCD, ZLNGCD *from unique index*
id': ZLNGCD, ZMTHCD *from unique index*

DELIVERY TERMS / DELTRM delivery terms (franco,...)

ZTRMCD char (2) delivery terms
ZTRMLG char (2) language code
ZTRMDE char (10) description
ZLADT date (10) last activity date
ZLATM numeric (6) last activity time field length : 6
ZUSER char (10) workstation user I.D.
ZDELCD char (3) new delivery terms
id: ZTRMCD, ZTRMLG *from unique index*
id': ZDELCD, ZTRMLG *from unique index*

DELMETHLEADTIME / PURLED vendor delivery method and lead time. The combination of the two allows to computes the delivery date.

G5STAT char (1) record activity code
G5CMNY char (2) company code
G5VEND char (7) vendor number
G5METH char (2) delivery method
G5LNGC char (2) language code
G5LT char (3) delivery lead time
id: G5VEND, G5METH, G5LNGC *from unique index*

HEADER COMMENT / HDRCOM comments regarding the factory (closed period,...)

QBCMNY char (2) company code
QBSEQ numeric (3) comment sequence number field length : 2
QBPCD numeric (1) print code field length : 1
QBCOM char (50) print comment
QBDATE date (10) last maint date

QBTIME numeric (6) last maint time field length : 6

QBUSER char (10) last maint user

LANGUAGE CODE / LNGCOD description of a language code

ZBCODE char (2) language code

ZBDESC char (25) description

ZBLADT date (10) last activity date

ZBLATM numeric (6) last activity time field length : 6

ZBUSER char (10) last activity user I.D.

id: ZBCODE from unique index

PART CROSS REFERENCE / PMCREF cross reference between the compressed and non-compressed (label-country-basic-suffix) part number

CRCOMP char (2) company code

CRLBL char (2) part number - label

CRCOM char (2) part number - country

CRBAS char (10) part number - basic code

CRSUFX char (2) part number - suffix

CRCMPR char (15) compressed version

id: CRCOMP, CRCMPR from unique index

id': CRCOMP, CRBAS, CRSUFX, CRLBL, CRCOM from unique index

PART MASTER / PARTMS description of the products

AACTCD char (1) record activity code

ACMPNY char (2) company code

APARTL char (2) part number - label

APARTC char (2) part number - country

APART char (10) part number - basic

APARTS char (2) part number - suffix

ADES15 char (15) short part description

AQCSTT char (2) standard qc receipt status

APACK numeric (7) packing factor field length : 4

AVOLUM numeric (9,2) volumefield length : 5

ALOLEV numeric (3)	low level cde field length : 2
APSRUN numeric (5)	prod struc run activity no. field length : 3
APSCNT numeric (3)	no of comps on structure field length : 2
ASECUR char (1)	part security code - U,S,B
ADRAW char (5)	drawing number
AUNMS char (2)	stocking unms
ADESC char (50)	part description
ACOMNT char (20)	part comment field
ATYPE char (1)	type code 1 - pur, 2 - mfg, 3 - ips, 4 - dummy mfg
ACOMM char (5)	commodity code
AACCT char (1)	accounting code for part
AACTNO char (3)	account number for part
AFGACT numeric (14)	f/g inv account no field length : 14
AVALUE char (1)	value class A,B,C,D,E
ACORPV char (1)	corporate value class
AISSUE char (1)	issue code : R= req, S = sir
AOPCD char (1)	order policy code
AOPQT numeric (9)	order policy qty field length : 5
ABAC char (2)	buyer analyst code
AMTCLS char (1)	material class
ASUFIX char (1)	suffix tracking flg
AREGRD char (1)	regrind flg
ABATRK char (1)	batch tracking flg
ADEPT char (3)	department number
AWHSE char (2)	warehouse
ASETCS numeric (3,1)	setup cost field length : 2
AVENDR char (7)	vendor number
AFLLOT char (1)	flow vs. lot code
AWETDR char (1)	wet vs. dry product
APOQTY numeric (9)	purchase order qty field length : 5
APRSTC numeric (11,5)	standard cost in units field length : 6
AEURCS numeric (15)	european unit cost field length : 8
AEURDC numeric (3)	european cost decimal cntrl field length : 2

APRCTC numeric (11,5) current cost in units field length : 6
APLTAN char (2) plant analyst
AUSEUM char (2) usage of u/m
ABUYUM char (2) buying u/m
ASELUM char (2) selling u/m for op
APRUM char (2) previous u/m
AUSCU numeric (15,7) usage to stocking U/M convers field length : 8
ABYCUM numeric (15,7) buying to stocking u/m convers field length: 8; value <> 0
ASLCUM numeric (15,7) selling to stocking u/m convers field length : 8
ASHPWT numeric (7,3) shipping weight field length : 4
AWGHT numeric (7,2) weight / 1000 field length : 4
ACUBI numeric (7,2) cubic inches/piece field length : 4
APFLG char (1) part number activity flag
ANOTE1 char (4) standard comments code
ANOTE2 char (4) standard comments code
AVAT char (1) vat code
ASSF char (1) source of supply flag
ASSSF char (1) secondary source of supply flag
AORIG char (1) origin code
ASPRTL char (2) substitute part - label
ASPRTC char (2) substitute part - cmfg
ASPART char (10) substitute part - basic
ASPRTS char (2) substitue part - suffix
ABLC char (2) basic language code
ALMDAT date (10) date of last file maint
AMTIME numeric (6) time of last file maint field length : 6
ALUSER char (10) last maintained user id
APRDTP char (3) product type
AMINI numeric (9) minimum qty for orders on call field length : 5
ACHAR1 char (50) part description
ACHAR2 char (50) part description
ACHAR3 char (50) part description

id: ACMPNY, APARTL, APARTC, APART, APARTS from unique index

PAYMENT CODES / IPSP03

CMNY03 char (2)	company code
ACTI03 char (1)	generic flag - system wide
PAF203 char (3)	intl affiliate code
DESC03 char (50)	generic description field
PGMN03 char (10)	pgm name
LMDA03 date (10)	date last maintained
LMTM03 numeric (6)	time last maintained field length : 6
LUSR03 char (10)	user ID that last maintained
WRKS03 char (10)	work station
id: CMNY03, PAF203	**if ACTI03 <> 'D' (from unique index)**

PAYMENT TERMS / PAYTRM description of payment term

ZPAYCD char (1)	payments terms
ZDAYS numeric (3)	number of days field length : 2; value <> 0
ZPAYLG char (2)	language code
ZPAYDE char (25)	description
ZLADT date (10)	last activity date
ZLATM numeric (6)	last activity time field length : 6
ZUSER char (10)	workstation user I.D.
id: ZPAYCD, ZPAYLG	*from unique index*

PRICE DEAL HEADER / POPDHD description of contracts with the suppliers

HDSTAT char (1)	status code (A = approved, D = deapproved)
HDCMNY char (2)	company code
HDPRTL char (2)	part number - label
HDPRTC char (2)	part number - country
HDPART char (10)	part number - basic code
HDPRTS char (2)	part number - suffix
HDCMPR char (15)	compressed part
HDVEND char (7)	vendor number
HDRANK char (1)	vendor ranking
HDSLED numeric (3)	supplier lead time field length : 3

HDDLED numeric (3) delivery lead time field length : 3
HDCONT char (10) contract number
HDHGP numeric (3,1) handling charge percent field length : 2
HDDTP numeric (3,1) duties percent field length : 2
HDDATE date (10) generic date - YYMMDD
HDTIME numeric (6) generic time - HHMMSS field length : 6
HDUSER char (10) workstation user I.D.
id: HDCMNY, HDPART, HDPRTS, HDPRTL, HDPRTC, HDVEND from unique index

PURCHASE COMMENT / PURCOM Comments on the order (header = general comments, vendor = message for the supplier, item = comment on one order line)

G8STAT char (1) O = open, C = close, S = suspended, D = deleted
G8PREV char (1) previous status
G8CMNY char (2) company code
G8TYPE char (1) H = header, V = vendor, P = part (= item)
G8BAC char (2) buyer analyst code
G8ORNO char (6) order number
G8LINE numeric (3) order line no. field length : 3
G8REQ char (8) requisition number
G8COMM numeric (2) comment line no. field length : 2
G8COMD char (50) comments text
G8WRT char (1) print (Y/N) ?
G8LADT date (10) generic date - YYMMDD
G8LATM numeric (6) generic time - HHMMSS field length : 6
G8USER char (10) workstation user I.D.

PURCHASE CONTROL FILE / IPSP06 used to check if someone works on a purchase order

CMNY06 char (2) company code
RECE06 char (6) receipt number
BAC06 char (2) buyer analyst
ORD06 char (6) order number
PGMN06 char (10) program name

LMDA06 date (10) date last maintained

LMTM06 numeric (6) time last maintained field length : 6

LUSR06 char (10) user I.D. that last maint

WRKS06 char (10) workstation

PURCHASE ITEM / PURITM item part of a purchase order

G2STAT char (1) O = open, C = closed, N = cancelled, D = deleted

G2PREV char (1) O = open, C = closed, N = cancelled, D = deleted

G2CMNY char (2) company code

G2BAC char (2) buyer analyst code

G2ORNO char (6) order number

G2LINE numeric (3) order line number field length : 3

G2REQ char (8) requisition number

G2TYPE char (1) purchase order type

G2RLIN numeric (3) requisition number field length : 3

G2PRTL char (2) part number - label

G2PRTC char (2) part number - country

G2PART char (10) part number - basic code

G2PRTS char (2) part number - suffix

G2CMPR char (15) compressed part

G2NONS char (50) non-stock part

G2COM char (1) N = none, D = default vendor comment, P = purcom item comment

G2ALOC char (1) allocated part (Y/N)

G2WHSE char (2) warehouse

G2DEPT char (3) department number

G2ITDT date (10) order date

G2IODT date (10) mrp and requisition requirement date

G2DLDT date (10) delivery date

G2DEST date (10) despatch date

G2OORQ numeric (9) order qty in stocking U/M field length : 5

G2IORQ numeric (9) init. order qty in stocking field length : 5

G2OBDU numeric (9) total bal. due in stocking field length : 5

G2PORG char (1)	E = entered price, S = standard cost, P = price deal
G2DORG char (1)	E = entered date, C = calculated date
G2PFLG char (1)	N = not changed, C = changed
G2CFLG char (1)	receipt consumption flag
G2UNMS char (2)	partms stocking unit measure
G2UNMB char (2)	partms buying unit measure
G2UNMP char (2)	U/M used last in P.O. maint
G2QTYR numeric (9)	total qty received in stocking field length : 5
G2DECL numeric (1)	local decimal control field length : 1
G2DECV numeric (1)	vendor decimal control field length : 1
G2COST numeric (15)	vendor stkg U/M price field length : 8
G2COSF numeric (15)	local stkg U/M price field length : 8
G2CSTB numeric (15)	vendor buying U/M price field length : 8
G2CSFB numeric (15)	local buying U/M price field length : 8
G2DCST numeric (15)	vendor curr stkg discount field length : 8
G2DCSF numeric (15)	local curr stkg discount
G2DCTB numeric (15)	vendor curr buying discount field length : 8
G2DCFB numeric (15)	local curr buying discount field length : 8
G2RATE numeric (10,5)	currency rate field length : 6
G2CSQ1 numeric (2)	current sequence no. 1 field length : 2
G2CSQ2 numeric (2)	advice time printed field length : 2
G2CSQ3 numeric (2)	recall time printed field length : 2
G2CDT1 date (10)	MRO transfer date
G2CDT2 date (10)	advice date printed
G2CDT3 date (10)	recall date printed
G2ADVF char (1)	advice type M = advmst, P = advitm
G2ADVS char (1)	advice status Y = yes, N = no, C = complt
G2RECS char (1)	recall status Y/N/C = complt
G2CQT1 numeric (9)	cur order stk qty no. 1 field length : 5
G2CQT2 numeric (9)	cur order stk qty no. 2 field length : 5
G2CQT3 numeric (9)	cur order stk qty no. 3 field length : 5
G2CTR1 numeric (9)	qty on dock in stk um field length : 5
G2CTR2 numeric (9)	qty on dock in buy um field length : 5

G2CTR3 numeric (9)	cur total recd stk no. 3	field length : 5
G2CBD1 numeric (9)	cur stk. bal due no. 1	field length : 5
G2CBD2 numeric (9)	cur stk. bal due no. 2	field length : 5
G2CBD3 numeric (9)	cur stk. bal due no. 3	field length : 5
G2MQTS numeric (9)	matched receipt qty-stock um	field length : 5
G2MQTB numeric (9)	matched receipt qty-buy um	field length : 5
G2OORB numeric (9)	order quantity (buying um)	field length : 5
G2IORB numeric (9)	initial order qty - buying um	field length : 5
G2OBDB numeric (9)	total ball due in buying	field length : 5
G2QTYB numeric (9)	total qty received in buying um	field length : 5
G2CQB1 numeric (9)	cur buying um qty no. 1	field length : 5
G2CQB2 numeric (9)	cur buying um qty no. 2	field length : 5
G2CQB3 numeric (9)	cur buying um qty no. 3	field length : 5
G2CBR1 numeric (9)	cur tot. rcvd buying no. 1	field length : 5
G2CBR2 numeric (9)	cur tot. rcvd buying no. 2	field length : 5
G2CBR3 numeric (9)	cur tot. rcvd buying no. 3	field length : 5
G2BBD1 numeric (9)	cur buying um bal. due no. 1	field length : 5
G2BBD2 numeric (9)	cur buying um bal. due no. 2	field length : 5
G2BBD3 numeric (9)	cur buying um bal. due no. 3	field length : 5
G2LRDT numeric (6)	last receipt date (YYMMDD)	field length : 6
G2REQ1 char (8)	requisition number 1	
G2REQ2 char (8)	requisition number 2	
G2REQ3 char (8)	requisition number 3	
G2CLSD date (10)	close date	
G2ACCT numeric (14)	account number	field length : 14
G2CONT char (10)	contract number	
G2VDIS numeric (4,2)	vendor discount %	field length : 3
G2PDIS numeric (4,2)	part discount %	field length : 3
G2PRIN char (1)	print the price	
G2ORTY char (1)	order type	
G2VATC char (1)	vat code	
G2MROD numeric (5)	MRO request number	field length : 5
G2CPAN numeric (6)	cpa number	field length : 6

G2ASST numeric (6) asset number field length : 6
G2MROF numeric (3) mro familly field length : 2
G2MROG numeric (2) mro sub-grp field length : 2
G2MROS numeric (3) produc sequence field length : 2
G2LADT date (10) last activity date (YYMMDD)
G2LATM numeric (6) last activity time (HHMMSS) field length : 6
G2USER char (10) last activity user I.D.
G2PROG char (5) program I.D.
id: G2CMNY, G2BAC, G2ORNO, G2LINE *from unique index* *G2TYPE = 'P'*

PURCHASE MASTER / PURMST master part of a purchase order

G1STAT char (1) status : o = open, c = close, s = suspended, d = deleted
G1PREV char (1) previous status : o = open, c = close, s = suspended, d = deleted
G1CMNY char (2) company code
G1BAC char (2) buyer analyst code
G1ORNO char (6) order number
G1REQ char (8) requisition number
G1PURT char (1) p = po, r = requisition
G1POOG char (1) G = generated, E = entered
G1ORDT date (10) order date
G1VND# char (7) vendor number
G1OVND char (7) original vendor
G1ORIG char (10) originator
G1PFLG char (1) print flag : N = don't print, C = print changes, P = print all
G1PNUM numeric (3) number times printed field length : 2
G1METH char (2) del method
G1DMDE char (10) del meth desc
G1LNCD char (2) language code
G1DELT char (2) delivery terms
G1DTDE char (10) deliv term description
G1PAYT char (1) payment terms
G1PTDE char (10) payment term description
G1VADD char (2) vendor address ovrd

G1OSEQ numeric (5)	one time vendor seq field length : 3
G1WHSE char (2)	warehouse
G1BILL numeric (6)	bill-to customer nr field length : 4
G1SHIP numeric (3)	ship-to customer nr field length : 2
G1COM char (1)	N = none, D = default vendor comment, P = purcom order
G1CURR char (2)	currency code
G1ODDT date (10)	original due - date
G1DELDT date (10)	delivery date
G1DESDT date (10)	despatch date
G1CLDT date (10)	order close date
G1CFLG char (1)	confirmation req flag
G1CDAY numeric (3)	# of days to follow confirm field length : 3
G1MPOC char (1)	meth of po comm
G1CONT char (10)	contract
G1STDV numeric (11)	std po total value field length : 6
G1VNDV numeric (11)	vendor po total value field length : 6
G1LOCV numeric (11)	local po total value field length : 6
G1STDC char (3)	standard comment
G1ORTY char (1)	order type (M = MRO, C = CODIFIED, N = NON CODIFIED)
G1CPA char (1)	cpa order (Y/N)
G1APPL char (1)	
G1LADT date (10)	last activity date
G1LATM numeric (6)	last activity time (HHMMSS) field length : 6
G1USER char (10)	last activity user I.D.
G1PROG char (5)	program I.D.

id: G1CMNY, G1BAC, G1ORNO ***from unique index*** ***G1PURT = 'P'***

STANDARD COMMENTS / IPSP02 standard comments

CMNY02 char (2)	company code
ACTI02 char (1)	generic flag - system wide
CODE02 char (3)	standard code
SHOR02 char (15)	short description
COMM02 char (50)	description

LINE02 numeric (3)	comment line number field length : 2
PGMN02 char (10)	program name
LMDA02 date (10)	date last maintained
LMTM02 numeric (6)	time last maintained field length : 6
LUSR02 char (10)	user I.D. that last maintained
WRKS02 char (10)	workstation

STATUS CODE / IPSP08

ACTI08 char (1)	activity code
STAT08 char (1)	status of the supplier
DESC08 char (20)	status description
AUDI08 numeric (3)	audit delay in months field length : 2
PGMN08 char (6)	program name
USID08 char (10)	user ID
WSTN08 char (10)	workstation
MDAT08 date (10)	last maintenance date
MTIM08 numeric (6)	last maintenance time field length : 4

id: STAT08 from unique index ACTI08 <> 'D'

SUPPLIER APPROBATION / IPSP01 tests a supplier regarding a product

PLAB01 char (2)	product label -dla-
PCOM01 char (2)	product country of manuf -dcm-
PBAS01 char (10)	basic part of the product code
PSUF01 char (2)	product suffix
SUPL01 char (7)	supplier no.
ADDR01 char (2)	address code
STAT01 char (1)	A = approved, E = evaluation
DATA01 date (10)	date of supplier approbation
DATE01 date (10)	date of last supplier evaluation
DATC01 date (10)	date of certification
DATI01 date (10)	date of integration
DACC01 date (10)	date of acceptation
SIT101 char (80)	site of production / not used

PDES01 char (50) product description
VNAM01 char (35) vendor name
VREF01 char (80) vendor product reference
USER01 char (10) user I.D.
MDAT01 date (10) mnce date
TIME01 numeric (6) time field length : 4
WSTN01 char (10) workstation
PGMN01 char (6) program name
id: PLAB01, PCOM01, PBAS01, PSUF01, SUPL01, ADDR01, STAT01 from unique index
id': SUPL01, ADDR01, STAT01, PLAB01, PCOM01, PBAS01, PSUF01 from unique index

UNIT OF MEASURE / UNITM description of a unit of measure
ZUNMS char (2) unit of measure
ZDESC char (25) unit of measure description
id: ZUNMS from unique index

VAT CONTROL / VATCTL VAT control file
ZCCODE char (1) VAT code
ZCPERC numeric (5,2) percentage field length : 3
ZCLADT date (10) last activity date
ZCLATM numeric (6) last activity time field length : 6
ZCUSER char (10) last activity user I.D.
id: ZCCODE from unique index

VENDOR ADDRESS / PURADD contains the different addresses of a supplier
G4CMNY char (2) company code
G4VEND char (7) vendor number
G4CODE char (2) address code
G4SEQ numeric (5) one time vendor seq field length : 3
G4ADD1 char (30) generic address field
G4ADD2 char (30) generic address field

G4ADD3 char (30) generic address field
G4ADD4 char (30) generic address field
G4ADD5 char (30) generic address field
G4CONT char (20) customer contract
G4MPOC char (1) method of po comm
G4VOLT numeric (3) vendor override lead time field length : 2
G4PCOD char (2) phone country code
G4PHAR char (5) phone area code
G4PHON char (10) phone number
G4TEAR char (5) telex area code
G4TELX char (8) telex number
G4FCOD char (2) fax country code
G4FXAR char (5) fax area code
G4FAX char (10) fax number
G4LADT date (10) generic date
G4LATM numeric (6) generic time HHMMSS field length : 6
G4USER char (10) workstation user I.D.
G4VNDN char (35) vendor name
id: G4CMNY, G4VEND, G4CODE, G4SEQ from unique index

VENDOR COMMENT / VENCOM comments for the vendor
QASTAT char (1) record activity code
QACMNY char (2) company code
QAVEND char (7) vendor number
QASEQ numeric (3) vendor sequence number field length : 2
QAPCD numeric (1) print code field length : 1
QACOM char (50) print comment
QADATE date (10) last maint date
QATIME numeric (6) last maint time field length : 6
QAUSER char (10) last maint user

VENDOR IDENTIFICATION / PURVEN contains the description of the suppliers

G3STAT char (1)	vendor status code, A/O/D
G3CMNY char (2)	company code
G3VND# char (7)	vendor number
G3TYPE char (2)	vendor type - grouping flag
G3VNDN char (35)	vendor name
G3ADD1 char (30)	vendor address line 1
G3ADD2 char (30)	vendor address line 2
G3ADD3 char (30)	vendor address line 3
G3ADD4 char (30)	vendor address line 4
G3ADD5 char (30)	vendor address line 5
G3PAY char (1)	vendor payment flag Y/N
G3ATN char (20)	name of contact at vendor
G3CUR char (2)	currency code code
G3PAYT char (1)	payment terms
G3DLMT char (2)	delivery method
G3SPLT numeric (3)	vendor lead time field length : 2
G3LGCD char (2)	language code
G3DELT char (2)	delivery method terms
G3MPOC char (1)	method of PO comm
G3PCOD char (2)	phone country code
G3PHAR char (5)	phone area code
G3PHON char (10)	phone number
G3TEAR char (5)	telex area code
G3TELX char (8)	telex number
G3FCOD char (2)	fax country code
G3FXAR char (5)	fax area code
G3FAX char (10)	fax number
G3CORP char (1)	corp flag
G3CONT char (1)	contract flag
G3WHSE char (2)	primary warehouse
G3DISC numeric (4,2)	discount field length : 3
G3CDAY numeric (3)	# of days to follow confirm field length : 2

G3BANK numeric (12) bank number field length : 12
G3VAT char (16) vat number
G3CVAT char (1) vat code Y/N
G3ADR1 char (50) foreign address line 01
G3ADR2 char (50) foreign address line 02
G3ADR3 char (50) foreign address line 03
G3BKF char (34) foreign bank number
G3IBLC char (3) iblc code
G3PAFF char (1) payment code part one
G3PAF2 char (3) payment code part two
G3UPDE char (1) updated by cgas Y/*BL
G3EURC char (3) european coutry code
G3LADT date (10) last activity date
G3LATM numeric (6) last activity time field length : 6
G3USER char (10) last activity user I.D.
id: G3CMNY, G3VND# *from unique index*

VENDOR/PART COMMENTS / PRTCOM comments on a product of a supplier

QCSTAT char (1) status code (A = approved, D = deapproved)
QCCMNY char (2) company code
QCPRTL char (2) part number - label
QCPRTC char (2) part number - country
QCPART char (10) part number - basic code
QCPRTS char (2) part number - suffix
QCCMPR char (15) compressed part
QCVEND char (7) vendor number
QCSEQ numeric (3) comment sequence number field length : 2
QCPCD numeric (1) print code field length : 1
QCCOM char (50) print comment
QCDATE date (10) last maint date
QCTIME numeric (6) last maint time field length : 6
QCUSER char (10) last maint user
id: QCCMNY, QCPART, QCPRTS, QCPRTL, QCPRTC, QCVEND, QCSEQ from unique index

id': QCCMNY, QCPART, QCPRTS, QCPRTL, QCPRTC, QCVEND, QCSEQ from unique index QCSTAT <> 'D'

WAREHOUSE / WHOUSE warehouse

Z1CMNY char (2)	company code
Z1WHSE char (2)	warehouse
Z1DESC char (35)	warehouse description
Z1ADD1 char (35)	warehouse address 1
Z1ADD2 char (35)	warehouse address 2
Z1ADD3 char (35)	warehouse address 3
Z1CSDE char (1)	consigned inventory
Z1INTP char (3)	in-transit inventory type
Z1DEPT char (3)	default department for QC moves
Z1RSDY numeric (3)	number of days ahead to set requested ship date
	field length : 2
ZLADT date (10)	last activity date
ZLATM numeric (6)	last activity time
ZUSER char (10)	last activity user
id: Z1CMNY, Z1WHSE	*from unique index*

PURCHASE ITEM
G2STAT
G2PREV
G2CMNY
G2BAC
G2ORNQ
G2LINE
G2REQ
G2TYPE
G2RLIN
G2PRTL
G2PRTC
G2PART
G2PRTS
G2CMPR
G2NONS
G2COM
G2ALOC
G2WHSE
G2DEPT
G2ITDT
G2IODT
G2DLDT
G2DEST
G2OORQ
G2IORQ
G2OBDU
G2PORG
G2DORG
G2PFLG
G2CFLG
G2UNMS
G2UNMB
G2UNMP
G2QTYR
G2DECL
G2DECV
G2COST
G2COSF
G2CSTB
G2CSFB
G2DCST
G2DCSF
G2DCTB
G2DCFB
G2RATE
G2CSQ1
G2CSQ2
G2CSQ3
G2CDT1
G2CDT2
G2CDT3
G2ADVF
G2ADVS
G2RECS
G2CQT1
G2CQT2
G2CQT3
G2CTR1
G2CTR2
G2CTR3
G2CBD1
G2CBD2
G2CBD3
G2MQTS
G2MQTB
G2OORB
G2IORB
G2OBDB
G2QTYB
G2CQB1
G2CQB2
G2CQB3
G2CBR1
G2CBR2
G2CBR3
G2BBD1
G2BBD2
G2BBD3
G2LRDT
G2REQ1
G2REQ2
G2REQ3
G2CLSD
G2ACCT
G2CONT
G2VDIS
G2PDIS
G2PRIN
G2ORTY
G2VATC
G2MROD
G2CPAN
G2ASST
G2MROF
G2MROG
G2MROS
G2LADT
G2LATM
G2USER
G2PROG
id: G2CMNY
G2BAC
G2ORNO
G2LINE

PART MASTER
AACTCD
ACMPNY
APARTL
APARTC
APART
APARTS
ADES15
AQCSTT
APACK
AVOLUM
ALOLEV
APSRUN
APSCNT
ASECUR
ADRAW
AUNMS
ADESC
ACOMNT
ATYPE
ACOMM
AACCT
AACTNC
AFGACT
AVALUE
ACORPV
AISSUE
AOPCD
AOPQT
ABAC
AMTCLS
ASUFIX
AREGRD
ABATRE
ADEPT
AWHSE
ASETCS
AVENDR
AFLLOT
AWETDS
APOQTY
APRSTC
AEURCS
AEURDC
APRCTC
APLTAN
AUSEUM
ABUYUM
ASELUM
APRUM
AUSCU
ABYCUM
ASLCUM
ASHPWT
AWGHT
ACUBI
APFLG
ANOTE1
ANOTE2
AVAT
ASSF
ASSSF
AORIG
ASPRTL
ASPRTC
ASPART
ASPRTS
ABLC
ALMDAT
AMTIME
ALUSER
APRDTP
AMINI
ACHAR1
ACHAR2
ACHAR3
id: ACMPNY
APARTL
APARTC
APART
APARTS

PURCHASE MASTER
G1STAT
G1PREV
G1CMNY
G1BAC
G1ORNO
G1REQ
G1PURT
G1POOG
G1ORDT
G1VND#
G1OVND
G1ORIG
G1PFLG
G1PNUM
G1METH
G1DMDE
G1LNCD
G1DELT
G1DTDE
G1PAYT
G1PTDE
G1VADD
G1OSEQ
G1WHSE
G1BILL
G1SHIP
G1COM
G1CURR
G1ODDT
G1DELDT
G1DESDT
G1CLDT
G1CFLG
G1CDAY
G1MPOC
G1CONT
G1STDV
G1VNDV
G1LOCV
G1STDC
G1ORTY
G1CPA
G1APPL
G1LADT
G1LATM
G1USER
G1PROG
id: G1CMNY
G1BAC
G1ORNO

SUPPLIER APPROBATION
PLAB01
PCOM01
PBAS01
PSUF01
SUPL01
ADDR01
STAT01
DATA01
DATE01
DATC01
DATI01
DACC01
SIT101
PDES01
VNAM01
VREF01
USER01
MDAT01
TIME01
WSTN01
PGMN01
id: PLAB01
PCOM01
PBAS01
PSUF01
SUPL01
ADDR01
STAT01
id': SUPL01
ADDR01
STAT01
PLAB01
PCOM01
PBAS01
PSUF01

PRICE DEAL HEADER
HDSTAT
HDCMNY
HDPRTL
HDPRTC
HDPART
HDPRTS
HDCMPR
HDVEND
HDRANK
HDSLED
HDDLED
HDCONT
HDHGP
HDDTP
HDDATE
HDTIME
HDUSER
id: HDCMNY
HDPART
HDPRTS
HDPRTL
HDPRTC
HDVEND

HEADER COMMENT
QBCMNY
QBSEQ
QBPCD
QBCOM
QBDATE
QBTIME
QBUSER

Belgium-Malta/Brut-(graph)

LANGUAGE CODE
ZBCODE
ZBDESC
ZBLADT
ZBLATM
ZBUSER
id: ZBCODE

CURRENCY
ZCURCOD
ZDESC1
ZDESC2
ZCONVF
ZCDATE
ZCTIME
ZCUSER
id: ZCURCOD

PART CROSS REFERENCE
CRCOMP
CRLBL
CRCOM
CRBAS
CRSUFX
CRCMPR
id: CRCOMP
CRCMPR
id': CRCOMP
CRBAS
CRSUFX
CRLBL
CRCOM

CURRENCY CONVERSION TABLE
CFILTA
CRECTA
CKEYTA
DATATA
DUPDTA
CTISTA

CALENDAR
LCALDT
LBAXDT
LPROD
LBAX
LBAXY
LPLAN
LPCLOS
LCOMNT
LLMDTE
LLMTME
LLMUSR
id: LCALDT
id': LBAXDT
LCALDT

PAYMENT TERMS
ZPAYCD
ZDAYS
ZPAYLG
ZPAYDE
ZLADT
ZLATM
ZUSER
id: ZPAYCD
ZPAYLG

STANDARD COMMENTS
CMNY02
ACTI02
CODE02
SHOR02
COMM02
LINE02
PGMN02
LMDA02
LMTM02
LUSR02
WRKS02

VENDOR/PART COMMENTS
QCSTAT
QCCMNY
QCPRTL
QCPRTC
QCPART
QCPRTS
QCCMPR
QCVEND
QCSEQ
QCPCD
QCCOM
QCDATE
QCTIME
QCUSER
id: QCCMNY
QCPART
QCPRTS
QCPRTL
QCPRTC
QCVEND
QCSEQ
id': QCCMNY
QCPART
QCPRTS
QCPRTL
QCPRTC
QCVEND
QCSEQ

BUYER ANALYST
ZBAC
ZDESCR
ZORTY
ZAPPRO
id: ZBAC

COMPANY
ZCMPNY
ZNAME
ZADD1
ZADD2
ZADD3
ZSOLD
ZGLAC
ZLADT
ZLATM
ZUSER
id: ZCMPNY

VENDOR COMMENT
QASTAT
QACMNY
QAVEND
QASEQ
QAPCD
QACOM
QADATE
QATIME
QAUSER

VENDOR IDENTIFICATION
G3STAT
G3CMNY
G3VND#
G3TYPE
G3VNDN
G3ADD1
G3ADD2
G3ADD3
G3ADD4
G3ADD5
G3PAY
G3ATN
G3CUR
G3PAYT
G3DLMT
G3SPLT
G3LGCD
G3DELT
G3MPOC
G3PCOD
G3PHAR
G3PHON
G3TEAR
G3TELX
G3FCOD
G3FXAR
G3FAX
G3CORP
G3CONT
G3WHSE
G3DISC
G3CDAY
G3BANK
G3VAT
G3CVAT
G3ADR1
G3ADR2
G3ADR3
G3BKF
G3IBLC
G3PAFF
G3PAF2
G3UPDE
G3EURC
G3LADT
G3LATM
G3USER
id: G3CMNY
G3VND#

DEAL QTY/PRICE
QPCMNY
QPPRTL
QPPRTC
QPPART
QPPRTS
QPCMPR
QPVEND
QPCURR
QPQTY
QPDECV
QPPRIC
QPDISP
QPAST
QPSTAT
QPFREF
QPTOEF
QPAPDT
QPAPUS
QPDEDT
QPDEUS
QPDATE
QPTIME
QPUSER

PURCHASE COMMENT
G8STAT
G8PREV
G8CMNY
G8TYPE
G8BAC
G8ORNO
G8LINE
G8REQ
G8COMM
G8COMD
G8WRT
G8LADT
G8LATM
G8USER

PAYMENT CODES
CMNY03
ACTI03
PAF203
DESC03
PGMN03
LMDA03
LMTM03
LUSR03
WRKS03
id: CMNY03
PAF203

WAREHOUSE
Z1CMNY
Z1WHSE
Z1DESC
Z1ADD1
Z1ADD2
Z1ADD3
Z1CSDE
Z1INTP
Z1DEPT
Z1RSDY
ZLADT
ZLATM
ZUSER
id: Z1CMNY
Z1WHSE

VENDOR ADDRESS
G4CMNY
G4VEND
G4CODE
G4SEQ
G4ADD1
G4ADD2
G4ADD3
G4ADD4
G4ADD5
G4CONT
G4MPOC
G4VOLT
G4PCOD
G4PHAR
G4PHON
G4TEAR
G4TELX
G4FCOD
G4FXAR
G4FAX
G4LADT
G4LATM
G4USER
G4VNDN
id: G4CMNY
G4VEND
G4CODE
G4SEQ

DELIVERY METHOD
ZMTHCD
ZLNGCD
ZDESC
ZINSU
ZLADT
ZLATM
ZUSER
id: ZMTHCD
ZLNGCD
id': ZLNGCD
ZMTHCD

VAT CONTROL
ZCCODE
ZCPERC
ZCLADT
ZCLATM
ZCUSER
id: ZCCODE

DELIVERY TERMS
ZTRMCD
ZTRMLG
ZTRMDE
ZLADT
ZLATM
ZUSER
ZDELCD
id: ZTRMCD
ZTRMLG
id': ZDELCD
ZTRMLG

PURCHASE CONTROL FILE
CMNY06
RECE06
BAC06
ORD06
PGMN06
LMDA06
LMTM06
LUSR06
WRKS06

STATUS CODE
ACTI08
STAT08
DESC08
AUDI08
PGMN08
USID08
WSTN08
MDAT08
MTIM08
id: STAT08

COUNTER
ZCMPNY
ZTYPE
ZBEGIN
ZNEXT
ZEND
ZLAST
ZDESC
ZLADT
ZLATM
ZUSER
id: ZCMPNY
ZTYPE

DELMETHLEADTIME
G5STAT
G5CMNY
G5VEND
G5METH
G5LNGC
G5LT
id: G5VEND
G5METH
G5LNGC

UNIT OF MEASURE
ZUNMS
ZDESC
id: ZUNMS

Annexe 4 : le schéma enrichi

Schema Belgium-Malta/Enriched-(text) : logical schema

BUYER ANALYST / BUYER description of a buyer/analyst

ZBAC char (2) buyer analyst code

ZDESCR char (25) buyer analyst description

ZORTY char (1) order type in {'M4, 'C', 'N'}

ZAPPRO char (1) approbation flag (Y/N) in {'Y', 'N'}

id: ZBAC from unique index; from program

CALENDAR / CALNDR calendar used to verify if the delivery date is valid

LCALDT date (10) calendar date

LBAXDT date (10) baxter date

LPROD[0-1] char (1) non prod day flag (Y/N)

LBAX[0-1] char (1) begin baxter mth flag (Y/N)

LBAXY[0-1] char (1) begin baxter year flag (Y/N)

LPLAN[0-1] char (1) plan date flag (M,W,B,BLK)

LPCLOS[0-1] char (1) period close flag (Y/N)

LCOMNT[0-1] char (10) date comment field

LLMDTE date (10) last maintenance date

LLMTME numeric (6) last maintenance time field length : 6

LLMUSR char (10) last maintenance user

id: LCALDT if LRPOD = 'Y' (from unique index; from program)

id': LBAXDT, LCALDT from unique index

group: LBAXY, LBAX LBAXY = 'Y' => LBAX = 'Y'

COMPANY / CMPANY company description file

ZCMPNY char (2) company code

ZNAME char (35) company name

ZADD1[0-1] char (35) company address line 1

ZADD2[0-1] char (35) company address line 2

ZADD3[0-1] char (35) company address line 3

ZSOLD[0-1] char (3) sold-to affiliate code

ZGLAC numeric (14) general ledger accounting no. field length : 14

ZLADT date (10) last activity date

ZLATM numeric (6) last activity time field length : 6

ZUSER char (10) workstation user I.D.

id: ZCMPNY from unique index; from program

COUNTER / COUNTR counter for the order nr

ZCMPNY char (2) company code

ZTYPE char (2) 'N' = ship note, 'O' = order, 'P' = Pick list, 'D' = ?

ZBEGIN numeric (6) beginning number field length : 6

ZNEXT numeric (6) next available number field length : 6

ZEND numeric (6) ending numberfield length : 6

ZLAST numeric (6) last period ending number field length : 6

ZDESC char (25) description of ?

ZLADT date (10) last activity date

ZLATM numeric (6) last activity time field length : 6

ZUSER char (10) workstation user ID

id: ZCMPNY, ZTYPE from unique index; from program

ref : ZCMPNY -> COMPANY.ZCMPNY

group: ZBEGIN, ZEND ZEND > ZBEGIN

CURRENCY / CURCOD checks the validity of a currency

ZCURCOD char (2) currency code code

ZDESC1 char (25) currency code description

ZDESC2 char (10) currency code short descr.

ZCONVF numeric (10,5) conversion factor field length : 6; valeur <> 0

ZCDATE date (10) date of last file maint

ZCTIME numeric (6) time of last file maint field length : 6

ZCUSER char (10) workstation user I.D.

id: ZCURCOD from unique index; from program

CURRENCY CONVERSION TABLE / T$MA31 currency conversion table

subtype (P) KEY2, KEY1

CFILTA char (2) file code = TA

CRECTA char (1) register code

DUPDTA date (10) date of last update

CTISTA char (2) flag TIS

DATA1 data file table. Contains the conversion values

is-a KEY1

is-a KEY2

? char (10)

DESC char (40)

DATA2 data file table. Contains the conversion values

is-a KEY1

is-a KEY2

T$REF char (20)

?1 char (2)

T$CONT char (9)

T$CONS char (9)

T$CONB char (9)

?2 char (1)

DEAL QTY/PRICE / POPDQP purchase order price deal qty/price breaks

QPCMNY char (2)	company code	
QPPRTL char (2)	part number - label	
QPPRTC char (2)	part number - country	
QPPART char (10)	part number - basic code	
QPPRTS char (2)	part number - suffix	
QPCMPR char (15)	compressed part	
QPVEND char (7)	vendor number	
QPCURR char (2)	currency code	
QPQTY numeric (9)	generic qty field	field length : 5

QPDECV numeric (1)	vendor decimal control	field length : 1
QPPRIC numeric (11)	po vendor price	field length : 6
QPDISP numeric (4,2)	discount percent	field length : 3
QPAST char (1)	recomm. order	
QPSTAT char (1)	A = approved, U = unapproved, D = deact	
QPFREF date (10)	from effective date	
QPTOEF date (10)	to effective date	
QPAPDT date (10)	approved by date	
QPAPUS char (10)	approved by user	
QPDEDT date (10)	date deactivated	
QPDEUS char (10)	deactivated by user	
QPDATE date (10)	last maint date	
QPTIME numeric (6)	last maint time field length : 6	
QPUSER char (10)	last maint user	

DELIVERY METHOD / DELMTH description of the delivery methods

ZMTHCD char (2)	delivery method	
ZLNGCD char (2)	language code; value = '**' =>	
		no reference to LANGUAGE CODE
ZDESC char (10)	description	
ZINSU char (1)	insurance required	in {'Y', 'N'}
ZLADT date (10)	last activity date	
ZLATM numeric (6)	last activity time	field length : 6
ZUSER char (10)	workstation user I.D.	
id: ZMTHCD, ZLNGCD	*from unique index; from program*	
id': ZLNGCD, ZMTHCD	*from unique index*	
ref : ZLNGCD -> LANGUAGE CODE.ZBCODE		

DELIVERY TERMS / DELTRM delivery terms (franco,...)

ZTRMCD char (2)	delivery terms	
ZTRMLG char (2)	language code; value = '**' =>	
		no reference to LANGUAGE CODE
ZTRMDE char (10)	description	

ZLADT date (10) last activity date

ZLATM numeric (6) last activity time field length : 6

ZUSER char (10) workstation user I.D.

ZDELCD char (3) new delivery terms

id: ZTRMCD, ZTRMLG from unique index; from program

id': ZDELCD, ZTRMLG from unique index

ref : ZTRMLG -> LANGUAGE CODE.ZBCODE

DELMETHLEADTIME / PURLED vendor delivery method and lead time. The combination of the two allows to computes the delivery date.

G5STAT char (1) record activity code

G5CMNY char (2) company code

G5VEND char (7) vendor number

G5METH char (2) delivery method

G5LNGC char (2) language code

G5LT char (3) delivery lead time

id: G5VEND, G5METH, G5LNGC from unique index

HEADER COMMENT / HDRCOM comments regarding the factory (closed period,...)

QBCMNY char (2) company code

QBSEQ numeric (3) comment sequence number field length : 2

QBPCD numeric (1) print code field length : 1

QBCOM char (50) print comment

QBDATE date (10) last maint date

QBTIME numeric (6) last maint time field length : 6

QBUSER char (10) last maint user

KEY1 key file table

is-a CURRENCY CONVERSION TABLE

subtype (P) DATA2, DATA1

T$ZTAB char (3)

T$CUR char (2)

T$YY char (2)

T$MM char (2)

? char (8)

KEY2 key file table

is-a CURRENCY CONVERSION TABLE

subtype (P) DATA2, DATA1

T$ZTAB char (3)

T$KEY char (3)

? char (11)

LANGUAGE CODE / LNGCOD description of a language code

ZBCODE char (2)	language code	
ZBDESC char (25)	description	
ZBLADT date (10)	last activity date	
ZBLATM numeric (6)	last activity time	field length : 6
ZBUSER char (10)	last activity user I.D.	

id: ZBCODE *from unique index; from program*

PART CROSS REFERENCE / PMCREF cross reference between the compressed and non-compressed (label-country-basic-suffix) part number

CRCOMP char (2)	company code
CRLBL char (2)	part number - label
CRCOM char (2)	part number - country
CRBAS char (10)	part number - basic code
CRSUFX char (2)	part number - suffix
CRCMPR char (15)	compressed version

id: CRCOMP, CRCMPR *from unique index*

id': CRCOMP, CRBAS, CRSUFX, CRLBL, CRCOM *from unique index*

PART MASTER / PARTMS description of the products

AACTCD char (1)	record activity code
ACMPNY char (2)	company code
APARTL char (2)	part number - label

APARTC char (2)	part number - country
APART char (10)	part number - basic
APARTS char (2)	part number - suffix
ADES15 char (15)	short part description
AQCSTT[0-1] char (2)	standard qc receipt status
APACK[0-1] numeric (7)	packing factor field length : 4
AVOLUM[0-1] numeric (9,2)	volumefield length : 5
ALOLEV[0-1] numeric (3)	low level cde field length : 2
APSRUN[0-1] numeric (5)	prod struc run activity no. field length : 3
APSCNT[0-1] numeric (3)	no of comps on structure field length : 2
ASECUR[0-1] char (1)	part security code - U,S,B
ADRAW[0-1] char (5)	drawing number
AUNMS[0-1] char (2)	stocking unms
ADESC[0-1] char (50)	part description
ACOMNT[0-1] char (20)	part comment field
ATYPE[0-1] char (1)	type code 1 - pur, 2 - mfg, 3 - ips, 4 - dummy mfg
ACOMM[0-1] char (5)	commodity code
AACCT[0-1] char (1)	accounting code for part
AACTNO[0-1] char (3)	account number for part
AFGACT[0-1] numeric (14)	f/g inv account no field length : 14
AVALUE[0-1] char (1)	value class A,B,C,D,E
ACORPV[0-1] char (1)	corporate value class
AISSUE[0-1] char (1)	issue code : R= req, S = sir
AOPCD[0-1] char (1)	order policy code
AOPQT[0-1] numeric (9)	order policy qty field length : 5
ABAC[0-1] char (2)	buyer analyst code
AMTCLS[0-1] char (1)	material class
ASUFIX[0-1] char (1)	suffix tracking flg
AREGRD[0-1] char (1)	regrind flg
ABATRK[0-1] char (1)	batch tracking flg
ADEPT[0-1] char (3)	department number
AWHSE[0-1] char (2)	warehouse
ASETCS[0-1] numeric (3,1)	setup cost field length : 2

AVENDR[0-1] char (7)	vendor number
AFLLOT[0-1] char (1)	flow vs. lot code
AWETDR[0-1] char (1)	wet vs. dry product
APOQTY[0-1] numeric (9)	purchase order qty field length : 5
APRSTC[0-1] numeric (11,5)	standard cost in units field length : 6
AEURCS[0-1] numeric (15)	european unit cost field length : 8
AEURDC[0-1] numeric (3)	european cost decimal cntrl field length : 2
APRCTC[0-1] numeric (11,5)	current cost in units field length : 6
APLTAN[0-1] char (2)	plant analyst
AUSEUM[0-1] char (2)	usage of u/m
ABUYUM[0-1] char (2)	buying u/m
ASELUM[0-1] char (2)	selling u/m for op
APRUM[0-1] char (2)	previous u/m
AUSCU[0-1] numeric (15,7)	usage to stocking U/M convers field length : 8
ABYCUM[0-1] numeric (15,7)	buying to stocking u/m convers field length: 8; value <> 0
ASLCUM[0-1] numeric (15,7)	selling to stocking u/m convers field length : 8
ASHPWT[0-1] numeric (7,3)	shipping weight field length : 4
AWGHT[0-1] numeric (7,2)	weight / 1000 field length : 4
ACUBI[0-1] numeric (7,2)	cubic inches/piece field length : 4
APFLG[0-1] char (1)	part number activity flag
ANOTE1[0-1] char (4)	standard comments code
ANOTE2[0-1] char (4)	standard comments code
AVAT[0-1] char (1)	vat code
ASSF[0-1] char (1)	source of supply flag
ASSSF[0-1] char (1)	secondary source of supply flag
AORIG[0-1] char (1)	origin code
ASPRTL[0-1] char (2)	substitute part - label
ASPRTC[0-1] char (2)	substitute part - cmfg
ASPART[0-1] char (10)	substitute part - basic
ASPRTS[0-1] char (2)	substitue part - suffix
ABLC[0-1] char (2)	basic language code
ALMDAT[0-1] date (10)	date of last file maint

AMTIME[0-1] numeric (6) time of last file maint field length : 6

ALUSER[0-1] char (10) last maintained user id

APRDTP[0-1] char (3) product type

AMINI[0-1] numeric (9) minimum qty for orders on call field length : 5

ACHAR1[0-1] char (50) part description

ACHAR2[0-1] char (50) part description

ACHAR3[0-1] char (50) part description

id: ACMPNY, APARTL, APARTC, APART, APARTS from unique index

ref : ABUYUM -> UNIT OF MEASURE.ZUNMS

ref : APLTAN -> BUYER ANALYST.ZBAC

ref : ABAC -> BUYER ANALYST.ZBAC

group: AOPCD, AOPQT AOPQT = 0 <=> AOPCD = 3

group: ACMPNY, AVENDR -> VENDOR IDENTIFICATION.(G3CMNY, G3VND#)
(if AACTCD <> 'D')

-> VENDOR ADDRESS.(G4CMNY,G4VEND) ou
-> VENDOR IDENTIFICATION.(G3CMNY,G3VND#)

PAYMENT CODES / IPSP03

CMNY03 char (2) company code

ACTI03 char (1) generic flag - system wide

PAF203 char (3) intl affiliate code

DESC03 char (50) generic description field

PGMN03 char (10) pgm name

LMDA03 date (10) date last maintained

LMTM03 numeric (6) time last maintained field length : 6

LUSR03 char (10) user ID that last maintained

WRKS03 char (10) work station

id: CMNY03, PAF203 if ACTI03 <> 'D' (from unique index)

PAYMENT TERMS / PAYTRM description of payment term

ZPAYCD char (1)	payments terms	
ZDAYS numeric (3)	number of days	field length : 2; value <> 0
ZPAYLG char (2)	language code; '**' => no reference to LANGUGAGE CODE	
ZPAYDE char (25)	description	
ZLADT date (10)	last activity date	
ZLATM numeric (6)	last activity time	field length : 6
ZUSER char (10)	workstation user I.D.	

id: ZPAYCD, ZPAYLG *from unique index; from program*

ref : ZPAYLG -> LANGUAGE CODE.ZBCODE

PRICE DEAL HEADER / POPDHD description of contracts with the suppliers

HDSTAT char (1)	status code (A = approved, D = deapproved)	
HDPRTL char (2)	part number - label	
HDPRTC char (2)	part number - country	
HDPART char (10)	part number - basic code	
HDPRTS char (2)	part number - suffix	
HDCMPR char (15)	compressed part	
HDRANK char (1)	vendor ranking	
HDSLED numeric (3)	supplier lead time	field length : 3
HDDLED numeric (3)	delivery lead time	field length : 3
HDCONT char (10)	contract number	
HDHGP numeric (3,1)	handling charge percent	field length : 2
HDDTP numeric (3,1)	duties percent	field length : 2
HDDATE date (10)	generic date - YYMMDD	
HDTIME numeric (6)	generic time - HHMMSS	field length : 6
HDUSER char (10)	workstation user I.D.	
HDCMNY char (2)	ref of HDVEND	
HDVEND char (7)	ref of HDVEND	

id: HDCMNY, HDVEND, HDPART, HDPRTS, HDPRTL, HDPRTC *from unique index*

group: HDCMNY, HDVEND *-> VENDOR IDENTIFICATION.(G2CMNY, G3VND#)*

(if HDSTAT <> 'D')

PURCHASE COMMENT / PURCOM Comments on the order (header = general comments, vendor = message for the supplier, item = comment on one order line)

G8STAT char (1)	O = open, C = close, S = suspended, D = deleted
G8PREV char (1)	previous status
G8CMNY char (2)	company code
G8TYPE char (1)	H = header, V = vendor, P = part (= item)
G8BAC char (2)	buyer analyst code
G8ORNO char (6)	order number
G8LINE numeric (3)	order line no. field length : 3
G8REQ char (8)	requisition number
G8COMM numeric (2)	comment line no. field length : 2
G8COMD char (50)	comments text
G8WRT char (1)	print (Y/N) ?
G8LADT date (10)	generic date - YYMMDD
G8LATM numeric (6)	generic time - HHMMSS field length : 6
G8USER char (10)	workstation user I.D.

ref : G8CMNY, G8BAC, G8ORNO -> PURCHASE MASTER.(G1CMNY, G1BAC, G1ORNO)

PURCHASE CONTROL FILE / IPSP06 used to check if someone works on a purchase order

CMNY06 char (2)	company code
RECE06 char (6)	receipt number
BAC06 char (2)	buyer analyst
ORD06 char (6)	order number
PGMN06 char (10)	program name
LMDA06 date (10)	date last maintained
LMTM06 numeric (6)	time last maintained field length : 6
LUSR06 char (10)	user I.D. that last maint
WRKS06 char (10)	workstation

ref : CMNY06, BAC06, ORD06 -> PURCHASE MASTER.(G1CMNY, G1BAC, G1ORNO)

PURCHASE ITEM / PURITM	item part of a purchase order
G2STAT[0-1] char (1)	O = open, C = closed, N = cancelled, D = deleted
G2PREV[0-1] char (1)	O = open, C = closed, N = cancelled, D = deleted
G2CMNY char (2)	company code
G2BAC char (2)	buyer analyst code
G2ORNO char (6)	order number
G2LINE numeric (3)	order line number field length : 3
G2REQ[0-1] char (8)	requisition number
G2TYPE char (1)	purchase order type
G2RLIN[0-1] numeric (3)	requisition number field length : 3
G2PRTL[0-1] char (2)	part number - label
G2PRTC[0-1] char (2)	part number - country
G2PART[0-1] char (10)	part number - basic code
G2PRTS[0-1] char (2)	part number - suffix
G2CMPR[0-1] char (15)	compressed part
G2NONS[0-1] char (50)	non-stock part
G2COM[0-1] char (1)	N = none, D = default vendor comment, P = purcom item comment
G2ALOC[0-1] char (1)	allocated part (Y/N)
G2WHSE char (2)	warehouse
G2DEPT[0-1] char (3)	department number
G2ITDT[0-1] date (10)	order date
G2IODT[0-1] date (10)	mrp and requisition requirement date
G2DLDT[0-1] date (10)	delivery date
G2DEST[0-1] date (10)	despatch date
G2OORQ[0-1] numeric (9)	order qty in stocking U/M field length : 5
G2IORQ[0-1] numeric (9)	init. order qty in stocking field length : 5
G2OBDU[0-1] numeric (9)	total bal. due in stocking field length : 5
G2PORG[0-1] char (1)	E = entered price, S = standard cost, P = price deal
G2DORG[0-1] char (1)	E = entered date, C = calculated date
G2PFLG[0-1] char (1)	N = not changed, C = changed
G2CFLG[0-1] char (1)	receipt consumption flag
G2UNMS[0-1] char (2)	partms stocking unit measure

G2UNMB[0-1] char (2)	partms buying unit measure
G2UNMP char (2)	U/M used last in P.O. maint
G2QTYR[0-1] numeric (9)	total qty received in stocking field length : 5
G2DECL[0-1] numeric (1)	local decimal control field length : 1
G2DECV[0-1] numeric (1)	vendor decimal control field length : 1
G2COST[0-1] numeric (15)	vendor stkg U/M price field length : 8
G2COSF[0-1] numeric (15)	local stkg U/M price field length : 8
G2CSTB[0-1] numeric (15)	vendor buying U/M price field length : 8
G2CSFB[0-1] numeric (15)	local buying U/M price field length : 8
G2DCST[0-1] numeric (15)	vendor curr stkg discount field length : 8
G2DCSF[0-1] numeric (15)	local curr stkg discount
G2DCTB[0-1] numeric (15)	vendor curr buying discount field length : 8
G2DCFB[0-1] numeric (15)	local curr buying discount field length : 8
G2RATE[0-1] numeric (10,5)	currency rate field length : 6
G2CSQ1[0-1] numeric (2)	current sequence no. 1 field length : 2
G2CSQ2[0-1] numeric (2)	advice time printed field length : 2
G2CSQ3[0-1] numeric (2)	recall time printed field length : 2
G2CDT1[0-1] date (10)	MRO transfer date
G2CDT2[0-1] date (10)	advice date printed
G2CDT3[0-1] date (10)	recall date printed
G2ADVF[0-1] char (1)	advice type M = advmst, P = advitm
G2ADVS[0-1] char (1)	advice status Y = yes, N = no, C = complt
G2RECS[0-1] char (1)	recall status Y/N/C = complt
G2CQT1[0-1] numeric (9)	cur order stk qty no. 1 field length : 5
G2CQT2[0-1] numeric (9)	cur order stk qty no. 2 field length : 5
G2CQT3[0-1] numeric (9)	cur order stk qty no. 3 field length : 5
G2CTR1[0-1] numeric (9)	qty on dock in stk um field length : 5
G2CTR2[0-1] numeric (9)	qty on dock in buy um field length : 5
G2CTR3[0-1] numeric (9)	cur total recd stk no. 3 field length : 5
G2CBD1[0-1] numeric (9)	cur stk. bal due no. 1 field length : 5
G2CBD2[0-1] numeric (9)	cur stk. bal due no. 2 field length : 5
G2CBD3[0-1] numeric (9)	cur stk. bal due no. 3 field length : 5
G2MQTS[0-1] numeric (9)	matched receipt qty-stock umfield length : 5

G2MQTB[0-1] numeric (9)	matched receipt qty-buy um	field length : 5
G2OORB[0-1] numeric (9)	order quantity (buying um)	field length : 5
G2IORB[0-1] numeric (9)	initial order qty - buying um	field length : 5
G2OBDB[0-1] numeric (9)	total ball due in buying	field length : 5
G2QTYB[0-1] numeric (9)	total qty received in buying um	field length : 5
G2CQB1[0-1] numeric (9)	cur buying um qty no. 1	field length : 5
G2CQB2[0-1] numeric (9)	cur buying um qty no. 2	field length : 5
G2CQB3[0-1] numeric (9)	cur buying um qty no. 3	field length : 5
G2CBR1[0-1] numeric (9)	cur tot. rcvd buying no. 1	field length : 5
G2CBR2[0-1] numeric (9)	cur tot. rcvd buying no. 2	field length : 5
G2CBR3[0-1] numeric (9)	cur tot. rcvd buying no. 3	field length : 5
G2BBD1[0-1] numeric (9)	cur buying um bal. due no. 1	field length : 5
G2BBD2[0-1] numeric (9)	cur buying um bal. due no. 2	field length : 5
G2BBD3[0-1] numeric (9)	cur buying um bal. due no. 3	field length : 5
G2LRDT[0-1] date (10)	last receipt date (YYMMDD)	field length : 6
G2REQ1[0-1] char (8)	requisition number 1	
G2REQ2[0-1] char (8)	requisition number 2	
G2REQ3[0-1] char (8)	requisition number 3	
G2CLSD[0-1] date (10)	close date	
G2ACCT[0-1] numeric (14)	account number	field length : 14
G2CONT[0-1] char (10)	contract number	
G2VDIS[0-1] numeric (4,2)	vendor discount %	field length : 3
G2PDIS[0-1] numeric (4,2)	part discount %	field length : 3
G2PRIN[0-1] char (1)	print the price	
G2ORTY char (1)	order type	in { 'N', 'C', 'M' }
G2VATC[0-1] char (1)	vat code	
G2MROD[0-1] numeric (5)	MRO request number	field length : 5
G2CPAN[0-1] numeric (6)	cpa number	field length : 6
G2ASST[0-1] numeric (6)	asset number	field length : 6
G2MROF[0-1] numeric (3)	mro familly	field length : 2
G2MROG[0-1] numeric (2)	mro sub-grp	field length : 2
G2MROS[0-1] numeric (3)	produc sequence	field length : 2
G2LADT date (10)	last activity date (YYMMDD)	

G2LATM numeric (6) last activity time (HHMMSS) field length : 6

G2USER char (10) last activity user I.D.

G2PROG char (5) program I.D.

id: G2CMNY, G2BAC, G2ORNO, G2LINE if G2TYPE = 'P' (from unique index)

ref : G2UNMP -> UNIT OF MEASURE.ZUNMS

ref : G2CMNY, G2BAC, G2ORNO -> PURCHASE MASTER.(G1CMNY, G1BAC, G1ORNO)

group: G2CMNY, G2BAC, G2ORNO, G2ORTY, G2WHSE

(G2ORTY, G2WHSE) = ORDER HEADER.PURCHASE MASTER.(G1ORTY, G1WHSE)

group: G2ORTY, G2NONS, G2CMPR, G2MROF, G2MROG, G2MROS, G2MROD

G2ORTY = 'N' => G2NONS <> NULL

G2ORTY = 'C' => G2CMPR <> NULL or G2NONS <> NULL

G2ORTY = 'M' => G2MROF, G2MROG, G2MROS, G2MROD <> NULL

group: G2CMNY, G2BAC, G2ORNO, G2ORTY, G2ACCT, G2CPAN, G2CMPR, G2PRTL, G2PRTC, G2PART, G2PRTS, G2CONT, G2OORQ, G2OORB, G2OBDU, G2QTYR, G2OBDB, G2QTYB, G2UNMS, G2UNMB, G2VATC Rem : G1... corresponds to a record of ORDER HEADER.PURCHASE MASTER

G2ORTY <> 'C' =>

G1CPA = 'Y' => G2ACCT = 0, G2CPAN <> 0

G1CPA = 'N' => G2ACCT <> 0

G2ORTY = 'M' => G2ACCT = 0

G2ORTY <> 'M' and G1CPA <> 'Y' =>

G2CMPR = NULL or G2ACCT = 0

if G2CMPR = NULL, G2ACCT <> 0

G2CMPR <> NULL => G2CMPR = (G2PRTL, G2PRTC, G2PART, G2PRTS)
(G2CMNY, G2PRTL, G2PRTC, G2PART, G2PRTS) -> PART MASTER.(ACMPNY, APARTL, APARTC, APART, APARTS)
(+AACTCD = 'A', ABYCUM <> 0)

if BUYER.(ZAPPRO) (BUYER.(ZBAC) = ORDER HEADER.PM.G1BAC) = 'Y',
(G2PRTL, G2PRTC, G2PART, G2PRTS, G1VND#, G1VADD) ->
IPSP01.(PLAB01, PCOM01, PBAS01, PSUF01, SUPL01, ADDR01)

if G1VND# <> '9999999' and G2NONS = NULL,

(G2CMNY, G2PART, G2PRTS, G2PRTL, G2PRTC, G1VND#) ->

POPDHD.(HDCMNY,HDPART,HDPRTS,HDPRTL,HDPRTC,HDVEND)

G2NONS <> NULL =>

G2CONT = NULL

G2OORB <> NULL

G2OORQ = G2OORB

G2OBDU = G2OORQ - G2QTYR

G2OBDB = G2OORB - G2QTYB

G2UNMS, G2UNMB <> NULL

G2UNMS -> UNITMS.(ZUNMS)

G2UNMB -> UNITMS.(ZUNMS)

G2VATC -> VATCTL.(ZCCODE)

group: G2CMNY, G2BAC, G2ORNO, G2CMPR, G2PART, G2PRTS, G2PRTL, G2PRTC

G2CMPR <> NULL =>

(G2CMNY, OH.PM.G1VND#, G2PART, G2PRTS, G2PRTL, G2PRTC) -> POPDQP.(QPCMNY, QPVEND, QPPART, QPPRTS, QPPRTL, QPPRTC)

OH = ORDER HEADER, PM = PURCHASE MASTER

group: G2CMNY, G2BAC, G2ORNO, G2ORTY, G2NONS, G2PART, G2PRTS, G2PRTL, G2PRTC, G2CONT, G2OBDU, G2OORQ, G2QTYR, G2UNMS, G2UNMB, G2OBDB, G2OORB, G2QTYB G2ORTY = 'C' and G2NONS = NULL =>

if (G2CMNY, G2PART, G2PRTS, G2PRTL, G2PRTC, OH.PM.G1VND#) ->

POPDHD.(HDCMNY, HDPART, HDPRTS, HDPRTL, HDPRTC, HDVEND),

then G2CONT -> POPDHD.(HDCONT)

else G2CONT = NULL

G2OBDU = G2OORQ - G2QTYR

G2UNMS -> PARTMS.(UNMS)

G2UNMB -> PARTMS.(UNMS)

(PARTMS which verifies (...) -> PARTMS.(.....))

G2OBDB = G2OORB - G2QTYB

group: G2ORTY, G2MROF, G2MROG, G2MROS, G2CMPR

if G2ORTY = 'M', G2CMPR = (G2MROF, G2MRG, G2MROS)

PURCHASE MASTER / PURMST master part of a purchase order

G1STAT[0-1] char (1)	status : o = open, c = close, s = suspended, d = deleted
G1PREV[0-1] char (1)	previous status : o = open, c = close, s = suspended, d = deleted
G1CMNY char (2)	company code
G1BAC char (2)	buyer analyst code
G1ORNO char (6)	order number
G1REQ[0-1] char (8)	requisition number
G1PURT[0-1] char (1)	p = po, r = requisition
G1POOG[0-1] char (1)	G = generated, E = entered
G1ORDT date (10)	order date
G1VND# char (7)	vendor number
G1OVND[0-1] char (7)	original vendor
G1ORIG[0-1] char (10)	originator
G1PFLG char (1)	print flag : N = don't print, C = print changes, P = print all in { 'C', 'N', 'P' }
G1PNUM[0-1] numeric (3)	number times printed field length : 2
G1METH[0-1] char (2)	del method
G1DMDE[0-1] char (10)	del meth desc
G1LNCD[0-1] char (2)	langugage code
G1DELT char (2)	delivery terms
G1DTDE[0-1] char (10)	deliv term description
G1PAYT char (1)	payment terms
G1PTDE[0-1] char (10)	payment term description
G1VADD[0-1] char (2)	vendor address ovrd
G1OSEQ[0-1] numeric (5)	one time vendor seq field length : 3
G1WHSE[0-1] char (2)	warehouse
G1BILL[0-1] numeric (6)	bill-to customer nr field length : 4
G1SHIP[0-1] numeric (3)	ship-to customer nr field length : 2

G1COM[0-1] char (1)	N = none, D = default vendor comment, P = purcom order
G1CURR[0-1] char (2)	currency code
G1ODDT[0-1] date (10)	original due - date
G1DELDT[0-1] date (10)	delivery date
G1DESDT[0-1] date (10)	despatch date
G1CLDT[0-1] date (10)	order close date
G1CFLG[0-1] char (1)	confirmation req flag
G1CDAY[0-1] numeric (3)	# of days to follow confirm field length : 3
G1MPOC char (1)	meth of po comm in { 'T', 'X', 'F', 'M' }
G1CONT[0-1] char (10)	contract
G1STDV[0-1] numeric (11)	std po total value field length : 6
G1VNDV[0-1] numeric (11)	vendor po total value field length : 6
G1LOCV[0-1] numeric (11)	local po total value field length : 6
G1STDC[0-1] char (3)	standard comment
G1ORTY char (1)	order type (M = MRO, C = CODIFIED, N = NON CODIFIED) in { 'C', 'M', 'N' }
G1CPA char (1)	cpa order (Y/N) in { 'Y', 'N' }
G1APPL[0-1] char (1)	
G1LADT date (10)	last activity date
G1LATM numeric (6)	last activity time (HHMMSS) field length : 6
G1USER char (10)	last activity user I.D.
G1PROG char (5)	program I.D.

id: G1CMNY, G1BAC, G1ORNO if G1PURT = 'P' (from unique index; from program)

ref : G1CMNY, G1VND# -> VENDOR IDENTIFICATION.(G3CMNY, G3VND#) if G1VND# <> '9999999' and PURVEN.(G3STAT) <> 'S','O'

ref : G1CMNY, G1VND#, G1VADD -> VENDOR ADDRESS.(G4CMNY, G4VEND, G4CODE) if G1VND# <> '9999999' and G1VADD <> NULL + G4OSEQ = 0

ref : G1CMNY, G1WHSE -> WAREHOUSE.(Z1CMNY, Z1WHSE)

ref : G1LNCD, G1DELT -> DELIVERY TERMS.(ZTRMCD, ZTRMLG)

ref : G1LNCD, G1METH -> DELIVERY METHOD .(ZLNGCD, ZMTHCD)

ref : G1CMNY, G1VND#, G1VADD -> VENDOR ADDRESS.(G4CMNY, G4VEND, G4CODE) if G1VADD <> NULL + G4OSEQ = 0

ref : G1PAYT, G1LNCD -> PAYMENT TERMS.(ZPAYCD, ZPAYLG)

ref : G1CMNY, G1STDC -> STANDARD COMMENTS.(CMNY02, CODE02)

ref : G1LNCD -> LANGUAGE CODE.ZBCODE

ref : G1CURR -> CURRENCY.ZCURCOD

group: G1BAC, G1ORTY -> BUYER.(ZBAC, ZORTY)

group: G1LNCD, G1CURR, G1CMNY, G1VND#, G1METH

if G1VND# <> '9999999',

(G1CMNY, G1VND#, G1METH, G1CURR, G1LNCD) ->

PURVEN.(G3CMNY, G3VND#, G3DLMT, G3CUR, G3LGCD)

group: G1WHSE, G1BILL, G1SHIP

G1WHSE = NULL <=> G1BILL <> NULL or G1SHIP <> 0

group: G1LNCD, G1DELT, G1DTDE -> DELTRM.(ZTRMCD, ZTRMLG, ZTRMDE)

group: G1CURR, G1VND#, G1METH, G1VADD

G1VND# = '9999999' => G1METH, G1CURR <> NULL, G1VADD = NULL

group: G1LNCD, G1METH, G1DMDE -> DELMTH.(ZMTHCD, ZLNGCD, ZDESC)

group: G1LNCD, G1PAYT, G1PTDE -> PAYTRM.(ZPAYCD, ZPAYLG, ZPAYDE)

group: G1MPOC, G1VADD

G1MPOC = 'X' => if G1VADD <> NULL, PURADD.(G4TELX) <> NULL

if G1VADD = NULL, PURVEN.(G3TELX) <> NULL

G1MPOC = 'F' => if G1VADD <> NULL, PURADD.(G4FAX) <> NULL

if G1VADD = NULL, PURVEN.(G3FAX) <> NULL

STANDARD COMMENTS / IPSP02 standard comments

CMNY02 char (2)	company code
ACTI02[0-1] char (1)	generic flag - system wide
CODE02 char (3)	standard code
SHOR02 char (15)	short description
COMM02[0-1] char (50)	description
LINE02 numeric (3)	comment line number field length : 2; value > 0
PGMN02 char (10)	program name
LMDA02 date (10)	date last maintained
LMTM02 numeric (6)	time last maintained field length : 6
LUSR02 char (10)	user I.D. that last maintained

WRKS02 char (10) workstation

id: CMNY02, CODE02 *from program*

STATUS CODE / IPSP08

ACTI08 char (1) activity code

STAT08 char (1) supplier status

DESC08 char (20) status description

AUDI08 numeric (3) audit delay in months field length : 2

PGMN08 char (6) program name

USID08 char (10) user ID

WSTN08 char (10) workstation

MDAT08 date (10) last maintenance date

MTIM08 numeric (6) last maintenance time field length : 4

id: STAT08 if ACTI08 <> 'D' (from unique index)

SUPPLIER APPROBATION / IPSP01 tests a supplier regarding a product

PLAB01 char (2) product label -dla-

PCOM01 char (2) product country of manuf -dcm-

PBAS01 char (10) basic part of the product code

PSUF01 char (2) product suffix

SUPL01 char (7) supplier no.

ADDR01[0-1] char (2) address code

ADDR01 <> NULL => (SUPL01, ADDR01, VNAM01) -> PURADD.(G4VEND, G4CODE, G4VNDN)

ADDR01 = NULL => (SUPL01, VNAM01) -> PURVEN.(G3VND#, G3VNDN)

STAT01 char (1) A = approved, E = evaluation

DATA01[0-1] date (10) date of supplier approbation

DATE01[0-1] date (10) date of last supplier evaluation

DATC01[0-1] date (10) date of certification

DATI01[0-1] date (10) date of integration

DACC01[0-1] date (10) date of acceptation

SIT101[0-1] char (80) site of production / not used

PDES01 char (50) product description

VNAM01[0-1] char (35) vendor name

VREF01[0-1] char (80) vendor product reference

USER01 char (10) user I.D.

MDAT01 date (10) mnce date

TIME01 numeric (6) time field length : 4

WSTN01 char (10) workstation

PGMN01 char (6) program name

id: PLAB01, PCOM01, PBAS01, PSUF01, SUPL01 from program

id': PLAB01, PCOM01, PBAS01, PSUF01, SUPL01, ADDR01, STAT01
from unique index

id': SUPL01, ADDR01, STAT01, PLAB01, PCOM01, PBAS01, PSUF01
from unique index

ref : STAT01 -> STATUS CODE.STAT08

group: PLAB01, PCOM01, PBAS01, PSUF01, PDES01 ->
PARTMS.(APRTL,APRTC,APART,APRTS,ADESC)

group: STAT01, DATI01 STAT01 = 'I' => DATI01 <> 0

UNIT OF MEASURE / UNITM description of a unit of measure

ZUNMS char (2) unit of measure

ZDESC char (25) unit of measure description

id: ZUNMS from unique index; from program

VAT CONTROL / VATCTL VAT control file

ZCCODE char (1) VAT code

ZCPERC numeric (5,2) percentage field length : 3

ZCLADT date (10) last activity date

ZCLATM numeric (6) last activity time field length : 6

ZCUSER char (10) last activity user I.D.

id: ZCCODE from unique index; from program

VENDOR ADDRESS / PURADD contains the different addresses of a supplier

G4CMNY char (2) company code
G4VEND char (7) vendor number
G4CODE char (2) address code
G4SEQ numeric (5) one time vendor seq field length : 3
G4ADD1 char (30) generic address field
G4ADD2[0-1] char (30) generic address field
G4ADD3[0-1] char (30) generic address field
G4ADD4[0-1] char (30) generic address field
G4ADD5[0-1] char (30) generic address field
G4CONT[0-1] char (20) customer contract
G4MPOC char (1) method of po comm in { 'T', 'X', 'F', 'M' }
G4VOLT[0-1] numeric (3) vendor override lead time field length : 2
(G4CMNY, G3VND#, G3VOLT) -> PURVEN.(G3CMNY, G3VND#, G3SPLT)
G4PCOD[0-1] char (2) phone country code
G4PHAR[0-1] char (5) phone area code
G4PHON[0-1] char (10) phone number
G4TEAR[0-1] char (5) telex area code
G4TELX[0-1] char (8) telex number
G4FCOD[0-1] char (2) fax country code
G4FXAR[0-1] char (5) fax area code
G4FAX[0-1] char (10) fax number
G4LADT date (10) generic date
G4LATM numeric (6) generic time HHMMSS field length : 6
G4USER char (10) workstation user I.D.
G4VNDN char (35) vendor name

id: G4CMNY, G4VEND, G4CODE from program
id': G4CMNY, G4VEND, G4CODE, G4SEQ from unique index
ref : G4CMNY, G4VEND -> VENDOR IDENTIFICATION.(G3CMNY, G3VND#)
if VENDOR IDENTIFICATION.G3STAT <> 'D'
group: G4CMNY, G4VEND, G4VNDN -> PURVEN.(G3CMNY, G3VND#, G3VNDN)
if PURVEN.G3STAT <> 'D'

group: G4PCOD, G4PHAR, G4PHON

G4PCOD <> NULL => G4PHAR, G4PHON <> NULL

group: G4FCOD, G4FXAR, G4FAX

G4FCOD <> NULL => G4FXAR, G3FAX <> NULL

group: G4PHAR, G4PHON

G4PHAR <> NULL => G4PHON <> NULL

group: G4FXAR, G4FAX

G4FXAR <> NULL => G3FAX <> NULL

group: G4TEAR, G4TELX

G3TEAR <> NULL => G3TELX <> NULL

group: G4MPOC, G4PHAR, G4PHON

G4MPOC = 'T' => G4PHAR, G4PHON <> NULL

group: G4MPOC, G4FXAR, G4FAX

G4MPOC = 'F' => G4FXAR, G4FAX <> NULL

group: G4MPOC, G4TEAR, G4TELX

G4MPOC = 'X' => G4TEAR, G4TELX <> NULL

VENDOR COMMENT / VENCOM comments for the vendor

QASTAT[0-1] char (1)	record activity code
QACMNY char (2)	company code
QAVEND char (7)	vendor number
QASEQ numeric (3)	vendor sequence number field length : 2
QAPCD[0-1] numeric (1)	print code field length : 1
QACOM char (50)	print comment
QADATE date (10)	last maint date
QATIME numeric (6)	last maint time field length : 6
QAUSER char (10)	last maint user

ref : QACMNY, QAVEND -> VENDOR IDENTIFICATION.(G3CMNY, G3VND#)

VENDOR IDENTIFICATION / PURVEN contains the description of the suppliers

G3STAT[0-1] char (1)	vendor status code, A/O/D
G3CMNY char (2)	company code
G3VND# char (7)	vendor number value <> '9999999'

G3TYPE[0-1] char (2)	vendor type - grouping flag
G3VNDN char (35)	vendor name
G3ADD1 char (30)	vendor address line 1
G3ADD2[0-1] char (30)	vendor address line 2
G3ADD3[0-1] char (30)	vendor address line 3
G3ADD4[0-1] char (30)	vendor address line 4
G3ADD5[0-1] char (30)	vendor address line 5
G3PAY[0-1] char (1)	vendor payment flag Y/N in { 'Y', 'N' }
G3ATN[0-1] char (20)	name of contact at vendor
G3CUR char (2)	currency code code
G3PAYT char (1)	payment terms
G3DLMT char (2)	delivery method
G3SPLT[0-1] numeric (3)	vendor lead time field length : 2
G3LGCD char (2)	language code
G3DELT char (2)	delivery method terms
G3MPOC char (1)	method of PO comm in { 'T', 'X', 'F', 'M' }
G3PCOD[0-1] char (2)	phone country code
G3PHAR[0-1] char (5)	phone area code
G3PHON[0-1] char (10)	phone number
G3TEAR[0-1] char (5)	telex area code
G3TELX[0-1] char (8)	telex number
G3FCOD[0-1] char (2)	fax country code
G3FXAR[0-1] char (5)	fax area code
G3FAX[0-1] char (10)	fax number
G3CORP char (1)	corp flag in { 'F', 'D', 'I' }
G3CONT[0-1] char (1)	contract flag in {'Y', 'N'}
G3WHSE[0-1] char (2)	primary warehouse
G3DISC[0-1] numeric (4,2)	discount field length : 3
G3CDAY[0-1] numeric (3)	# of days to follow confirm field length : 2
G3BANK[0-1] numeric (12)	bank number field length : 12
G3VAT[0-1] char (16)	vat number
G3CVAT char (1)	vat code Y/N in { 'Y', 'N' }
G3ADR1[0-1] char (50)	foreign address line 01

G3ADR2[0-1] char (50) foreign address line 02

G3ADR3[0-1] char (50) foreign address line 03

G3BKF[0-1] char (34) foreign bank number

G3IBLC char (3) iblc code

G3PAFF[0-1] char (1) payment code part one in { 'T' }

G3PAF2 char (3) payment code part two

G3UPDE[0-1] char (1) updated by cgas Y/*BL

G3EURC char (3) european coutry code

G3LADT date (10) last activity date

G3LATM numeric (6) last activity time field length : 6

G3USER char (10) last activity user I.D.

id: G3CMNY, G3VND# from unique index; from program

ref : G3CMNY, G3WHSE -> WAREHOUSE.(Z1CMNY, Z1WHSE)

ref : G3DELT, G3LGCD -> DELIVERY TERMS.(ZTRMCD, ZTRMLG)

ref : G3PAYT, G3LGCD -> PAYMENT TERMS.(ZPAYCD, ZPAYLG)

ref : G3LGCD -> LANGUAGE CODE.ZBCODE

ref : G3DLMT, G3LGCD -> DELIVERY METHOD .(ZLNGCD, ZMTHCD)

ref : G3CMNY, G3PAF2 -> PAYMENT CODES.(CMNY03, PAF203)

ref : G3CUR -> CURRENCY.ZCURCOD

group: G3PCOD, G3PHAR, G3PHON

G3PCOD <> NULL => G3PHAR, G3PHON <> NULL

group: G3FCOD, G3FXAR, G3FAX

G3FCOD <> NULL => G3FXAR, G3FAX <> NULL

group: G3PHAR, G3PHON

G3PHAR <> NULL => G3PHON <> NULL

group: G3FXAR, G3FAX

G3FXAR <> NULL => G3FAX <> NULL

group: G3TEAR, G3TELX

G3TEAR <> NULL => G3TELX <> NULL

group: G3MPOC, G3PHAR, G3PHON

G3MPOC = 'T' => G3PHAR, G3PHON <> NULL

group: G3MPOC, G3FXAR, G3FAX

G3MPOC = 'F' => G3FXAR, G3FAX <> NULL

group: G3MPOC, G3TEAR, G3TELX

G3MPOC = 'X' => G3TEAR, G3TELX <> NULL

group: G3CVAT, G3VAT

G3CVAT = 'Y' => G3VAT <> NULL

G3CVAT = 'N' => G3VAT = NULL

group: G3BKF, G3ADR1, G3ADR2, G3ADR3

G3BKF <> NULL => G3ADDR1, G3ADDR2, G3ADDR3 <> NULL

group: G3CMNY, G3VND#, G3LGCD

Exists PURLED.(G5CMNY, G5VEND, G5LNGC) -> (G3CMNY, G3VND#, G3LGCD) and PURLED.(G5METH, G5LNGC) -> DELMTH.(ZMTHCD, ZLNGCD)

VENDOR/PART COMMENTS / PRTCOM comments on a product of a supplier

QCSTAT char (1)	status code (A = approved, D = deapproved)
QCCMNY char (2)	company code
QCPRTL char (2)	part number - label
QCPRTC char (2)	part number - country
QCPART char (10)	part number - basic code
QCPRTS char (2)	part number - suffix
QCCMPR char (15)	compressed part
QCVEND char (7)	vendor number
QCSEQ numeric (3)	comment sequence number field length : 2
QCPCD numeric (1)	print code field length : 1
QCCOM char (50)	print comment
QCDATE date (10)	last maint date
QCTIME numeric (6)	last maint time field length : 6
QCUSER char (10)	last maint user

id: QCCMNY, QCPART, QCPRTS, QCPRTL, QCPRTC, QCVEND, QCSEQ

from unique index

id': QCCMNY, QCPART, QCPRTS, QCPRTL, QCPRTC, QCVEND, QCSEQ

if QCSTAT <> 'D' (from unique index)

WAREHOUSE / WHOUSE warehouse

Z1CMNY char (2)	company code
Z1WHSE char (2)	warehouse
Z1DESC char (35)	warehouse description
Z1ADD1[0-1] char (35)	warehouse address 1
Z1ADD2[0-1] char (35)	warehouse address 2
Z1ADD3[0-1] char (35)	warehouse address 3
Z1CSDE char (1)	consigned inventory in {'Y', 'N'}
Z1INTP char (3)	in-transit inventory type
Z1DEPT[0-1] char (3)	default department for QC moves
Z1RSDY numeric (3)	number of days ahead to set requested ship date
	field length : 2; in {1,..,365}
ZLADT date (10)	last activity date
ZLATM numeric (6)	last activity time
ZUSER char (10)	last activity user
id: Z1CMNY, Z1WHSE	*from unique index; from program*

Program Main/Executed (graph)

PURCHASE ITEM
G2STAT[0-1]
G2PREV[0-1]
G2CMNY
G2BAC
G2ORNO
G2LINE
G2RSEQ[0-1]
G2TYPE
G2RLIN[0-1]
G2PRTL[0-1]
G2PRTC[0-1]
G2PART[0-1]
G2PRTS[0-1]
G2CMPR[0-1]
G2MONS[0-1]
G2COM[0-1]
G2ALOC[0-1]
G2WHSE
G2DEPT[0-1]
G2TDT[0-1]
G2DOT[0-1]
G2DLDT[0-1]
G2DEST[0-1]
G2OORQ[0-1]
G2SORQ[0-1]
G2OREDU[0-1]
G2FORG[0-1]
G2DORG[0-1]
G2PFLG[0-1]
G2CFLG[0-1]
G2UNMS[0-1]
G2UNMB[0-1]
G2UMSP
G2QTYR[0-1]
G2DSCL[0-1]
G2DECV[0-1]
G2COST[0-1]
G2COSP[0-1]
G2CSTN[0-1]
G2CSPR[0-1]
G2DCST[0-1]
G2DCSP[0-1]
G2DCTR[0-1]
G2DCPR[0-1]
G2RATE[0-1]
G2CSQ1[0-1]
G2CSQ2[0-1]
G2CSQ3[0-1]
G2CDT1[0-1]
G2CDT2[0-1]
G2CDT3[0-1]
G2ADV1[0-1]
G2ADV3[0-1]
G2SRCE[0-1]
G2CQT1[0-1]
G2CQT2[0-1]
G2CQT3[0-1]
G2CTR1[0-1]
G2CTR2[0-1]
G2CTR3[0-1]
G2CBD1[0-1]
G2CBD2[0-1]
G2CBD3[0-1]
G2MQTS1[0-1]
G2MQTS2[0-1]
G2MQTS3[0-1]
G2MQTR[0-1]
G2OORS[0-1]
G2SORS[0-1]
G2LADT
G2LATM
G2USER
G2PROG
id: G2CMNY G2BAC G2ORNO G2LINE
ref: G2UNMP
ref: G2CMNY G2BAC G2ORNO
gr: G2CMNY G2BAC G2ORNO G2ORTY G2WHSE
gr: G2ORTY G2MONS G2CMPR G2MROP G2MROG G2MROS G2MROD
gr: G2CMNY G2BAC G2ORNO G2ORTY G2ACCT G2CPAN G2CMPR G2PRTL G2PRTC G2PART G2PRTS G2CCNT G2OORQ G2SORB G2OREDU G2QTYR G2SRDB G2QTYB G2UNMS G2UNMB G2VATC
gr: G2MONS G2CMNY G2BAC G2ORNO
gr: G2CMNY G2BAC G2ORNO G2CMPR G2PART G2PRTS G2PRTL G2PRTC
gr: G2CMNY G2BAC G2ORNO G2ORTY G2MONS G2PART G2PRTS G2PRTL G2PRTC G2CCNT G2OORQ G2SORB G2OREDU G2QTYR G2SRDB G2QTYB
gr: G2ORTY G2MROP G2MROG G2MROS G2CMPR

VAT CONTROL
ZCODE
ZCPERC
ZCLADT
ZCLATM
ZCUSER
id: ZCODE

DELMETHLEADTIME
GSSTAT
GSCMNY
GSVEND
GSMETH
GSLNGC
GSLT
id: GSVEND GSMETH GSLNGC

PART CROSS REFERENCE
CRCOMP
CRLIN
CRCOM
CRBAS
CRSUFX
CRCMPR
id: CRCOMP CRCMPR
id': CRCOMP CRBAS CRSUFX CRLIN CRCOM

UNIT OF MEASURE
ZUNMS
ZDESC
id: ZUNMS

PURCHASE MASTER
G1STAT[0-1]
G1PREV[0-1]
G1CMNY
G1BAC
G1ORNO
G1REQ[0-1]
G1PURT[0-1]
G1POOQ[0-1]
G1ORDT
G1VND#
G1OVND[0-1]
G1ORIG[0-1]
G1PFLG
G1PNUM[0-1]
G1METH[0-1]
G1DMDE[0-1]
G1LNCD[0-1]
G1DELT
G1DTDE[0-1]
G1PAYT
G1PTDE[0-1]
G1VADD[0-1]
G1WHSE[0-1]
G1BILL[0-1]
G1SHIP[0-1]
G1COM[0-1]
G1CURR[0-1]
G1DDDT[0-1]
G1BILD[0-1]
G1DESD[0-1]
G1CLDT[0-1]
G1CFLG[0-1]
G1CDAY[0-1]
G1MPOC
G1CONT[0-1]
G1STDV[0-1]
G1VNDV[0-1]
G1LOCV[0-1]
G1STDC[0-1]
G1ORTY
G1CPA
G1APPL[0-1]
G1LADT
G1LATM
G1USER
G1PROG
id: G1CMNY G1BAC G1ORNO
ref: G1CMNY G1VND#
ref: G1CMNY G1VND# G1VADD
ref: G1CMNY G1WHSE
ref: G1LNCD G1DELT
ref: G1LNCD G1METH
ref: G1CMNY G1VND# G1VADD
ref: G1PAYT G1LNCD
ref: G1CMNY G1STDC
ref: G1LNCD
ref: G1CURR
gr: G1BAC G1ORTY
gr: G1LNCD G1CURR G1CMNY G1VND# G1METH
gr: G1WHSE G1BILL G1SHIP
gr: G1LNCD G1DELT G1DTDE
gr: G1CURR G1VND# G1METH G1DMDE
gr: G1LNCD G1PAYT G1PTDE
gr: G1MPOC G1VADD

BUYER ANALYST
ZBAC
ZDESCR
ZORTY
ZAPPRO
id: ZBAC

PART MASTER
AACTCD
ACMPNY
APARTL
APARTC
APART
APARTS
ADESC5
AQCSTT[0-1]
APACK[0-1]
AVOLUM[0-1]
ALOLEV[0-1]
APSRUN[0-1]
APSCNT[0-1]
ASECUR[0-1]
ADRAW[0-1]
AUNMS[0-1]
ADESC[0-1]
ACOMVT[0-1]
ATYPE[0-1]
ACOMM[0-1]
AACCT[0-1]
AACTNO[0-1]
APGACT[0-1]
AVALUE[0-1]
ACORPV[0-1]
ASSUER[0-1]
AOPCD[0-1]
AOPQT[0-1]
ABAC[0-1]
AMTCLS[0-1]
ASUFIX[0-1]
ASEGRD[0-1]
ABATEK[0-1]
ADEPT[0-1]
AWHSE[0-1]
ASETCS[0-1]
AVENDR[0-1]
APLLOT[0-1]
AWSTDR[0-1]
APOQTY[0-1]
APRSTC[0-1]
ASURCS[0-1]
ASURDC[0-1]
APRCTC[0-1]
APLTAN[0-1]
AUSEUM[0-1]
ABUYUM[0-1]
ASELUM[0-1]
APRUM[0-1]
AUSCU[0-1]
ABYCUM[0-1]
ASLCUM[0-1]
ASHPWT[0-1]
AWGHT[0-1]
ACUBE[0-1]
APPLG[0-1]
ANOTE1[0-1]
ANOTE2[0-1]
AVAT[0-1]
ASSP[0-1]
ASSSP[0-1]
AORIG[0-1]
ASPRTL[0-1]
ASPRTC[0-1]
ASPART[0-1]
ASPRTS[0-1]
ABLC[0-1]
ALMDAT[0-1]
AMTIME[0-1]
ALUSER[0-1]
APRDTP[0-1]
AMENE[0-1]
ACHAR1[0-1]
ACHAR2[0-1]
ACHAR3[0-1]
id: ACMPNY APARTL APARTC APART APARTS
ref: ABUYUM
ref: APLTAN
ref: ABAC
gr: AOPCD AOPQT
gr: ACMPNY AVENDR

SUPPLIER APPROBATION
PLAI01
PCOM01
PBAS01
PSUP01
SUPL01
ADDR01[0-1]
STAT01
DATA01[0-1]
DATE01[0-1]
DATC01[0-1]
DAT301[0-1]
DACC01[0-1]
SIT101[0-1]
PDES01
VNAM01[0-1]
VSEP01[0-1]
USER01
MDAT01
TIME01
WSTN01
PGMN01
id: PLAI01 PCOM01 PBAS01 PSUP01 SUPL01
id': PLAI01 PCOM01 PBAS01 PSUP01 SUPL01 ADDR01 STAT01
id': SUPL01 ADDR01 STAT01 PLAI01 PCOM01 PBAS01 PSUP01
ref: STAT01
gr: PLAI01 PCOM01 PBAS01 PSUP01 PDES01
gr: STAT01 DATE01

STATUS CODE
ACTI08
STAT08
DESC08
AUDI08
PGMN08
USED08
WSTN08
MDAT08
MTIM08
id: STAT08

CALENDAR
LCALDT
LBAXDT
LPROD[0-1]
LBAX[0-1]
LBAXY[0-1]
LPLAN[0-1]
LPCLOS[0-1]
LCOMNT[0-1]
LUMDTE
LUMTME
LUMUSR
id: LCALDT
id': LBAXDT LCALDT
gr: LBAXY LBAX

DEAL QTY/PRICE
QPCMNY
QPPRTL
QPPRTC
QPPART
QPPRTS
QPCMPR
QPVEND
QPCURR
QPQTY
QPDISCV
QPPRIC
QPDISP
QPAST
QPSTAT
QPPRIF
QPTOEF
QPAPDT
QPAPUS
QPDEDT
QPDELS
QPDATE
QPTIME
QPUSER

HEADER COMMENT
QBCMNY
QBSEQ
QBPCD
QBCOM
QBDATE
QBTIME
QBUSER

PRICE DEAL HEADER
HDSTAT
HDPRTL
HDPRTC
HDPART
HDPRTS
HDCMPR
HDRANK
HDSLED
HDDLED
HDCONT
HDSOP
HDDTP
HDDATE
HDTIME
HDUSER
HDCMNY
HDVEND
id: HDCMNY HDVEND HDPART HDPRTS HDPRTL HDPRTC
gr: HDCMNY HDVEND

VENDOR ADDRESS
G4CMNY
G4VEND
G4CODE
G4SEQ
G4ADD1
G4ADD2[0-1]
G4ADD3[0-1]
G4ADD4[0-1]
G4ADD5[0-1]
G4CONT[0-1]
G4MPOC
G4VOLT[0-1]
G4PCOD[0-1]
G4PHAR[0-1]
G4PHON[0-1]
G4TEAR[0-1]
G4TELX[0-1]
G4PCOD[0-1]
G4FXAR[0-1]
G4FAX[0-1]
G4LADT
G4LATM
G4USER
G4VNDN
id: G4CMNY G4VEND G4CODE
id': G4CMNY G4VEND G4CODE G4SEQ
ref: G4CMNY G4VEND
gr: G4CMNY G4VEND G4VNDN
gr: G4PCOD G4PHAR G4PHON
gr: G4PCOD G4FXAR G4FAX
gr: G4PHAR G4PHON
gr: G4PCOD G4FXAR G4FAX
gr: G4TEAR G4TELX
gr: G4MPOC G4PHAR G4PHON
gr: G4MPOC G4FXAR G4FAX
gr: G4MPOC G4TEAR G4TELX

CURRENCY
ZCURCOD
ZDESC1
ZDESC2
ZCONVF
ZCDATE
ZCTIME
ZCUSER
id: ZCURCOD

WAREHOUSE
Z1CMNY
Z1WHSE
Z1DESC
Z1ADD1[0-1]
Z1ADD2[0-1]
Z1ADD3[0-1]
Z1CSDE
Z1INTP
Z1DEPT[0-1]
Z1RSDY
ZLADT
ZLATM
ZLUSER
id: Z1CMNY Z1WHSE

DELIVERY TERMS
ZTRMCD
ZTRMLG
ZTRMDE
ZLADT
ZLATM
ZLUSER
ZDELCD
id: ZTRMCD ZTRMLG
id': ZDELCD ZTRMLG
ref: ZTRMLG

PAYMENT TERMS
ZPAYCD
ZDAYS
ZPAYLG
ZPAYDE
ZLADT
ZLATM
ZLUSER
id: ZPAYCD ZPAYLG
ref: ZPAYLG

LANGUAGE CODE
ZRCODE
ZRDESC
ZRLADT
ZRLATM
ZRUSER
id: ZRCODE

DELIVERY METHOD
ZMTHCD
ZLNGCD
ZDESC
ZENSU
ZLADT
ZLATM
ZUSER
id: ZMTHCD ZLNGCD
id': ZLNGCD ZMTHCD
ref: ZLNGCD

COUNTER
ZCMPNY
ZTYPE
ZMSGN
ZNEXT
ZEND
ZLAST
ZDESC
ZLADT
ZLATM
ZLUSER
id: ZCMPNY ZTYPE
ref: ZCMPNY
gr: ZMSGN ZEND

COMPANY
ZCMPNY
ZNAME
ZADD1[0-1]
ZADD2[0-1]
ZADD3[0-1]
ZSOLD[0-1]
ZOLAC
ZLADT
ZLATM
ZLUSER
id: ZCMPNY

VENDOR COMMENT
QASTAT[0-1]
QACMNY
QAVEND
QASEQ
QAPCD[0-1]
QACOM
QADATE
QATIME
QAUSER
ref: QACMNY QAVEND

PAYMENT CODES
CMNY03
ACTI03
PAY203
DESC03
PGMN03
LMDA03
LMTM03
LUSR03
WRES03
id: CMNY03 PAY203

VENDOR IDENTIFICATION
G3STAT[0-1]
G3CMNY
G3VND#
G3TYPE[0-1]
G3VNDN
G3ADD1
G3ADD2[0-1]
G3ADD3[0-1]
G3ADD4[0-1]
G3ADD5[0-1]
G3PAY[0-1]
G3ATN[0-1]
G3CUR
G3PAYT
G3DLMT
G3SPLT[0-1]
G3LOCD
G3DELT
G3MPOC
G3PCOD[0-1]
G3PHAR[0-1]
G3PHON[0-1]
G3TEAR[0-1]
G3TELX[0-1]
G3PCOD[0-1]
G3FXAR[0-1]
G3FAX[0-1]
G3CORP
G3CONT[0-1]
G3WHSE[0-1]
G3DESC[0-1]
G3CDAY[0-1]
G3BANE[0-1]
G3VAT[0-1]
G3CVAT
G3ADR1[0-1]
G3ADR2[0-1]
G3ADR3[0-1]
G3SRCF[0-1]
G3APPL[0-1]
G3PAF2
G3UPDE[0-1]
G3SURC
G3LADT
G3LATM
G3USER
id: G3CMNY G3VND#
ref: G3CMNY G3WHSE
ref: G3DELT G3LOCD
ref: G3PAYT G3LOCD
ref: G3LOCD
ref: G3DLMT G3LOCD
ref: G3CMNY G3PAF2
ref: G3CUR
gr: G3PCOD G3PHAR G3PHON
gr: G3PCOD G3FXAR G3FAX
gr: G3PHAR G3PHON
gr: G3FXAR G3FAX
gr: G3TEAR G3TELX
gr: G3MPOC G3PHAR G3PHON
gr: G3MPOC G3FXAR G3FAX
gr: G3MPOC G3TEAR G3TELX
gr: G3CVAT G3VAT
gr: G3SRCF G3ADR1 G3ADR2 G3ADR3
gr: G3CMNY G3VND# G3LOCD

PURCHASE CONTROL FILE
CMNY06
RECS06
BAC006
ORD006
PGMN06
LMDA06
LMTM06
LUSR06
WRES06
ref: CMNY06 BAC006 ORD006

PURCHASE COMMENT
G8STAT
G8PREV
G8CMNY
G8TYPE
G8BAC
G8ORNO
G8LINE
G8RSEQ
G8COMM
G8COMD
G8WRT
G8LADT
G8LATM
G8USER
ref: G8CMNY G8BAC G8ORNO

STANDARD COMMENTS
CMNY02
ACTI02[0-1]
CODE02
SHOR02
COMM02[0-1]
LINE02
PGMN02
LMDA02
LMTM02
LUSR02
WRES02
id: CMNY02 CODE02

CURRENCY CONVERSION TABLE
CPLTA
CSECTA
DUPDTA
CSSTA
P
KEY1
TSZTAB
TSCUR
TSYY
TSMM
?
KEY2
TSZTAB
TSKEY
?
P
P
DATA1
?
DESC
DATA2
TSSEP
??
TSCONT
TSCONS
TSCOMB
?2

VENDOR/PART COMMENTS
QCSTAT
QCCMNY
QCPRTL
QCPRTC
QCPART
QCPRTS
QCCMPR
QCVEND
QCSEQ
QCDATE
QCTIME
QCUSER
id: QCCMNY QCPART QCPRTS QCPRTL QCPRTC QCVEND QCSEQ
id': QCCMNY QCPART QCPRTS QCPRTL QCPRTC QCVEND QCSEQ

Annexe 5 : le schéma simplifié

Beugum-Maita/Simplified-(gra)

PURCHASE ITEM
Status[0-1]
Previous status[0-1]
Company code
Buyer analyst code
Order number
Order line number
Requisition number[0-1]
Purchase order type
Requisition line number[0-1]
Part code - Label[0-1]
Part code - Country[0-1]
Part code - Basic[0-1]
Part code - Suffix[0-1]
Part code - Compressed[0-1]
Non-stock part[0-1]
Comment flag[0-1]
Allocation flag[0-1]
Department code[0-1]
Order date[0-1]
Requirement date[0-1]
Delivery date[0-1]
Despatch date[0-1]
Order qty in stock uom[0-1]
Initial order qty in stock uom[0-1]
Tot. balance due in stck uom[0-1]
Price flag[0-1]
Date flag[0-1]
Change flag[0-1]
Receipt consumption flag[0-1]
Stocking uom[0-1]
Buying uom[0-1]
Last used unit of measure
Tot. qty rec in stock uom[0-1]
Local decimal ctrl[0-1]
Vendor decimal control[0-1]
Vendor stock uom price[0-1]
Local stock uom price[0-1]
Vendor buying uom price[0-1]
Local buying uom price[0-1]
Vendor curr stock discount[0-1]
Local curr stock discount[0-1]
Vendor curr buy discount[0-1]
Local curr buying discount[0-1]
Currency rate[0-1]
Current sequence 1[0-1]
Advice time printed[0-1]
Recall time printed[0-1]
MRO transfer date[0-1]
Advice date printed[0-1]
Recall date printed[0-1]
Advice type[0-1]
Advice status[0-1]
Recall status[0-1]
Curr order stock qty 1[0-1]
Curr order stock qty 2[0-1]
Curr order stock qty 3[0-1]
Qty on dock in stock uom[0-1]
Qty on dock in buying uom[0-1]
Curr tot rec stock 3[0-1]
Curr stock bal due 1[0-1]
Curr stock bal due 2[0-1]
Curr stock bal due 3[0-1]
Matched rec qty-stock uom[0-1]
Matched receipt qty-buy uom[0-1]
Order qty - buy uom[0-1]
Init. order qty - buying uom[0-1]
Tot. bal due - buy uom[0-1]
Tot. qty rec in buy uom[0-1]
Curr buy uom qty 1[0-1]
Curr buy uom qty 2[0-1]
Curr buy uom qty 3[0-1]
Curr tot rec buy 1[0-1]
Curr tot rec buy 2[0-1]
Curr tot rec buy 3[0-1]
Curr buy uom bal due 1[0-1]
Curr buy uom bal due 2[0-1]
Curr buy uom bal due 3[0-1]
Last receipt date[0-1]
Requisition 1[0-1]
Requisition 2[0-1]
Requisition 3[0-1]
Close date[0-1]
Account number[0-1]
Contract number[0-1]
Vendor discount[0-1]
Part discount[0-1]
Print price flag[0-1]
VAT code[0-1]
MRO request number[0-1]
CPA number[0-1]
Asset number[0-1]
MRO family[0-1]
MRO group[0-1]
MRO sequence[0-1]
id Buyer analyst code
Company code
Order number
Order line number
ref Last used unit of measure
ref Company code
Buyer analyst code
Order number
gr Buyer analyst code
Company code
Order number
Non-stock part
Part code - Compressed
MRO family
MRO group
MRO sequence
MRO request number
gr Buyer analyst code
Company code
Order number
Account number
CPA number
Part code - Compressed
Part code - Label
Part code - Country
Part code - Basic
Part code - Suffix
Contract number
Order qty in stock uom
Order qty - buy uom
Tot. balance
gr Non-stock part
Buyer analyst code
Company code
Order number
gr Buyer analyst code
Company code
Order number
Part code - Compressed
Part code - Basic
Part code - Suffix
Part code - Label
Part code - Country
gr Buyer analyst code
Company code
Order number
Non-stock part
Part code - Basic
Part code - Suffix
Part code - Label
Part code - Country
Contract number
Tot. balance due in stck uom
Order qty in stock uom
Tot. qty rec in stock uom
Stocking u
gr Buyer analyst code
Company code
Order number
MRO family
MRO group
MRO sequence
Part code - Compressed

VAT CONTROL
VAT code
Percentage
id VAT code

PART CROSS REFERENCE
Company code
Part code - label
Part code - Country
Part code - Basic
Part code - Suffix
Part code - Compressed
id Company code
Part code - Compressed
id' Company code
Part code - Basic
Part code - Suffix
Part code - label
Part code - Country

SUPPLIER APPROBATION
Part number - Label
Part number - Country
Part number - Basic
Part number - Suffix
Vendor number
Vendor address code[0-1]
Status code
Approbation date[0-1]
Last evaluation date[0-1]
Certification date[0-1]
Integration date[0-1]
Acceptation date[0-1]
Site of production[0-1]
Vendor name[0-1]
Vendor product reference[0-1]
id Part number - Label
Part number - Country
Part number - Basic
Part number - Suffix
Vendor number
ref Status code
gr Part number - Label
Part number - Country
Part number - Basic
Part number - Suffix
gr Status code
Integration date

STATUS CODE
Description
Audit delay

NON D-STATUS CODE
Activity code
Status code
id Status code

D-STATUS CODE
Status code

PART MASTER
Activity code
Company code
Part code - Label
Part code - Country
Part code - Basic
Part code - Suffix
Part code - Compressed
QC receipt status[0-1]
Packing actor[0-1]
Volume[0-1]
Low level code[0-1]
Product run activity number[0-1]
Number of comps in structure[0-1]
Part security code[0-1]
Drawing number[0-1]
Unit of measure[0-1]
Description[0-1]
Comment[0-1]
Type[0-1]
Commodity code[0-1]
Accounting code[0-1]
Accounting number[0-1]
Flag for accounting[0-1]
Value class[0-1]
Corporate value class[0-1]
Issue code[0-1]
Order policy code[0-1]
Order policy qty[0-1]
Default buyer[0-1]
Material class[0-1]
Suffix tracking flag[0-1]
Kegrind flag[0-1]
Batch tracking flag[0-1]
Department code[0-1]
Warehouse code[0-1]
Setup cost[0-1]
Vendor number[0-1]
Flow vs lot code[0-1]
Wet or dry flag[0-1]
Purchase order qty[0-1]
Standard cost[0-1]
European unit cost[0-1]
European cost decimal ctrl[0-1]
Current cost[0-1]
Plant analyst[0-1]
Usage of uom[0-1]
Buying uom[0-1]
Selling uom[0-1]
Previous uom[0-1]
usage to stock uom conv [0-1]
Buying to stock uom conv [0-1]
Selling to stock uom conv [0-1]
Shipping weight[0-1]
Weight / 1000[0-1]
Cubic inches by piece[0-1]
Part number activity flag[0-1]
Standard comment code[0-1]
Standard comment code2[0-1]
VAT code[0-1]
Source of supply flag[0-1]
Sec. source of supply flag[0-1]
Origin code[0-1]
Subst. part number - Label[0-1]
Subst. part number - Country[0-1]
Subst. part number - Basic[0-1]
Subst. part number - suffix[0-1]
Basic language code[0-1]
Product type[0-1]
Min qty for orders on call[0-1]
Description1[0-1]
Description2[0-1]
Description3[0-1]
id Company code
Part code - Label
Part code - Country
Part code - Basic
Part code - Suffix
ref Buying uom
ref Plant analyst
ref Default buyer
gr Order policy code
Order policy qty
gr Company code
Vendor number

DEAL QTY/PRICE
Company code
Part code - Label
Part code - Country
Part code - Basic
Part code - Suffix
Part code - Compressed
Vendor number
Currency code
Quantity
Vendor decimal control
Price
Discount percent
Recomm. order
Approbation status
From effective date
To effective date
Approbation date
Approved by user
Deactivated date
Deactivated by user

CALENDAR
Date
Banker date
Beginning Banker month flag[0-1]
Beginning Banker year flag[0-1]
Plan date flag[0-1]
Period closed flag[0-1]
Comment[0-1]
id Banker date
Date
gr Beginning Banker year flag
Beginning Banker month flag

NON PROD DATE CAL.
Date
id Date
gr Date

PROD DATE CAL.
Non production date flag[0-1]

UNIT OF MEASURE
Unit of measure
Description
id Unit of measure

PURCHASE MASTER
Status[0-1]
Previous status[0-1]
Company code
Buyer analyst code
Order number
Requisition number[0-1]
Type
Number generation flag[0-1]
Order date
Vendor number
Original vendor number[0-1]
Origination[0-1]
Print flag
Number times printed[0-1]
Delivery method code[0-1]
Language code[0-1]
Delivery terms code
Payment terms code
Vendor address code[0-1]
One time vendor sequence[0-1]
Warehouse code[0-1]
Bill - to customer number[0-1]
Ship - to customer number[0-1]
Comment flag[0-1]
Currency code[0-1]
Original due - date[0-1]
Delivery date[0-1]
Despatch date[0-1]
Order close date[0-1]
Confirmation req. flag[0-1]
nr of days to follow confirm.[0-1]
Communic. method
Contract number[0-1]
Standard PO tot. value[0-1]
Vendor PO tot. value[0-1]
Local PO tot. value[0-1]
Standard comment code[0-1]
Order type
CPA flag
Appl.[0-1]
id Company code
Buyer analyst code
Order number
ref Company code
Warehouse code
ref Company code
Standard comment code
ref Currency code
ref Company code
Vendor number
Vendor address code
ref Delivery method code
Language code
ref Delivery terms code
Language code
ref Payment terms code
Language code
ref Company code
Vendor number
ref Buyer analyst code
gr Language code
Currency code
Company code
Vendor number
Delivery method code
gr Warehouse code
Bill - to customer number
Ship - to customer number
gr Currency code
Vendor number
Delivery method code
Vendor address code
gr Communic. method
Vendor address code
gr Buyer analyst code
Order type

BUYER ANALYST
Buyer analyst code
Description
Order type
Approbation flag
id Buyer analyst code

CURRENCY
Currency code
Description
Short description
Conversion factor
id Currency code

WAREHOUSE
Company code
Warehouse code
Description
Address1[0-1]
Address2[0-1]
Address3[0-1]
Consigned inventory
In-transit inventory type
Default department code[0-1]
Nr days to set req. ship date
id Company code
Warehouse code

DELIVERY TERMS
Delivery terms code
Delivery terms language
Delivery terms description
New delivery terms code
id Delivery terms code
Delivery terms language
id' New delivery terms code
Delivery terms language
ref Delivery terms language

PAYMENT TERMS
Payment terms code
# of days
Language code
Description
id Payment terms code
Language code
ref Language code

LANGUAGE CODE
Language code
Description
id Language code

VENDOR ADDRESS
Company code
Vendor number
Vendor address code
Sequence number
Address1
Address2[0-1]
Address3[0-1]
Address4[0-1]
Address5[0-1]
Customer contract[0-1]
Communic. method
Vendor lead time[0-1]
Phone country code[0-1]
Phone area[0-1]
Phone number[0-1]
Telex area[0-1]
Telex number[0-1]
Fax country code[0-1]
Fax area[0-1]
Fax number[0-1]
id Company code
Vendor number
Vendor address code
ref Company code
Vendor number
gr Phone country code
Phone area
Phone number
gr Fax country code
Fax area
Fax number
gr Phone area
Phone number
gr Fax area
Fax number
gr Telex area
Telex number
gr Communic. method
Phone area
Phone number
gr Communic. method
Fax area
Fax number
gr Communic. method
Telex area
Telex number

HEADER COMMENT
Company code
Sequence number
Print code
Print comment

PRICE DEAL HEADER
Status
Company code
Part code - Label
Part code - Country
Part code - Basic
Part code - Suffix
Part code - Compressed
Vendor number
Vendor ranking
Supplier lead time
Delivery lead time
Contract number
Handling charge percent
Duties percent
id Company code
Vendor number
Part code - Basic
Part code - Suffix
Part code - Label
Part code - Country
gr Company code
Vendor number

VENDOR IDENTIFICATION
Status[0-1]
Company code
Vendor number
Vendor type[0-1]
Vendor name
Address1
Address2[0-1]
Address3[0-1]
Address4[0-1]
Address5[0-1]
Payment flag[0-1]
contact name[0-1]
Currency code
Payment terms code
Delivery method code
Vendor lead time[0-1]
Language code
Delivery terms code
Communic. method
Phone country code[0-1]
Phone area[0-1]
Phone number[0-1]
Telex area[0-1]
Telex number[0-1]
Fax country code[0-1]
Fax area[0-1]
Fax number[0-1]
Corp. flag
Contract flag[0-1]
Warehouse code[0-1]
Discount[0-1]
Nr of days to follow confirm[0-1]
Bank number[0-1]
VAT number[0-1]
VAT code
Foreign address 1[0-1]
Foreign address 2[0-1]
Foreign address 3[0-1]
Foreign bank number[0-1]
IBLC code
Payment code - part 1[0-1]
International affiliate code
Update flag[0-1]
European country code
id Company code
Vendor number
ref Company code
Warehouse code
ref Delivery terms code
Language code
ref Payment terms code
Language code
ref Delivery method code
Language code
ref Company code
International affiliate code
ref Currency code
gr Phone country code
Phone area
Phone number
gr Fax country code
Fax area
Fax number
gr Phone area
Phone number
gr Fax area
Fax number
gr Telex area
Telex number
gr Communic. method
Phone area
Phone number
gr Communic. method
Fax area
Fax number
gr Communic. method
Telex area
Telex number
gr VAT code
VAT number
gr Foreign bank number
Foreign address 1
Foreign address 2
Foreign address 3
gr Company code
Vendor number
Language code

CURRENCY CONVERSION TABLE
File code
Register code
Date of last update
Flag TDI

KEY1
TSCTAB
TSCUR
TSYY
TSMM
?

KEY2
TSCTAB
TSKEY
?

DATA1
?
Description

DATA2
TSREF
?1
TSCONT
TSCONS
TSCONB
Y2

DELMETHLEADTIME
Activity code
Company code
Vendor number
Delivery method code
Language code
Lead time
id Vendor number
Delivery method code
Language code

DELIVERY METHOD
Delivery method code
Language code
Description
Insurance
id Delivery method code
Language code
ref Language code

COUNTER
Company code
Counter type
Beginning number
Next available number
Ending number
Last period ending number
Description
id Company code
Counter type
ref Company code
gr Beginning number
Ending number

COMPANY
Company code
Name
Address1[0-1]
Address2[0-1]
Address3[0-1]
Sold to code[0-1]
Accounting code
id Company code

VENDOR/PART COMMENTS
Status
Company code
Part code - Label
Part code - Country
Part code - Basic
Part code - Suffix
Part code - Compressed
Vendor number
Sequence number
Print code
Print comment
id Company code
Part code - Basic
Part code - Suffix
Part code - Label
Part code - Country
Vendor number
Sequence number

VENDOR COMMENT
activity code[0-1]
Company code
Vendor number
Sequence number
Print code[0-1]
Print comment
ref Company code
Vendor number

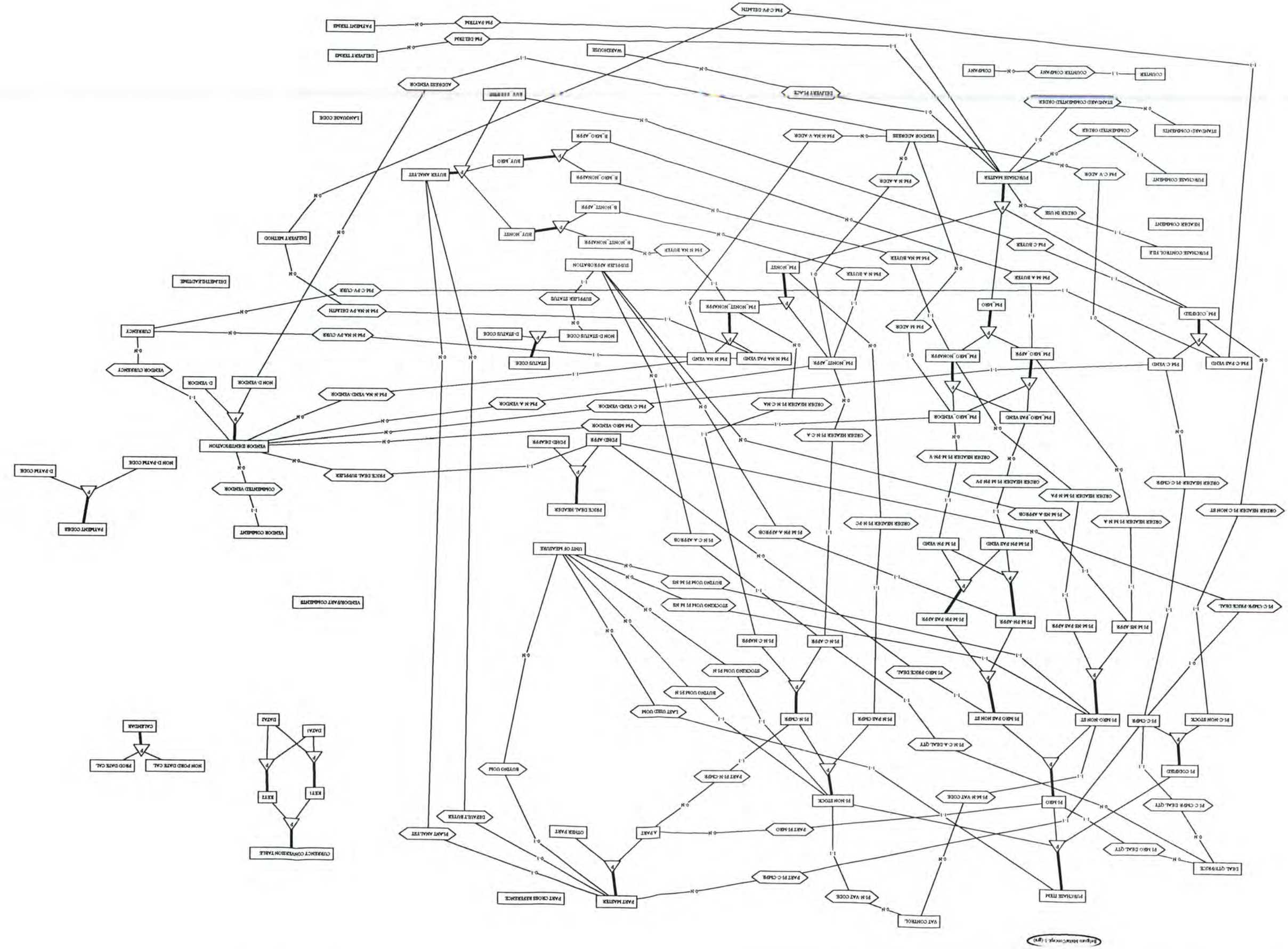
PAYMENT CODES
Company code
Activity code
International affiliate code
Description
id Company code
International affiliate code

PURCHASE CONTROL FILE
Company code
Receipt number
Buyer analyst code
Order number
ref Company code
Buyer analyst code
Order number

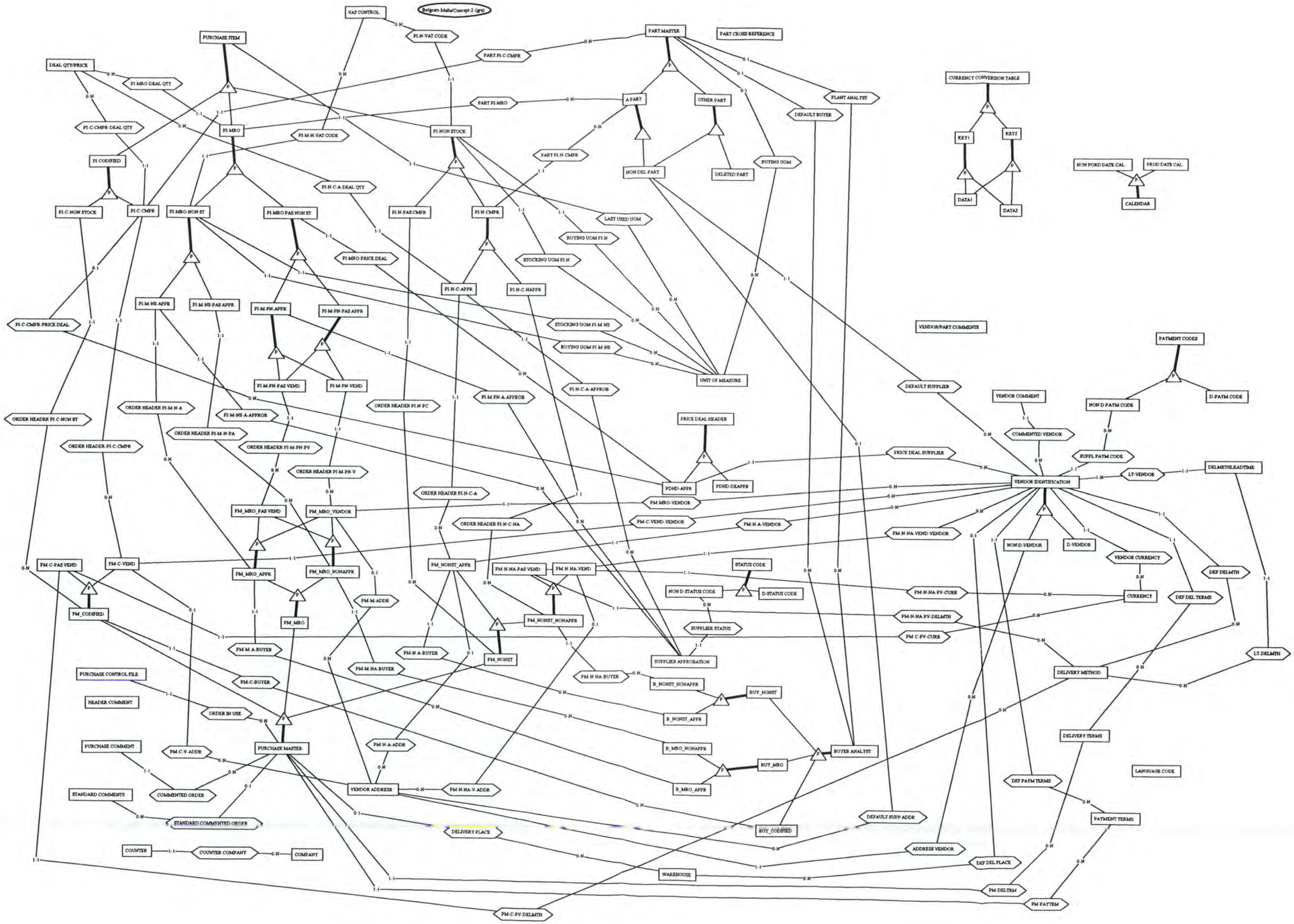
PURCHASE COMMENT
Status
Previous status
Company code
Comment type
Buyer analyst code
Order number
Order line number
Requisition number
Comment line number
Comment text
Print flag
ref Company code
Buyer analyst code
Order number

STANDARD COMMENTS
Company code
Unused flag[0-1]
Standard comment code
Short description
Comment description[0-1]
Comment line number
id Company code
Standard comment code

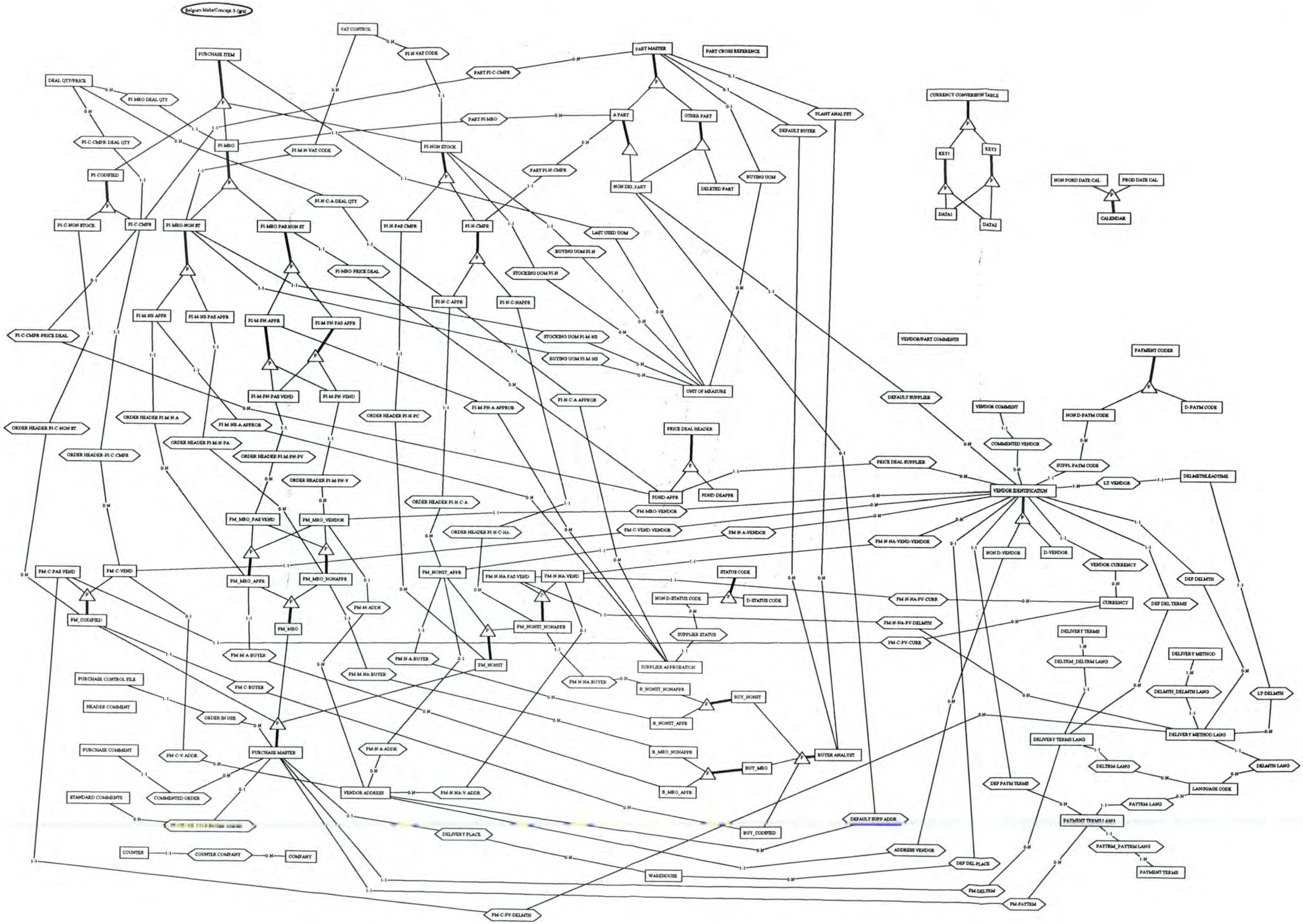
Annexe 6 : le schéma conceptuel (1)


PM_C-PV_DELATN
PM_PATTRN
PATIENT TRANS
PM_DELTRN
DELIVERY TRANS
WAREHOUSE
ADDRESS VENDOR
DELIVERY PLACE
COUNTER
COUNTER COMPANY
COMPANY
LANGUAGE CODE
BUY_JOB
STANDARD COMPANIES
STANDARD COMPANIED ORDER
COMPANIED ORDER
PURCHASE COMPANY
VENDOR ADDRESS
PM_N-NA-V_ADDR
PM_C-V_ADDR
PM_N-A_ADDR
PURCHASE MASTER
ORDER IN USE
HEADER COMPANY
PURCHASE CONTROL FILE
B_JOB1_APPR
BUY_JOB1
BUYER ANALYST
B_JOB1_NONAPPR
B_NONST_APPR
BUY_NONST
B_NONST_NONAPPR
PM_N-NA-BUYER
PM_M-NA-BUYER
PM_N-A-BUYER
PM_M-A-BUYER
PM_C-BUYER
PM_NONST
SUPPLIER APPROBATION
DELIVERY METHOD
DELAETRTLAVTIME
PM_C-PV-CURR
PM_N-NA-PV-DELATN
PM_N-NA-PV-CURR
SUPPLIER STATUS
NON D-STATUS CODE
D-STATUS CODE
STATUS CODE
PM_N-NA-PAS VEND
PM_N-NA-VEND
PM_NONST_NONAPPR
PM_NONST_APPR
PM_JOB1
PM_M-ADDR
PM_JOB1_NONAPPR
PM_JOB1_APPR
PM_COMPANIED
PM_C-PAS VEND
PM_C-VEND
CURRENCY
VENDOR CURRENCY
D_VENDOR
NON D_VENDOR
PM_N-NA_VEND_VENDOR
PM_N-A_VENDOR
PM_C_VEND_VENDOR
ORDER HEADER PI_N-C_NA
PM_JOB1_VENDOR
PM_JOB1_PAS VEND
VENDOR IDENTIFICATION
PM_JOB1_PAS VEND
ORDER HEADER PI_N-C_A
ORDER HEADER PI_M-PM_V
PRICE DEAL_SUPPLIER
PRICE_APPR
PRICE_DEAPPR
ORDER HEADER PI_M PM_PV
ORDER HEADER PI_M_N_PA
ORDER HEADER PI_C-CMPR
ORDER HEADER PI_M-A_APPROB
ORDER HEADER PI_M_N_A
ORDER HEADER PI_C-NON ST
ORDER HEADER PI_N_PC
PI_M_PM_A_APPROB
PI_N_C_A_APPROB
PI_M_PM_VEND
PI_M_PM_PAS VEND
COMPANIED VENDOR
VENDOR COMPANY
PATMENT CODES
D_PATM CODE
NON D_PATM CODE
UNIT OF MEASURE
VENDOR/PART COMPANIES
BUYING UOM PI_M_NS
STOCKING UOM PI_M_NS
PI_M_PM_PAS APPR
PI_M_PM_APPR
PI_M_NS_PAS APPR
PI_M_NS_APPR
PI_C_CMPR_PRICE DEAL
PI_N_C_NAPPR
PI_N_C_APPR
PI_M_JOB1_PRICE DEAL
STOCKING UOM PI_N
BUYING UOM PI_N
LAST USED UOM
PI_N_CMPR
PI_N_PAS CMPR
PI_M_JOB1_PAS NON ST
PI_M_JOB1_NON ST
PI_C_CMPR
PI_C_NON STOCK
PI_N_C_A_DEAL_QTY
PI_C_CODED1ED
BUYING UOM
PART PI_N_CMPR
PI_M_N_VAT CODE
PI_M_JOB1
PI_C_CMPR_DEAL_QTY
PI_NON STOCK
PART PI_M_JOB1
PI_M_JOB1_DEAL_QTY
DEFAULT BUYER
PLANT ANALYST
OTHER PART
A PART
PART MASTER
PART CROSS REFERENCE
PART PI_C_CMPR
PI_N_VAT CODE
VAT CONTROL
PURCHASE ITEM
DEAL_QTY/PRICE
CALENDAR
NON POED DATE CAL
POED DATE CAL
DATA1
DATA2
KEY1
KEY2
CURRENCY CONVERSION TABLE

Annexe 7 : le schéma conceptuel (2)


Belgium-Malta/Concept 2 (grs)
VAT CONTROL
PI-N-VAT CODE
PURCHASE ITEM
PART MASTER
PART CROSS REFERENCE
DEAL QTY/PRICE
PART PI-C-CMPR
PI-MRO-DEAL QTY
PART PI-MRO
A PART
OTHER PART
PLANT ANALYST
DEFAULT BUYER
PI-C-CMPR-DEAL QTY
PI-MRO
PI-M-M-VAT CODE
PI-NON STOCK
PART PI-N-CMPR
BUYING UOM
PI-CODIFIED
NON DEL PART
DELETED PART
PI-N-C-A-DEAL QTY
PI-C-NON STOCK
PI-C-CMPR
PI-MRO-NON ST
PI-MRO PAS NON ST
PI-N-PAS CMPR
PI-N-CMPR
LAST USED UOM
BUYING UOM PI-N
PI-MRO-PRICE DEAL
STOCKING UOM PI-N
PI-N-C-APPR
PI-N-C-NAPPR
PI-M-NS-APPR
PI-M-NS-PAS APPR
PI-M-PN-APPR
PI-M-PN-PAS APPR
STOCKING UOM PI-M-NS
BUYING UOM PI-M-NS
UNIT OF MEASURE
PI-C-CMPR-PRICE DEAL
PI-M-PN-PAS VEND
PI-M-PN-VEND
PI-M-PN-A-APPROB
PI-N-C-A-APPROB
ORDER HEADER PI-M-N-A
ORDER HEADER PI-N-PC
ORDER HEADER PI-C-NON ST
PI-M-NS-A-APPROB
ORDER HEADER PI-M-N-PA
ORDER HEADER-PI-C-CMPR
ORDER HEADER PI-M-PN-PV
ORDER HEADER PI-M-PN-V
PRICE DEAL HEADER
PDHD-APPR
PDHD-DEAPPR
ORDER HEADER PI-N-C-A
PM-MRO-VENDOR
PM_MRO_PAS VEND
PM_MRO_VENDOR
ORDER HEADER PI-N-C-NA
PM-C-VEND-VENDOR
PM-N-A-VENDOR
PM-N-NA-VEND-VENDOR
PM-C-PAS VEND
PM-C-VEND
PM_NONST_APPR
PM-N-NA-PAS VEND
PM-N-NA-VEND
STATUS CODE
PM_MRO_APPR
PM_MRO_NONAPPR
NON D-STATUS CODE
D-STATUS CODE
PM-M-ADDR
PM-N-NA-PV-CURR
PM_CODFIED
PM_MRO
PM_NONST_NONAPPR
SUPPLIER STATUS
PM-N-NA-PV-DELMTH
PM-C-PV-CURR
PM-M-A-BUYER
PM-N-A-BUYER
PM_NONST
SUPPLIER APPROBATION
PM-M-NA-BUYER
PURCHASE CONTROL FILE
PM-C-BUYER
PM-N-NA BUYER
B_NONST_NONAPPR
BUY_NONST
HEADER COMMENT
B_NONST_APPR
ORDER IN USE
PURCHASE COMMENT
PM-C-V-ADDR
PURCHASE MASTER
PM-N-A-ADDR
B_MRO_NONAPPR
BUY_MRO
BUYER ANALYST
STANDARD COMMENTS
COMMENTED ORDER
VENDOR ADDRESS
PM-N-NA-V-ADDR
B_MRO_APPR
STANDARD COMMENTED ORDER
BUY_CODFIED
DEFAULT SUPP ADDR
DELIVERY PLACE
COUNTER
COUNTER COMPANY
COMPANY
WAREHOUSE
ADDRESS VENDOR
DEF DEL PLACE
PM-DELTRM
PM-PAYTRM
PM-C-PV-DELMTH
CURRENCY CONVERSION TABLE
KEY1
KEY2
DATA1
DATA2
NON PORD DATE CAL
PROD DATE CAL
CALENDAR
VENDOR/PART COMMENTS
PAYMENT CODES
DEFAULT SUPPLIER
VENDOR COMMENT
NON D-PAYM CODE
D-PAYM CODE
COMMENTED VENDOR
PRICE DEAL SUPPLIER
SUPPL-PAYM CODE
VENDOR IDENTIFICATION
LT-VENDOR
DELMETHLEADTIME
NON D-VENDOR
D-VENDOR
VENDOR CURRENCY
DEF DELMTH
CURRENCY
DEF DEL TERMS
LT-DELMTH
DELIVERY METHOD
DELIVERY TERMS
LANGUAGE CODE
DEF PAYM TERMS
PAYMENT TERMS

Annexe 8 : le schéma conceptuel (3)



VAT CONTROL
PI-N-VAT CODE
PART MASTER
PART CROSS REFERENCE
PURCHASE ITEM
PART PI-C-CMPR
DEAL QTY/PRICE
PI-MRO-DEAL QTY
PART PI-MRO
A PART
OTHER PART
PLANT ANALYST
DEFAULT BUYER
CURRENCY CONVERSION TABLE
KEY1
KEY2
DATA1
DATA2
NON PORD DATE CAL
PROD DATE CAL
CALENDAR
PI-C-CMPR-DEAL QTY
PI-MRO
PI-M-N-VAT CODE
PI-NON STOCK
PI-CODIFIED
PART PI-N-CMPR
NON DEL PART
DELETED PART
BUYING UOM
PI-N-C-A-DEAL QTY
PI-C-NON STOCK
PI-C-CMPR
PI-MRO-NON ST
PI-MRO-PAS NON ST
PI-N-PAS CMPR
PI-N-CMPR
LAST USED UOM
BUYING UOM PI-N
STOCKING UOM PI-N
PI-MRO-PRICE DEAL
PI-N-C-APPR
PI-N-C-NAPPR
PI-M-NS-APPR
PI-M-NS-PAS APPR
PI-M-PN-APPR
PI-M-PN-PAS APPR
STOCKING UOM PI-M-NS
BUYING UOM PI-M-NS
PI-C-CMPR-PRICE DEAL
VENDOR/PART COMMENTS
PAYMENT CODES
UNIT OF MEASURE
PI-M-PN-PAS VEND
PI-M-PN-VEND
PI-M-PN-A-APPROB
PI-N-C-A-APPROB
DEFAULT SUPPLIER
VENDOR COMMENT
NON D-PAYM CODE
D-PAYM CODE
ORDER HEADER PI-M-N-A
PI-M-NS-A-APPROB
ORDER HEADER PI-N-PC
ORDER HEADER PI-C-NON ST
ORDER HEADER PI-M-N-PA
PRICE DEAL HEADER
COMMENTED VENDOR
ORDER HEADER-PI-C-CMPR
ORDER HEADER PI-M-PN-PV
PRICE DEAL SUPPLIER
SUPPL-PAYM CODE
ORDER HEADER PI-M-PN-V
PDHD-APPR
PDHD-DEAPPR
VENDOR IDENTIFICATION
LT-VENDOR
DELMETHLEADTIME
ORDER HEADER PI-N-C-A
PM-MRO-VENDOR
PM_MRO_PAS VEND
PM_MRO_VENDOR
ORDER HEADER PI-N-C-NA
PM-C-VEND-VENDOR
PM-N-A-VENDOR
PM-N-NA-VEND-VENDOR
NON D-VENDOR
D-VENDOR
VENDOR CURRENCY
PM-C-PAS VEND
PM-C-VEND
PM_MRO_APPR
PM_MRO_NONAPPR
PM_NONST_APPR
PM-N-NA-PAS VEND
PM-N-NA-VEND
STATUS CODE
DEF DELMTH
PM-M-ADDR
NON D-STATUS CODE
D-STATUS CODE
PM-N-NA-PV-CURR
CURRENCY
DEF DEL TERMS
PM_CODIFIED
PM_MRO
PM_NONST_NONAPPR
PM-N-NA-PV-DELMTH
SUPPLIER STATUS
PM-C-PV-CURR
DELIVERY TERMS
PM-M-A-BUYER
PM-N-A-BUYER
PM_NONST
SUPPLIER APPROBATION
DELTRM_DELTRM LANG
DELIVERY METHOD
PURCHASE CONTROL FILE
PM-M-NA-BUYER
PM-N-NA BUYER
B_NONST_NONAPPR
DELMTH_DELMTH LANG
LT-DELMTH
PM-C-BUYER
BUY_NONST
HEADER COMMENT
ORDER IN USE
B_NONST_APPR
DELIVERY METHOD LANG
DELIVERY TERMS LANG
PURCHASE COMMENT
PM-C-V-ADDR
PURCHASE MASTER
PM-N-A-ADDR
B_MRO_NONAPPR
BUYER ANALYST
DELTRM-LANG
DELMTH-LANG
BUY_MRO
COMMENTED ORDER
VENDOR ADDRESS
PM-N-NA-V-ADDR
B_MRO_APPR
DEF PAYM TERMS
LANGUAGE CODE
STANDARD COMMENTS
PAYTRM-LANG
DEFAULT SUPP ADDR
DELIVERY PLACE
BUY_CODIFIED
PAYMENT TERMS LANG
ADDRESS VENDOR
PAYTRM_PAYTRM LANG
COUNTER
COUNTER COMPANY
COMPANY
WAREHOUSE
DEF DEL PLACE
PAYMENT TERMS
PM-DELTRM
PM-PAYTRM
PM-C-PV-DELMTH