Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

La gestion de contraintes dans le SGBD MySQL

Bastin, Jean-François

Award date:
2004

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2026



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

Année académique 2003-2004

La gestion de contraintes dans le SGBD MySQL

Jean-François Bastin

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

Résumé

Le système de gestion de bases de données MySQL se répand largement depuis sa création. Cet outil a la particularité d'être disponible gratuitement via Internet.
Le présent mémoire va analyser, de manière complète, les divers aspects du système de gestion de bases de données MySQL. Cette analyse va se focaliser également sur les différentes étapes de la méthodologie de conception d'une base de données MySQL.
Etant donné l'absence de certaines fonctionnalités, dont notamment celle permettant la gestion des clés étrangères, ce mémoire va tenter de proposer une solution pour combler ce manque. La solution présentée utilisera les techniques et les outils les plus adéquats possibles afin de présenter un résultat homogène et facilement utilisable.

Mots clés : Bases de données : conception et implémentation ; Clés étrangères : déclaration, gestion et programmation.

Abstract

The MySQL database management system sucess has increased considerably since its creation. This tool has the characteristic to be available free via Internet.
The present thesis will analyze in a complete way the various aspects of the MySQL DBMS. The analysis will be also focused on the various stages of the methodology of MySQL database design.
Considering the absence of certain functionalities of which in particular that allowing the management of the foreign keys, this report will try to propose a solution to resolve this lack. The presented solution will use the most adapted possible techniques and tools in order to have a homogeneous and easily usable result.

Key words : Databases : design and implementation; Foreign keys : declaration, management and programming.

Avant-Propos

Remerciements

Je tiens à remercier tout d'abord les personnes suivantes :

- Monsieur Jean-Luc Hainaut pour avoir accepté d'être mon promoteur et de m'avoir permis de réaliser ce mémoire dans un domaine passionnant que sont les bases de données.
- Monsieur Jean-Marc Hick pour m'avoir guidé tout au long de la réalisation de ce travail. Cela n'aurait été possible sans sa disponibilité, ses réponses et ses conseils.
- Monsieur Fernand Focant pour m'avoir permis de concilier activités professionnelles et études en cours du soir de façon harmonieuse.

Mes remerciements sont aussi adressés à mes proches pour leur soutien et leur compréhension exprimés tout au long de ces années d'études.

Un grand merci également à mes collègues du Service informatique universitaire des Facultés Universitaires Notre-Dame de la Paix et à mes amis qui, par un encouragement, une contribution ou un conseil m'ont permis de mener à bien ces études et m'ont permis de mettre en œuvre ce mémoire.

Table des matières

Chapitre 1 : Introduction **9**
1.1. Sujet – objectifs 9
1.2. Etat de l'art 9
1.2.1. Introduction 9
1.2.2. Conception d'une base de données 10
1.2.3. Techniques de gestion de contraintes dans les SGBD 11
1.3. Structure du mémoire 13
Chapitre 2 : Description de MySQL **15**
2.1. Introduction 15
2.1.1. Qu'est-ce que MySQL ? 15
2.1.2. Pourquoi choisir MySQL ? 15
2.1.3. Les outils de MySQL 16
2.1.4. Les utilisateurs et les applications typiques de MySQL 16
2.2. Fonctionnement de MySQL 17
2.2.1. Connexion à MySQL 17
2.2.2. Gestion des bases de données 17
2.2.3. Gestion des tables 18
2.2.4. Types de stockage des tables 21
2.2.5. Types de données 23
2.2.6. Types de colonnes 24
2.2.7. Gestion des index 27
2.3. Les fonctionnalités non supportées de MySQL 28
2.3.1. Les clés étrangères et l'intégrité référentielle 28
2.3.2. Les procédures stockées et les déclencheurs 28
2.3.3. Les vues 28
2.3.4. Les sous-requêtes 29
2.3.5. Les transactions et les instructions commit/rollback 29
2.4. Problématique 29
2.5. Comparatif avec d'autres SGBD 31
Chapitre 3 : Méthodologie de développement de bases de données **34**
3.1. Point de départ 35
3.2. L'analyse conceptuelle 35
3.3. La conception logique 36
3.3.1. Le modèle strict 36
3.3.2. Le modèle étendu 36
3.4. La conception physique 37
3.5. Le codage 37
Chapitre 4 : Conception d'une base de données MySQL **38**
4.1. Expression du modèle de données en GER (notion de schéma conforme) 38
4.1.1. Le schéma conceptuel 38
4.1.2. Le schéma logique 41
4.1.3. Le schéma physique 42
4.2. Conception logique (plan de transformation) 44
4.2.1. Transformations 44
4.2.2. Transformations logiques propres à MySQL 45
4.2.3. Conventions des noms 49

4.3. Conception physique (règles et heuristiques) 50
4.3.1. Les index et les clés d'accès 50
4.3.2. Les types de stockage 50
4.4. Codage 50
4.4.1. Les constructions déclarées 50
4.4.2. Constructions exprimées de manière procédurale 52
Chapitre 5 : Développement d'une base de données MySQL avec l'atelier de génie logiciel DB-MAIN 53
5.1. Description de l'atelier de génie logiciel DB-MAIN 53
5.1.1. Présentation générale 53
5.1.2. Particularités 53
5.2. Description du langage de programmation « Voyager » 54
5.3. Analyse conceptuelle 55
5.3.1. Création du schéma conceptuel 55
5.4. Conception logique 56
5.4.1. Paramétrisation des nouveaux types « SET » et « ENUM » dans DB-MAIN 56
5.4.2. Fonctionnalité de génération automatique du modèle relationnel 58
5.4.3. Assistant de transformations globales 59
5.4.4. Assistant de transformations globales avancées 60
5.4.5. Traitement des noms 62
5.5. Conception physique : spécification des paramètres physiques 63
5.5.1. Définition des clés d'accès 63
5.5.2. Paramétrisation du type de stockage des tables 63
5.6. Codage 66
5.6.1. Génération du code DDL 66
5.6.2. Génération du fichier représentant les clés étrangères de la base de données 66
Chapitre 6 : Environnement de programmation MySQL 68
6.1. Introduction à la programmation avec MySQL 68
6.2. Paramètres de développement à prendre en considération 68
6.3. Composants de l'environnement du module de gestion des clés étrangères 69
6.3.1. Librairie de fonctions (DLL) 69
6.3.2. Technologie ODBC 69
6.3.3. Utilisation d'un fichier de type « header file » 69
6.3.4. Développement - langage de programmation C/C++ 70
6.4. Structure d'un programme utilisant le module de gestion des clés étrangères 71
6.4.1. Définition du lien ODBC 71
6.4.2. Compilation de la librairie de fonctions et ajout dans le programme 72
6.4.3. Déclaration des fonctions et des procédures importées 72
6.4.4. Appel des fonctions et des procédures importées 73
Chapitre 7 : Extension de l'atelier DB-MAIN 74
7.1. Le générateur de code DDL de MySQL 74
7.1.1. Module de départ 74
7.1.2. Objectifs 74
7.1.3. Architecture du programme 74
7.1.4. Gestion des types « SET » et « ENUM » 77
7.1.5. Gestion du type de stockage d'une table 78
7.1.6. Gestion des clés étrangères et des contraintes de type « check » 78

7.2. Le générateur du fichier contenant la représentation des clés étrangères 79
7.2.1. Objectif 79
7.2.2. Architecture du programme 79
7.2.3. Exemples de formats de clés générés 83
7.3. La librairie de fonctions relatives à la gestion des clés étrangères 84
7.3.1. Objectifs 84
7.3.2. Traitement « Insertion » 84
7.3.3. Traitement « Suppression » 85
7.3.4. Traitement « Modification – Mise à jour » 85
7.3.5. Architecture 87
Chapitre 8 : Etude de cas 93
8.1. Mise en place d'une base de données MySQL 93
8.1.1. Description du domaine d'application 93
8.1.2. Analyse conceptuelle 94
8.1.3. Conception logique 94
8.1.4. Conception physique 95
8.1.5. Codage 96
8.1.6. Déploiement de la base de données 99
8.2. Gestion des clés étrangères dans un programme 100
8.2.1. Hypothèses 100
8.2.2. Exemple d'une fonction n'utilisant pas la fonction de validation 100
8.2.3. Exemple de fonction utilisant la fonction de validation 101
Chapitre 9 : Conclusion 103
Perspectives et améliorations 106
Bibliographie 107

ANNEXES
Annexe 1 : Le script Voyager relatif au générateur de code DDL de MySQL 110
Annexe 2 : Le script Voyager relatif au générateur du fichier contenant la représentation des clés étrangères 122
Annexe 3 : Le script de la librairie de fonctions de gestion des clés étrangères 128
Annexe 4 : Le script d'un programme faisant appel à la librairie de fonctions de gestion des clés étrangères 154

Table des figures

Figure 2.1. : Clé étrangère reliant deux tables 30
Figure 3.1. : Méthodologie de développement de bases de données 34
Figure 4.1. : Schéma conceptuel d'une base de données 39
Figure 4.2. : Schéma logique relationnel d'une base de données 41
Figure 4.3. : Schéma physique MySQL d'une base de données 43
Figure 4.4. : Transformation d'une liste d'attributs en un seul attribut de type « SET » 45
Figure 4.5. : Transformation d'une liste d'attributs en un seul attribut de type « ENUM » 46
Figure 4.6. : Transformation d'un type d'entités en un attribut de type « SET » 47
Figure 4.7. : Transformation d'un type d'entités en un attribut de type « ENUM » 48
Figure 5.1. : Menu et palette d'outils permettant la phase d'analyse conceptuelle dans DB-MAIN 55
Figure 5.2. : Création des domaines de valeurs « SET » et « ENUM » 56
Figure 5.3. : Déclaration du type « SET » et « ENUM» 57
Figure 5.4. : Création de la liste de valeurs d'un champ de type « SET » 57
Figure 5.5. : Création de la liste de valeurs d'un champ de type « ENUM » 58
Figure 5.6. : Fonctionnalité « Relational model » du menu « Transform » 58
Figure 5.7. : L'assistant de transformations globales 59
Figure 5.8. : L'assistant de transformations globales avancées 60
Figure 5.9. : Fenêtre de définition de règles d'analyse d'objets 61
Figure 5.10. : Fenêtre de définition de fonctions mises en place par l'utilisateur 62
Figure 5.11. : Ecran de traitement des noms 62
Figure 5.12. : Définition des clés d'accès 63
Figure 5.13. : Sélection de la fenêtre des Méta-propriétés 64
Figure 5.14. : Création de la propriété dynamique « type-table » 64
Figure 5.15. : Déclaration de la liste des valeurs prédéfinies reprenant les différents types de stockage des tables MySQL 65
Figure 5.16. : Déclaration d'un type de stockage à une table du schéma physique 65
Figure 5.17. : Appel du générateur de code DDL de MySQL 66
Figure 5.18. : Appel du générateur du fichier représentatif des clés étrangères de la BD 67
Figure 6.1. : L'environnement de programmation MySQL 70
Figure 6.2. : Pilote ODBC pour MySQL installé sur le système d'exploitation 71
Figure 6.3. : Création d'un lien ODBC vers une base de données MySQL 72
Figure 8.1. : Etude de cas : schéma conceptuel 94
Figure 8.2. : Etude de cas : schéma logique 95
Figure 8.3. : Etude de cas : schéma physique 96
Figure 8.4. : Etude de cas : déploiement de la base de données 99
Figure 8.5. : Etude de cas : description des tables de la base de données dans l'environnement de MySQL 99

Chapitre 1 : Introduction

1.1. Sujet – objectifs

Pour la réalisation de ce mémoire de fin d'études, il m'a été proposé d'analyser le système de gestion de bases de données *(SGBD)* MySQL. Un système de gestion de bases de données possède les fonctionnalités principales suivantes : le stockage, la gestion, la mise à jour et la consultation d'un ensemble d'informations. Une telle démarche d'analyse a pour objectif de se focaliser sur plusieurs points importants comme par exemple, l'étude de la conception d'une base de données en fonction du modèle auquel le SGBD appartient et également, la mise en évidence des caractéristiques et des particularités propres à MySQL.
Dans les chapitres qui vont suivre, l'exposé de l'analyse du SGBD MySQL va s'effectuer de manière intégrale. Le processus complet de conception et d'utilisation d'une base de données MySQL va donc être décrit depuis le schéma conceptuel jusqu'à la phase de codage.

Etant donné que MySQL ne gère pas de façon native et explicite les contraintes de type « clés étrangères », une solution permettant la mise en œuvre de cette fonctionnalité manquante a été élaborée. Une clé étrangère relie deux tables entre elles. La colonne ou la combinaison de colonnes faisant partie de l'identifiant de la table de référence est ajoutée dans l'autre table. La concordance entre les colonnes permet alors d'établir et d'imposer un lien entre les données de ces deux tables.
La solution mise au point est présentée sous la forme d'une méthodologie utilisant plusieurs techniques qui, mises en commun, fournissent un outil permettant la gestion des clés étrangères.

1.2. Etat de l'art

1.2.1. Introduction

MySQL fait partie de la famille des SGBD de type « Relationnels » dans laquelle plusieurs SGBD sont connus et avérés comme des références en la matière. Dans cette offre, certains sont payants et sont livrés puis maintenus par de grandes entreprises *(Oracle, MS Access, MS SQL-server, Sybase, ...)* tandis que d'autres, dont MySQL fait partie avec PostgreSQL, SAP DB et mSQL, sont disponibles gratuitement sur Internet et leur code source est mis à la disposition des utilisateurs dans le but de permettre les modifications ou l'évolution en fonction des besoins spécifiques de chacun.

Le texte qui va suivre présente brièvement les points forts des logiciels libres et est extrait des articles rédigés par M. Mattou [Mattou 2003] et J. Sing [Sing 2002] sur le sujet.

Les principaux avantages d'un SGBD dont le code source est public sont les suivants :

- Diminution des coûts grâce à la disparition des licences d'utilisation.
- Flexibilité, permettant de modifier le SGBD pour le personnaliser.
- Amélioration constante grâce à l'apport des connaissances provenant de la communauté du « Logiciel Libre ».
- Documentation riche, variée et abondante provenant des développeurs mais également des utilisateurs.

Les bases de données sont au cœur du système d'information des entreprises. Leur choix est donc très important. Les SGBD de type « Logiciels Libres » constituent une véritable alternative pour un bon nombre de PME, qui payent des licences d'utilisation pour des bases de données commerciales et propriétaires et dont le support engendre des coûts relativement élevés. Considérés auparavant comme un phénomène de mode, les logiciels libres apparaissent désormais comme un modèle crédible pour le développement et la diffusion d'outils et de solutions logicielles. Performants et fiables, ces logiciels libres séduisent de plus en plus de sociétés.

Cependant, les logiciels libres présentent aussi certains points faibles. En effet, le développement de tels logiciels est très rapide mais en même temps très fragile. Il est fréquent de voir apparaître de façon régulière de multiples versions qui, avant tout, corrigent les erreurs des versions précédentes. Cela ne permet pas aux entreprises sollicitant des logiciels stables de faire confiance en de tels produits. Il est à noter également que le support et la maintenance des logiciels libres peuvent disparaître à tout instant car ces tâches sont parfois confiées à des groupes de personnes ou à des entreprises éphémères. Pour terminer, le coût des logiciels libres n'est pas totalement gratuit car il faut tenir compte des paramètres suivants qui n'apparaissent pas toujours lors de l'acquisition de tels logiciels : les formations et le support technique offerts par les sociétés qui proposent des logiciels libres sont, quant à eux, payants.

1.2.2. Conception d'une base de données

La mise en place d'une base de données s'effectue en plusieurs étapes. La première est l'analyse conceptuelle qui génère comme résultat un schéma représentatif du domaine d'application repris dans le cahier des charges. Par la suite, vient l'étape de conception logique qui a pour but d'obtenir une représentation conforme au modèle de famille auquel le SGBD utilisé appartient. Après cette étape, arrive la phase de conception physique qui a pour objectif d'exploiter les techniques d'optimisation offertes par le SGBD. Pour terminer, la dernière phase consiste à procéder au codage qui permettra le déploiement de la base de données dans l'environnement du SGBD sélectionné.
Afin d'aider le concepteur dans ce travail, des ateliers de génie logiciel *(AGL)* sont disponibles. Les outils de ce type proposent diverses fonctionnalités permettant au concepteur d'accomplir les différentes étapes de création d'une base de données. Il existe un bon nombre d'ateliers de génie logiciel, allant des plus généralistes aux plus spécifiques dédiés à un SGBD précis. Parmi la multitude d'ateliers de génie logiciel existants, les plus connus sont les suivants : Rational Rose *(Rational Software),* Visual CASE *(Artiso),* Together *(Borland),* DB-MAIN *(FUNDP),* Studio2 *(CharonWare), ...*

Etant donné que la notoriété du système de gestion de bases de données MySQL doit encore s'imposer, il existe, à l'heure actuelle, peu d'ateliers de génie logiciel permettant d'aider le concepteur à mettre en place une base de données MySQL. Les logiciels proposés actuellement pour assister le concepteur de bases de données MySQL sont les ateliers de génie logiciel « Studio 2 » et « DeZign For Databases ». Ces deux ateliers permettent de représenter un schéma conforme au modèle Entité-Association lors de la phase d'analyse conceptuelle. Ce schéma peut, par après, être transformé en un schéma logique relationnel et être optimisé en utilisant les techniques offertes par MySQL. Ces opérations sont effectuées via les diverses fonctionnalités offertes par ces logiciels. Enfin, ces ateliers de génie logiciel sont utilisés pour générer un script contenant les instructions de création de la base de données mise en place.

Les informations concernant les deux outils présentés ci-dessus peuvent être obtenues via les références suivantes : [CharonWare 2003] et [Datanamic 2003].

Il est à noter qu'il existe également des outils spécifiquement conçus pour faciliter uniquement la phase de codage d'une base de données MySQL. Le plus utilisé est le logiciel « MySQL Control Center ». Ce logiciel assiste la personne chargée du déploiement de la base de données. Via ce logiciel, l'utilisateur a la possibilité de déployer une base de données MySQL de façon visuelle. De plus amples informations sur ce logiciel sont disponibles en consultant la référence [MySQLCC 2003].

1.2.3. Techniques de gestion de contraintes dans les SGBD

Le texte de présentation des techniques de gestion de contraintes qui va suivre est une synthèse de différents articles écrits sur le sujet. Les lecteurs intéressés par les articles peuvent consulter les références suivantes dans la bibliographie : [Ceri, Cochrane, Widom 2000], [Estier, Pigneur 2000] et [Leblanc 2003].

La mise en place d'une technique de gestion de contraintes peut s'effectuer en faisant appel à des constructions internes ou externes au SGBD. Parmi les éléments internes au SGBD, se retrouve premièrement, la technique de déclaration et de gestion explicite. Une contrainte est alors exprimée sous la forme d'un prédicat associé à une colonne ou à une table. La norme SQL-92 fournit plusieurs mécanismes pour la déclaration explicite des contraintes d'intégrité. Les formats de contraintes les plus populaires *(contraintes sur les clés, les contraintes null ou not null et les contraintes d'intégrité référentielle)* ont chacun leur syntaxe définie par cette norme. Cette technique permet par exemple, de déclarer des contraintes relatives au domaine de valeurs prises par les attributs. Elle permet également de déclarer des identifiants primaires ou secondaires et, pour certains SGBD, il est même possible de déclarer explicitement des contraintes de type « check » ou des contraintes relatives à la cohérence entre les clés étrangères et les clés primaires des tables référencées.
Comme seconde possibilité offerte par le SGBD, lorsqu'il n'est pas possible de déclarer les contraintes explicitement, la gestion des contraintes peut alors s'effectuer via les déclencheurs ou encore via les procédures stockées. Ces méthodes de gestion permettent le contrôle de la cohérence des données lors de l'interaction avec la base de données. Ces techniques définissent la réaction à adopter en cas d'interaction avec la base de données. Par rapport aux contraintes explicitement déclaratives, les déclencheurs sont procéduraux. En effet, dans les déclencheurs, se trouve obligatoirement un événement *(une insertion, une suppression ou une mise à jour dans une table particulière)* qui implique qu'une condition doit être respectée pour le déclenchement de la procédure de vérification de la contrainte. Le déclencheur définit également la réponse à apporter en cas de violation. Les déclencheurs ont une période d'activation *(avant ou après l'évènement défini)* et une granularité *(réaction sur chaque ligne de la table ou non).*
Une procédure stockée est un ensemble d'instructions SQL et est intégrée à la base de données. Les programmes d'applications invoquent ces procédures plutôt que les primitives SQL. Cette façon de faire facilite la tâche du programmeur.
Les déclencheurs et les procédures stockées sont utilisés pour une multitude de besoins significatifs dans les SGBD. Leur utilisation principale est toujours actuellement relative à la gestion des diverses contraintes d'intégrité. Par exemple, dans une base de

données contenant des tables reliées via des clés étrangères, un déclencheur peut être mis en place afin de vérifier que les valeurs des enregistrements insérés dans une table sont bien reprises dans les tables de référence. L'évènement sera activé avant chaque insertion d'un nouvel enregistrement et, si la condition est respectée pour le lancement de la procédure de vérification, le déclencheur fait alors appel à une procédure stockée dans laquelle se trouve l'ensemble d'instructions SQL qui permet d'effectuer la validation. En fonction de la réaction programmée dans le déclencheur, l'insertion sera acceptée ou refusée.

Dans le cas où le SGBD n'offre pas les possibilités précédemment décrites, la gestion des contraintes peut alors s'effectuer de façon modulaire via les écrans de saisie ou via les programmes ; éléments externes au SGBD dans lesquels s'opère une vérification avant la validation effective dans la base de données. Toutefois, il faudra tenir compte du fait que toutes les applications clientes qui effectuent des interactions avec la base de données doivent invoquer ces modules afin que la gestion des contraintes programmées puisse s'effectuer. Les programmes externes accèdent aux bases de données via des interfaces *(API)*. Les langages utilisés pour l'élaboration de tels modules sont les langages de programmation de troisième génération tels que Cobol, Pascal, C, C++ et Java. Ces langages permettent l'imbrication d'ordres SQL dans les programmes. Un programmeur peut générer des chaînes de caractères représentant des instructions SQL et les passer en paramètre à des fonctions chargées de les exécuter. Les résultats obtenus sont agrégés dans des structures de données adéquates pour le langage de programmation utilisé. Pour permettre de telles interactions, des librairies de classes normalisées ont vu le jour *(ODBC, JDBC)*. Ces librairies se veulent indépendantes des éditeurs de SGBD. Ce compromis entre indépendance et connectivité peut toutefois occasionner des pertes de performance et d'expressivité. Les exemples de ce type d'applications externes sont multiples et variés. Grâce à cette technique, il est possible de programmer des modules de gestion de contraintes comme par exemple, les contraintes de clés étrangères ou encore d'autres types de contraintes. Ces programmes externes sont souvent également mis en place pour des besoins spécifiques et personnalisés.

Concernant le SGBD MySQL, vu que celui-ci n'offre pas pour le moment la possibilité de mettre en place des déclencheurs ni des procédures stockées, la gestion des contraintes s'exécutera donc de manière modulaire via les éléments externes au SGBD. Un des éléments externes le plus majoritairement associé à une base de données MySQL est le langage de programmation de scripts PHP avec lequel sont créés les programmes d'interactions avec la base de données. Dans ce genre d'applications, le programmeur réalise toute une série de contrôles lui permettant de valider ou de refuser une instruction affectant la base de données. Le programme d'interaction communique avec la base de données via l'utilisation d'interfaces programmables avec la base de données.
En utilisant cette technique pour la gestion des clés étrangères, le programmeur doit avoir à sa disposition le schéma physique contenant les informations relatives aux liens unissant les tables de la base de données. De plus, une bonne coordination entre le concepteur de la base de données et le programmeur est nécessaire dans le cas où il s'agit de personnes différentes. Il en va de même pour les programmeurs utilisant des langages de programmation autres que PHP, ils doivent également gérer le contrôle de cohérence des données lors de l'élaboration de leurs programmes.

D'autres techniques permettent de gérer les clés étrangères. Avec l'utilisation du moteur de stockage mis au point par la société Innobase et fourni avec le SGDB MySQL, il est possible de déclarer et de gérer les clés étrangères. Les informations relatives à ce moteur de stockage peuvent être obtenues en consultant la référence suivante : [Innodb 2003]. Il est à noter cependant que ce moteur de stockage n'est pas fourni avec l'entièreté des versions de MySQL et que les anciennes versions ne peuvent pas profiter de cette nouvelle fonctionnalité.
Des applications de type « consoles graphiques » ont également été mises au point par des sociétés extérieures. Ces consoles jouent le rôle d'écrans de saisie dans lesquels s'effectue la gestion des clés étrangères lors de l'interaction avec une base de données. Des informations complémentaires concernant ces logiciels peuvent être obtenues en consultant les références suivantes : [Webyog 2003] et [JETA Software 2003]. Ce genre d'applications ne permet toutefois pas de valider une interaction qui serait effectuée en dehors de l'environnement de la console graphique. Un programme externe ne pourra donc pas faire appel à la fonctionnalité de gestion des clés étrangères développée dans ces outils graphiques.

1.3. Structure du mémoire

Dans ce mémoire, les principes de fonctionnement du SGBD MySQL vont être exposés et seront suivis d'une présentation théorique des concepts généraux relatifs à la construction d'une base de données. Ces concepts vont être appliqués à MySQL et leur mise en pratique sera illustrée par l'utilisation d'un atelier logiciel spécifique à la création des bases de données. Pour rendre possible le déploiement de la base de données mise en place dans l'atelier logiciel, l'appel et la description d'un module d'extension vont être présentés en détails. Par la suite, dans le cadre de la gestion des contraintes, un rappel des différentes techniques de programmation utilisant les services d'une base de données va être réalisé. Ce rappel sera suivi de la présentation de l'utilisation des modules développés pour la gestion des clés étrangères dans l'environnement de MySQL.
Pour terminer, la description technique des différents modules mis en place sera mentionnée et une étude de cas complète synthétisera les différents concepts abordés dans ce mémoire.

Le présent document est structuré en neuf chapitres. Les concepts abordés dans chacun d'eux sont les suivants :

- Chapitre 1 : Introduction, présentation des objectifs et description du contexte.

- Chapitre 2 : Présentation du système de gestion de bases de données MySQL, analyse des différents principes de fonctionnement. Enoncé des fonctionnalités absentes, description de la problématique et comparatif avec d'autres SGBD.

- Chapitre 3 : Rappel théorique des concepts généraux de la méthodologie de conception de bases de données.

- Chapitre 4 : Etude détaillée du processus complet de création d'une base de données MySQL. Expression générique du modèle de données. Description des phases de conception logique, physique et de codage relatives au SGBD MySQL.

- Chapitre 5 : Présentation concrète de la méthode de conception d'une base de données MySQL avec l'aide de l'atelier logiciel DB-MAIN. Illustration de l'appel du module de création de code de la base de données et du module de définition des clés étrangères.

- Chapitre 6 : Présentation des paramètres et de la structure d'un programme utilisant les services d'une base de données MySQL. Présentation des techniques prises en compte pour la réalisation du module développé pour la gestion des clés étrangères dans le cadre de ce mémoire. Illustration de l'appel des fonctions contenues dans le module de gestion des clés étrangères.

- Chapitre 7 : Description et analyse des outils de gestion mis en place dans le cadre de ce mémoire. Les trois modules présentés sont :

    - Le module de génération de code de définition d'une base de données MySQL.
    - Le module de génération du fichier représentatif des clés étrangères.
    - Le module de gestion des clés étrangères, réalisé sous la forme d'une librairie de fonctions.

- Chapitre 8 : Etude de cas. Illustration d'un exemple complet de la création d'une base de données depuis le domaine d'application jusqu'à la phase de déploiement dans l'environnement de MySQL. Comparatif de deux fonctions. La première n'utilise pas le module de gestion des clés étrangères tandis que la seconde y fait appel.

- Chapitre 9 : Conclusions et perspectives.

Chapitre 2 : Description de MySQL

Avant d'aborder l'analyse du processus de conception d'une base de données MySQL dans les chapitres suivants, il est nécessaire de présenter le SGBD MySQL de façon relativement détaillée. La description qui va suivre va mettre en évidence les particularités, les points forts et les points faibles de ce SGBD. Cette description est, en grande partie, extraite des ouvrages dont les références sont les suivantes : [Aquilina 2003] et [Dubois 2000].
Ce chapitre est composé de cinq parties : la première va permettre de prendre contact avec le système de gestion de bases de données MySQL. Ensuite, la seconde partie énonce les bases de son fonctionnement. La troisième partie souligne les fonctionnalités absentes de MySQL et la quatrième présente la problématique de l'absence de gestion des clés étrangères et de l'intégrité référentielle. Enfin, la dernière partie de ce chapitre propose un bref comparatif entre MySQL, trois autres SGBD et la norme SQL-92.

2.1. Introduction

2.1.1. Qu'est-ce que MySQL ?

MySQL est un système de gestion de bases de données relationnelles performant, rapide et simple à utiliser, fondé en 1994 par Michael Monty pour les besoins d'une compagnie suédoise *(TcX)*. Actuellement, suite à la distribution de MySQL via Internet et son succès croissant, la société MySQL AB s'occupe du développement, du support et de la vente des produits MySQL.
MySQL est doté d'une architecture Client-Serveur reprenant un serveur de bases de données multitâche et multi-utilisateur, ainsi que plusieurs programmes clients.

2.1.2. Pourquoi choisir MySQL ?

MySQL a été développé dans le cadre des logiciels libres. De ce fait, il est totalement gratuit pour les particuliers, à l'exception de ceux qui veulent vendre des services en utilisant une base de données MySQL *(dans ce cas, une licence est nécessaire)*.
Même en payant une licence, MySQL reste un système de bases de données peu onéreux par rapport à d'autres systèmes dont le prix les destine à de grosses entreprises.
MySQL, tout comme d'autres Systèmes de Gestion de Bases de Données, possède les caractéristiques suivantes :

- Le support du langage des requêtes : MySQL supporte SQL, le langage de choix de tous les systèmes modernes de bases de données. Il est possible également d'accéder à MySQL via ODBC, un protocole de communication développé par Microsoft.
- Les capacités : plusieurs clients peuvent se connecter sur un serveur en même temps et utiliser différentes bases de données simultanément.
- Les connexions sécurisées : MySQL est un produit conçu pour les réseaux et il est possible d'accéder aux bases de données à partir d'Internet. Un système de contrôle intégré interdira la manipulation des données à ceux qui n'en ont pas l'autorisation.
- La portabilité : MySQL peut s'exécuter sous différentes variétés de systèmes Unix ainsi que sous d'autres systèmes tels que Windows, Mac OS X, Solaris, ...

MySQL possède également les points forts suivants :

- La vitesse, la facilité d'utilisation et le faible coût.
- La distribution ouverte : Il est facile de se procurer MySQL, de le comprendre, de l'analyser et même de le modifier.
- L'amélioration continue.

2.1.3. Les outils de MySQL

La distribution de MySQL *(version 4.0)* comprend les outils suivants :

- Un serveur SQL : c'est le moteur qui anime MySQL et qui permet l'accession aux bases de données
- Les programmes clients pour accéder au serveur :
    - Le programme interactif permettant d'entrer les instructions et de visualiser les résultats instantanément
    - Les utilitaires permettant d'effectuer plusieurs tâches d'administration ou de maintenance

Quatre versions du serveur de bases de données MySQL sont disponibles :

- MySQL Standard : cette version reprend le moteur de stockage standard MySQL et le moteur de stockage de type InnoDB(1). Cette version est destinée aux utilisateurs désirant des hautes performances des bases de données MySQL avec un support de transactions.

- MySQL Max : destiné aux utilisateurs voulant avoir accès plus rapidement aux nouvelles fonctionnalités étant par ailleurs toujours en phase de tests.

- MySQL Pro : c'est la version pour l'utilisation commerciale de MySQL standard.

- MySQL Classic : cette version inclus seulement le moteur de stockage MySQL.

2.1.4. Les utilisateurs et les applications typiques de MySQL

Les utilisateurs sont, en grande partie, des administrateurs de bases de données, des fournisseurs d'accès Internet, des développeurs Web ou encore des petites PME.
Le type d'applications développées consiste en de petites et moyennes applications de gestion utilisant le Web comme ressource principale. Souvent, dans ce type d'applications, quelques personnes s'occupent de la gestion des données *(insertions et mises à jour)* et ces dernières sont généralement consultées par un bon nombre de personnes via des formulaires Web.

(1) le Moteur de stockage InnoDB est présenté dans la section « Type de stockage des tables » de ce chapitre

2.2. Fonctionnement de MySQL

2.2.1. Connexion à MySQL

La connexion s'effectue au moyen du client mysql depuis une interface de commande du système d'exploitation en mode terminal.

Procédure de connexion :

```
>mysql –h hostname –u username -p
```

-h : nom du serveur
-u : nom de l'utilisateur
-p : mot de passe

Procédure de déconnexion :

```
mysql> quit
```

Les instructions MySQL sont insensibles à la casse. Elles se terminent par « ; » et peuvent s'étendre sur plusieurs lignes.

Les règles d'attribution des noms sont les suivantes :

- Caractères autorisés : alphanumériques, « _ » et « $ »
- Peuvent débuter par un chiffre
- Non exclusivement composés de chiffres
- De longueur de 64 caractères maximum *(pour les alias, 256 caractères)*.

2.2.2. Gestion des bases de données

Les instructions relatives à la gestion d'une base de données sont les suivantes :

Lister les bases de données du serveur :	mysql> SHOW DATABASES ;
Créer une base de données :	mysql> CREATE DATABASE *database* ;
Supprimer une base de données :	mysql> DROP DATABASE *database* ;
Activer une base de données :	mysql> USE *database* ;
Connaître la base de données courante :	mysql> SELECT DATABASE() ;

L'instruction USE n'est pas utilisée dans le cas où le nom de la base de données est inséré dans l'instruction de connexion.

2.2.3. Gestion des tables

Les instructions principales relatives à la gestion d'une table sont les suivantes :

Lister les tables :	mysql> SHOW TABLES ;
Créer une table :	mysql> CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tab_name (spécification des colonnes)* ;
Supprimer une table :	mysql> DROP TABLE *tab_name* [IF EXISTS] ;
Décrire une table :	mysql> DESCRIBE *tab_name* ; ou mysql> SHOW COLUMNS FROM *tab_name* ;
Modifier une table :	mysql> ALTER TABLE *tab_name action action_description* [, *action action_description,* ...] les actions possibles sont : ADD ALTER CHANGE DROP MODIFY RENAME AS
Copier une table :	mysql> CREATE [TEMPORARY] TABLE *tab_name* SELECT ... ; La façon dont le select est écrit permet de faire trois types de copies : - Une copie identique de la table - Une copie vide de la table *(on conserve la structure uniquement)* - Une copie temporaire d'une table. *(en utilisant TEMPORARY).*

L'option [IF NOT EXIST] permet de créer la table uniquement si elle n'existe pas déjà. Cette commande est utilisée dans des applications qui ne peuvent déterminer si les tables dont elles ont besoin ont été définies par avance, et qui tentent systématiquement de créer les tables. En mode batch, l'instruction ordinaire CREATE TABLE ne fonctionne pas très bien. Les tables seront créées au cours de la première exécution du batch mais une erreur se produira dès la deuxième exécution vu que les tables sont déjà existantes. En utilisant IF NOT EXIST, le problème disparaît.

L'option [IF EXIST], lors d'un DROP TABLE, permet de ne pas générer d'erreur si la table n'existe pas.

Une table temporaire [TEMPORARY] disparaît automatiquement à la fin de la session client *(il n'est pas nécessaire de générer l'instruction DROP TABLE)*. Une table temporaire n'est visible que par le client responsable de sa création. Le nom de la table temporaire peut être celui d'une table permanente existante. Cette possibilité permet de faire des manipulations *(par exemple : des tests)* sans risques d'erreurs sur une table permanente.

Insérer des lignes :

```
mysql> INSERT [LOW_PRIORITY | DELAYED][IGNORE][INTO]
            tab_name [(column_name, ...)]
            VALUES ...
mysql> INSERT [LOW_PRIORITY | DELAYED][IGNORE][INTO]
            tab_name [(column_name, ...)]
            SELECT ...
mysql> INSERT [LOW_PRIORITY | DELAYED][IGNORE][INTO]
            tab_name
            SET column_name = expression ...
```

Trois syntaxes permettent d'insérer des enregistrements dans une table.
La première utilise VALUES et permet l'insertion à partir de valeurs fournies dans l'instruction.
Le seconde utilise SELECT et permet d'insérer des valeurs extraites d'une autre table.
La troisième insère les colonnes définies dans la clause SET ainsi que les valeurs correspondantes.

[LOW_PRIORITY] permet de différer l'insertion au moment où aucun autre client n'accède en lecture à la table.

[DELAYED] permet de différer l'insertion des lignes en les plaçant dans une file d'attente.

Si [IGNORE] est spécifié, les lignes contenant des doublons de valeurs sur un même index sont ignorées. En l'absence de cet attribut, une telle situation provoquerait une erreur.

Modifier des lignes :

```
mysql> UPDATE [LOW_PRIORITY] tab_name
            SET column_name = expression1, column_name =
            expression2, ...
            [WHERE condition] [LIMIT n]
```

Supprimer des lignes :

```
mysql> DELETE [LOW_PRIORITY] FROM tab_name
            [WHERE condition] [LIMIT n]
```

La clause [LIMIT *n*] est utile pour limiter le résultat d'une action au n premières rangées.

Sélectionner des lignes

```
mysql> SELECT
       [options_de_selection]
       expression_de_selection
       [INTO OUTFILE 'nom_de_fichier'
       options_export]
       [FROM table_de_reference
       [WHERE expression_where]
       [GROUP BY column_name, ..., ...]
       [HAVING expression_where]
       [ORDER BY
       {entier_non_signé|column_name|formule}
       [ASC|DESC], ...]
       [LIMIT [décalage,] n]
       [PROCEDURE nom_procédure] ]
```

Dans sa forme la plus simple, l'instruction SELECT ne nécessite qu'une expression « *expression_de_selection* », le reste *(mis entre [])* est facultatif.

« expression_de_selection » spécifie la liste des colonnes à retourner. Chaque colonne doit être séparée de la suivante par une virgule. Il est aussi possible d'utiliser le caractère « * » pour afficher toutes les colonnes de toutes les tables de la base de données en cours ou la syntaxe « *table_de_reference.** » pour retourner toutes les colonnes de la table *table_de_reference*. Les colonnes peuvent se voir assigner un alias en employant la syntaxe « *AS alias_name* ». Afin d'effectuer des calculs ou de renvoyer un résultat exploitable, MySQL dispose également de nombreuses fonctions spécialisées comme par exemple, les fonctions count( ), sum( ), max( ), concat( ), ...

Les options de sélections sont les suivantes :

[ALL | DISTINCT | DISTINCTROW] : cette option permet de contrôler que des rangées sans doublons sont retournées. ALL permet de renvoyer toutes les rangées *(par défaut)*. DISTINCT et DISTINCTROW indiquent qu'on doit éliminer toutes les rangées dupliquées de l'ensemble du résultat.

[STRAIGHT_JOIN] : cette option permet de forcer l'association des tables dans l'ordre de la clause FROM.

[SQL_SMALL_RESULT] [SQL_BIG_RESULT] : permet d'indiquer si le résultat attendu est de petite ou de grande taille.

La clause [INTO OUTFILE *'nom_de_fichier' options_export*] permet d'écrire le résultat de l'instruction SELECT dans un fichier.

La clause [FROM *table_de_reference* ...] spécifie le nom de la table à partir de laquelle les lignes seront lues. L'instruction FROM peut faire référence à plusieurs tables.

L'option [WHERE *expression_where*] est utilisée pour spécifier une expression permettant de sélectionner les lignes devant être lues. Les rangées qui ne satisfont pas aux critères donnés par l'expression « *expression_where* » seront rejetées.

La clause [GROUP BY *column_name*, ..., ...] permet de grouper les résultats en fonction des colonnes spécifiées par après. Plusieurs colonnes, séparées par une virgule, peuvent y figurer.

L'option [HAVING *expression_where*] permet de limiter les lignes de la clause WHERE, au moyen de « *expression_where* ». Les rangées qui ne satisfont pas à la condition HAVING sont rejetées. HAVING est utile pour des expressions impliquant des fonctions récapitulatives qui ne peuvent pas être examinées dans la clause WHERE.

La clause [ORDER BY {*entier_non_signé*|*column_name*} [ASC|DESC], ...] permet de trier les résultats en fonction des options disponibles. « *entier_non_signé* » représente le numéro de position d'une colonne. « *column_name* » représente le nom d'une colonne. Le tri peut être effectué de manière ascendante *(ASC)* ou descendante *(DESC)*.

La clause [LIMIT [décalage,] *n*] permet de limiter le nombre de lignes retournées à un nombre défini par « n ». Il est possible aussi de spécifier la position de la première ligne qui doit être retournée au moyen de l'argument optionnel « décalage ». Les rangées sont numérotées en commençant par 0 et non pas par 1.

[PROCEDURE *nom_procédure*] précise que les résultats doivent être premièrement envoyés à une procédure avant d'être envoyés au client.

2.2.4. Types de stockage des tables

Les différents types de stockage sont les suivants :

- ISAM
- MyISAM
- HEAP
- InnoDB
- MERGE

Il est possible de spécifier le type de stockage lors de la création de la table ou de le modifier après sa création. Pour se faire, il faut utiliser une des deux instructions suivantes :

```
mysql> CREATE TABLE table_name (définition des colonnes) TYPE = ... ;
ou
mysql> ALTER TABLE table_name TYPE = ... ;
```

ISAM
Avant la version 3.23 de MySQL, toutes les tables créées par l'utilisateur faisaient appel à une méthode de stockage de type ISAM. Dorénavant, ce type est appelé à disparaître pour laisser place au type par défaut actuel qui est le type MyISAM.

MyISAM
Type par défaut de MySQL depuis la version 3.23 proposant de nombreuses fonctionnalités pratiques. Chaque table MyISAM utilise 3 fichiers stockés dans le répertoire « DATA » de MySQL. La description de la table est contenue dans un fichier « .FRM ». Les données sont stockées dans un fichier avec l'extension « .MYD » et l'index est stocké dans un fichier avec l'extension « .MYI ». La gestion des index numériques occupe moins d'espace et les tailles des fichiers autorisés sont plus importantes qu'avec la méthode de stockage ISAM. Les données, quant à elles, sont stockées dans un format indépendant de la plate-forme. Grâce à cela, il est possible de copier des tables d'une machine vers une autre même si leurs architectures sont différentes.

HEAP
Les tables HEAP sont stockées en mémoire, elles sont très rapides mais lors d'un accident ou lors du redémarrage du serveur, toutes les données stockées seront perdues *(la structure de la table (.FRM), n'étant pas stockée en mémoire, reste conservée)*. Les tables HEAP sont partagées par tous les clients *(comme pour une table standard)*. Toutefois, ces tables utilisent un format d'enregistrement fixe et ne supportent pas certains types de colonnes.

InnoDB
InnoDB est un moteur de stockage transactionnel intégré dans MySQL et fourni par la société Innobase. Ce mode transactionnel reprenant les instructions COMMIT et ROLLBACK est conforme à la norme « ACID ». Cette norme prend en compte les quatre critères suivants :

- Atomicity *(atomicité)* : l'ensemble des instructions de la section sont exécutées complètement ou pas du tout, mais jamais partiellement.
- Consistency *(cohérence)* : si la base de données est cohérente avant une transaction, elle le restera à la sortie de celle-ci *(respect des contraintes d'intégrité)*.
- Isolation *(indépendance)* : les interactions entre la transaction et la base de données s'effectuent comme si la transaction était seule à travailler sur la base de données.
- Durability *(permanence)* : si la transaction s'est terminée correctement, les modifications faites dans la base de données sont garanties permanentes, quels que soient les incidents ultérieurs.

Une table de type transactionnel est moins rapide et utilise plus de ressources qu'une table non-transactionnelle mais a pour avantage qu'elle est plus sécurisée, en effet, cela permet par exemple le rétablissement rapide suite à un accident.
Il est à noter que le type InnoDB permet également le verrouillage au niveau des enregistrements. De plus, il est à signaler que dans les versions récentes de MySQL, ce moteur de stockage inclut la gestion des clés étrangères en imposant néanmoins les conditions suivantes : les deux tables doivent être de type InnoDB et il doit y avoir un

index dans lequel les colonnes des clés étrangères et des clés référencées sont définies car le moteur InnoDB ne crée pas automatiquement des index pour les clés étrangères et les clés référencées. De plus, comme dans la plupart des autres SGBD, les colonnes reliées par une clé étrangère doivent être de même type.

MERGE

Une table de type MERGE est une collection de tables MyISAM ayant la même définition et étant représentées comme une seule table. Les seules commandes qu'il est possible d'utiliser avec ce type de tables sont : SELECT, INSERT, DELETE et UPDATE.

Une table de type MERGE consiste en un fichier de définition « .FRM » et un fichier de liste de tables « .MRG »

L'avantage principal de ce type est qu'il est possible d'éclater une grande table en plusieurs petites pour améliorer la maintenance, la rapidité et contourner la limite maximale d'un fichier selon l'OS.

L'inconvénient est que ce n'est utilisé qu'avec des tables MyISAM

Exemple d'utilisation de tables MERGE

```
mysql> CREATE TABLE table1 (colonne char(20)) ;
mysql> CREATE TABLE table2 (colonne char(20)) ;
mysql> INSERT INTO table1 VALUES ('libelle1') ;
mysql> INSERT INTO table2 VALUES ('libelle2') ;
mysql> CREATE TABLE tablemerge (colonne char(20)) TYPE = MERGE
                UNION = (table1, table2) ;
mysql> SELECT * FROM tablemerge ;
```

le résultat de la requête sera :

libelle1
libelle2

2.2.5. Types de données

MySQL supporte de nombreux types de données qu'il est possible de classer en plusieurs catégories définies selon les valeurs qu'elles représentent.

Les valeurs numériques

Les nombres désignent des valeurs telles que 48, -12 ou 193,62. MySQL interprète les nombres spécifiés comme des entiers ou comme des valeurs décimales. Les entiers peuvent être spécifiés dans un format décimal ou hexadécimal. Les nombres peuvent être précédés d'un signe moins (-) pour signifier une valeur négative.

Les valeurs de chaîne de caractères

Les valeurs de chaîne de caractères désignent des expressions telles que « maison », « remplacer une pièce ». Il est possible d'utiliser des guillemets simples ou doubles pour délimiter la chaîne.

Les valeurs temporelles

MySQL comprend également des valeurs combinées de date et d'heure telles que « 1999-06-17 12:30:43 ». Les dates ont un ordre bien déterminé : année – mois – jour *(format standard ANSI SQL).*

La valeur NULL

NULL correspond à une valeur non saisie. Normalement, cela signifie « aucune valeur », « valeur non connue », « valeur manquante », « valeur en dehors de l'intervalle » ...

2.2.6. Types de colonnes

Chaque table dans une base de données est composée d'une ou de plusieurs colonnes. Lors d'un CREATE TABLE, il faut spécifier un type pour chaque colonne de la table ainsi créée. Le type de colonnes caractérise précisément la nature des valeurs qu'une colonne peut contenir.

Colonnes numériques

MySQL propose plusieurs types de colonnes pour les valeurs numériques de type entier ou décimal. Les colonnes de type entier peuvent être signées ou non. Un attribut spécial permet aux valeurs des colonnes de type entier d'être générées automatiquement *(utile dans le cas d'une application produisant des séquences uniques ou des numéros identifiants).*

Nom du type	Description
TINYINT	Entier très petit (taille : 1 octet)
SMALLINT	Entier petit (taille : 2 octets)
MEDIUMINT	Entier de taille moyenne (taille : 3 octets)
INT	Entier standard (taille : 4 octets)
INTEGER	Entier standard (taille : 4 octets)
BIGINT	Entier grand (taille : 8 octets)
FLOAT	Point décimal de précision simple
DOUBLE	Point décimal de précision double
DECIMAL	Point décimal représenté sous forme de chaîne

Les attributs qu'il est possible d'ajouter aux colonnes de type numérique sont les suivantes :

- ZEROFILL : remplit avec des zéros à gauche pour atteindre la taille maximale d'affichage.
- UNSIGNED : interdit l'utilisation de valeurs négatives
- AUTO_INCREMENT : génère des identifiants uniques ou des valeurs en séries. Les valeurs AUTO_INCREMENT commencent normalement par 1 et augmentent par pas de 1.

Colonnes « chaîne »

Les chaînes peuvent supporter n'importe quel caractère, même des données binaires telles que des images ou du son. Les chaînes peuvent être comparées sous un mode sensible ou non à la casse.

Nom du type	Description
CHAR	Une chaîne de caractères de longueur fixe
VARCHAR	Une chaîne de caractères de longueur variable
TINYBLOB	Un très petit BLOB (Binary Large Object) (0->255 octets) sensible à la casse
BLOB	Un petit BLOB (-> 64Ko) sensible à la casse
MEDIUMBLOB	BLOB de taille moyenne (->16Mo) sensible à la casse
LONGBLOB	Un grand BLOB (-> 4Go) sensible à la casse
TINYTEXT	Une très petite chaîne de texte (0->255 octets) insensible à la casse
TEXT	Une petite chaîne de texte (-> 64Ko) insensible à la casse
MEDIUMTEXT	Une chaîne de texte de taille moyenne (-> 16Mo) insensible à la casse
LONGTEXT	Une grande chaîne de texte (-> 4Go) insensible à la casse
ENUM	Une énumération de valeurs prédéfinies parmi lesquelles une seule peut être assignée (-> 64Ko)
SET	Un ensemble de valeurs prédéfinies parmi lesquelles une ou plusieurs valeurs peuvent être assignées (-> 64Ko)

CHAR et VARCHAR sont les types de colonnes les plus courants pour représenter des chaînes. La différence entre ces deux types concerne la longueur qui est fixe pour le type CHAR et qui est variable en fonction du contenu pour une colonne de type VARCHAR.

Le terme BLOB désigne un grand objet binaire qui pourra supporter tout ce qu'il recevra *(image, son, ...)*. Le type BLOB désigne plutôt une famille de type *(TINYBLOB, MEDIUMBLOB, LONGBLOB)*, tous identiques mis à part le volume maximal d'information qu'ils peuvent contenir.
Le terme TEXT est semblable à son homologue BLOB sauf que le type TEXT n'est pas sensible à la casse et qu'il gère différemment les opérations de comparaison et de tri.

ENUM et SET sont des types spéciaux de chaînes de caractères pour lesquels des valeurs de colonnes doivent être choisies à partir d'un ensemble fixe de chaînes de caractères. La différence principale entre ces deux types est la suivante : ENUM est utilisé pour les valeurs mutuellement exclusives *(une et une seule)*, tandis que SET permet le choix multiple dans une liste de valeurs. Ces types de données sont un avantage du SGBD MySQL par rapport à la norme SQL-92.

Le type de colonnes ENUM définit une énumération. Ces dernières sont généralement employées pour représenter des valeurs de catégorie *(par exemple : « oui » ou « non », « essence » ou « diesel » ou « lpg »)*. Ce type est utilisé dans les formulaires comme un « bouton radio ».

Le type SET est semblable à ENUM, car lors de la création d'une colonne SET, une liste d'éléments valides est définie. Cependant, chaque valeur de colonne peut se composer de n'importe quel nombre d'éléments de l'ensemble *(par exemple : représenter la liste des options d'une automobile)*. Ce type est utilisé dans les formulaires comme un ensemble de « cases à cocher ».

Colonnes temporelles

Pour les valeurs temporelles, MySQL propose des types pour les dates *(avec ou sans heure)*, les heures et les durées. Il existe également un type pour représenter plus efficacement les valeurs des années lorsqu'il n'est pas nécessaire d'avoir une date entière.

Nom du type	Description
DATE	Une valeur date, au format « YYYY-MM-DD »
TIME	Une heure ou une durée, au format « hh:mm:ss » ou même hhh:mm:ss *(dans le cas d'une durée)*
DATETIME	Une valeur date et heure, au format « YYYY-MM-DD hh:mm:ss »
TIMESTAMP	Un moment précis, au format YYYYMMDDhhmmss
YEAR	Une valeur année, au format « YYYY »

MySQL représente toujours les dates avec l'année d'abord, selon les spécifications de la norme ANSI.

Les attributs généraux

Les attributs généraux, qu'il est possible d'associer à n'importe quel type de colonnes sont les suivants :

- NULL ou NOT NULL : utilisé pour indiquer si une colonne supporte des valeurs NULL ou non. Cela revient à dire, utilisé pour définir si une colonne doit obligatoirement être remplie ou non.

- DEFAULT *valeur_par_défaut* : utilisé pour signaler qu'une colonne reçoit automatiquement la *valeur_par_défaut* mentionnée dès qu'un nouvel enregistrement est créé. La valeur mentionnée doit être une constante. *(Il n'est pas possible de définir une valeur par défaut pour les colonnes de type BLOB ou TEXT)*.

2.2.7. Gestion des index

Le principal objectif des index est d'accélérer l'accès au contenu des tables, en particulier, pour les requêtes impliquant des jointures de plusieurs tables.
Avec MySQL, la construction des index est assez souple. Il est possible d'indexer une ou plusieurs colonnes. Une table peut également contenir plusieurs index.

Création des index

Il est possible de créer des index dans une nouvelle table à la suite d'une instruction CREATE TABLE ou d'ajouter par après des index dans des tables existantes.
Pour se faire, il suffit d'utiliser une des commandes suivantes :

```
mysql> CREATE INDEX index_name ON table_name (liste_colonnes) ;
mysql> CREATE UNIQUE INDEX index_name ON table_name (liste_colonnes) ;

mysql> ALTER TABLE table_name ADD INDEX index_name (liste_colonnes) ;
mysql> ALTER TABLE table_name ADD UNIQUE index_name (liste_colonnes) ;
mysql> ALTER TABLE table_name ADD PRIMARY KEY (liste_colonnes) ;
```

« table_name » est le nom de la table dans laquelle l'index doit être ajouté et *« liste_colonnes »* indique la ou les colonnes à indexer. Si l'index est constitué de plusieurs colonnes, celles-ci seront séparées par des virgules.

Il est possible d'autoriser ou d'interdire les doublons dans les valeurs de l'index. Dans le second cas, l'index doit être créé comme tel avec les contraintes PRIMARY KEY ou UNIQUE. Dans le cas d'un index unique sur une colonne, cette dernière sera garantie sans doublons et dans le cas d'un index unique sur plusieurs colonnes, aucune combinaison de valeur n'est dupliquée.
PRIMARY KEY et UNIQUE sont deux types d'index très proches. PRIMARY KEY est simplement un index UNIQUE auquel on attribue le nom de PRIMARY. Lorsqu'une contrainte d'unicité de type PRIMARY KEY est définie dans une table, un index ayant le nom PRIMARY est automatiquement créé. Il ne sera pas possible de créer une seconde clé primaire puisque deux index ne peuvent pas porter le même nom dans une table.

Suppression des index

Pour supprimer un index, il suffit de lancer une des deux instructions équivalentes suivantes :

```
mysql> DROP INDEX index_name on table_name ;
mysql> ALTER TABLE table_name DROP INDEX index_name ;
```

La suppression de colonnes dans une table peut affecter les index. Si une colonne faisant partie d'un index est effacée, celle-ci sera aussi supprimée de l'index. Si toutes les colonnes constituant un index sont supprimées, l'index complet disparaîtra également.

2.3. Les fonctionnalités non supportées de MySQL

Cette section décrit les fonctionnalités existantes dans la norme SQL-92 et dans d'autres systèmes de gestion de bases de données, mais que MySQL ne supporte pas. L'absence de ces fonctionnalités, présentant un intérêt pratique et intéressant, est principalement justifiée par leur influence négative sur les performances. Toutefois, dans les versions futures de MySQL, il est prévu qu'elles soient implémentées.

2.3.1. Les clés étrangères et l'intégrité référentielle

Une clé étrangère permet de déclarer qu'une colonne dans une table est liée à la colonne d'une autre table et, l'intégrité référentielle permet d'imposer des contraintes sur les actions autorisées dans la table contenant la clé étrangère.
Les clés étrangères permettent d'améliorer la cohérence des données et offrent un certain niveau de confort. Si MySQL ne les supporte nativement, c'est principalement à cause de certains effets négatifs sur les performances de la base de données. Les développeurs de MySQL ne considèrent pas cette fonctionnalité comme « essentielle » compte tenu du type d'applications utilisant une base de données MySQL.
La syntaxe d'écriture est supportée, il est possible de déclarer une FOREIGN KEY lors d'un CREATE TABLE mais MySQL n'appliquera pas les contraintes liées aux clés étrangères. Cependant, depuis les récentes versions de MySQL et en faisant appel au moteur de stockage de type InnoDB, il est quand même possible de déclarer et de gérer les clés étrangères en respectant les conditions définies par ce moteur de stockage.
La mise en place intégrale de la gestion des clés étrangères et de l'intégrité référentielle est quand même prévue dans les versions futures de MySQL.

2.3.2. Les procédures stockées et les déclencheurs

Une procédure stockée est constituée de code SQL compilé et stocké dans le serveur. Il est possible de faire référence au code par après sans avoir besoin de le transférer depuis le client ni de l'analyser de nouveau. Cela permet aussi d'effectuer des modifications dans une procédure *(un seul endroit de gestion)* qui seront répercutées dans toute application client qui l'utilise.
Le déclencheur, quant à lui, permet d'activer une procédure stockée lorsqu'un événement se produit, comme par exemple, lors de l'insertion d'un enregistrement dans une table.
Ces possibilités permettent de faciliter le contrôle et le traitement des données afin que ces dernières conservent leur cohérence et leur intégrité.
De plus, de tels outils apportent une souplesse et un gain de temps non négligeable lors des phases de développement et de maintenance des programmes d'application.

Un langage de procédures stockées est prévu dans les versions futures de MySQL.

2.3.3. Les vues

Une vue est une entité logique se comportant comme une table sans en être une *(table virtuelle)*. Elle permet de rassembler des colonnes issues de plusieurs tables et de les considérer comme faisant partie d'un seul ensemble.

Cette fonctionnalité sera implémentée dans une future version de MySQL.

2.3.4. Les sous-requêtes

Une sous-requête consiste en un SELECT imbriqué dans un autre SELECT. Le résultat d'un SELECT peut être utilisé dans la clause WHERE. Cette possibilité est annoncée pour la version 4.1 de MySQL. En attendant leur disponibilité, un certain nombre de requêtes s'appuyant sur cette possibilité peuvent être réécrites sous la forme d'une jointure. Néanmoins, il est impossible de mettre en place, sous forme de jointure, des requêtes de type :

SELECT ... FROM ... WHERE *column_name* **NOT IN** (SELECT ... FROM ...)

Spécifier l'inexistence d'associations n'est pas faisable en utilisant les jointures. Ces dernières permettent uniquement de matérialiser des associations entre les lignes. L'absence de cette fonctionnalité implique l'absence de requêtes ayant des conditions de non-association.

2.3.5. Les transactions et les instructions commit/rollback

Une transaction est un ensemble d'instructions SQL exécutées comme une unité sans être interrompues par d'autres clients. Les fonctions de commit/rollback permettent d'établir si les instructions doivent être exécutées dans leur ensemble ou pas du tout. Cela signifie que si une instruction de la transaction échoue, toutes celles qui sont déjà exécutées seront annulées.

MySQL verrouille automatiquement les instructions SQL individuelles pour éviter qu'un client n'interfère avec un autre. De plus il est possible d'utiliser les instructions LOCK TABLES et UNLOCK TABLES pour grouper des instructions dans une même unité d'exécution, ce qui permet d'effectuer des opérations pour lesquelles le contrôle de concurrence d'accès au niveau de l'instruction individuelle n'est pas suffisant. Le problème de MySQL avec les transactions est que les instructions ne seront pas groupées automatiquement afin de pouvoir éventuellement les annuler lors d'un échec. Cela paraît primordial lors du déroulement d'un ensemble d'instructions critiques comme par exemple une transaction de paiement bancaire.

Une solution éventuelle pour obtenir un support transactionnel consiste à utiliser les tables de type InnoDB.

2.4. Problématique

Parmi les fonctionnalités manquantes à MySQL, l'absence de la fonctionnalité de gestion des clés étrangères et de l'intégrité référentielle va être prise en considération dans les chapitres suivants et ce mémoire va tenter de proposer une solution pour combler ce manque. Ne pas gérer les clés étrangères et l'intégrité référentielle peut provoquer des effets négatifs dans certains cas qui vont être exposés dans ce qui va suivre.

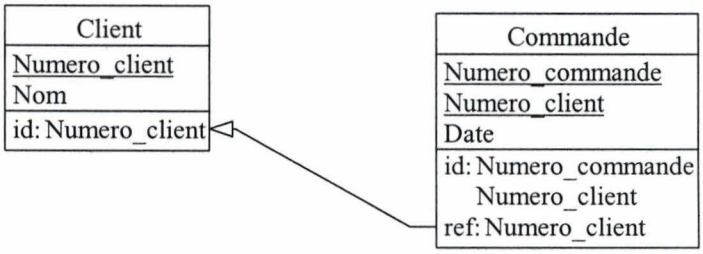


Figure 2.1. : Clé étrangère reliant deux tables

La figure 2.1 illustre un petit exemple significatif dans lequel deux tables sont reliées via une clé étrangère. L'absence de gestion des clés étrangères et de l'intégrité référentielle dans le SGBD MySQL peut produire dans cet exemple la situation suivante : Certaines commandes peuvent contenir un numéro de client qui n'est pas repris dans la table de référence *(la table Client)*. Il est à noter que : plus le nombre de clés reliant les tables est grand et plus les risques d'avoir une pareille situation sont élevés.

Quand les clés étrangères et l'intégrité référentielle sont prises en considération dans un SGBD, un contrôle est effectué lors des interactions avec les tables de la base de données. Les interactions nécessitant un contrôle sont :

- Les insertions : par exemple, dans le cas de la figure 2.1, lors de l'insertion d'une commande, une vérification va être effectuée afin de s'assurer que le numéro de client de la commande insérée correspond bien à un numéro repris dans la table des clients.

- Les suppressions : par exemple, dans le cas de la figure 2.1, si un client est supprimé de la table des clients, un contrôle va être réalisé pour vérifier si le client possède des commandes. Si des commandes sont associées au client à supprimer, le résultat de l'interaction sera différent en fonction de la stratégie mise en place. *(Les stratégies vont être expliquées par après).*

- Les mises à jour : deux types de contrôles vont être effectués. Premièrement, dans le cas de la figure 2.1, lors de la modification du numéro de client pour une commande, une vérification va se faire pour s'assurer que le nouveau numéro de client correspond bien à un numéro repris dans la table des clients. Deuxièmement, lorsque un numéro de client est modifié dans la table des clients, un contrôle va être effectué pour voir si des commandes existaient avec l'ancien numéro de client. Si c'est le cas, le résultat de l'interaction sera différent en fonction de la stratégie mise en place. *(Les stratégies vont être expliquées ci-après).*

Dans les cas de suppressions et de mises à jour, les stratégies évoquées précédemment sont les suivantes :

- La stratégie « No Action » qui consiste à refuser l'interaction si celle-ci altère l'intégrité des données.

- La stratégie « Cascade » qui répercute la nouvelle valeur ou qui supprime les enregistrements dans les tables qui référencent la table dans laquelle a eu lieu une

modification *(répercussion de la nouvelle valeur)* ou une suppression *(suppression des enregistrements).*

- La stratégie « Set null/default » qui consiste à mettre une valeur nulle ou une autre valeur définie par défaut pour les enregistrements dans les tables qui référencent la table dans laquelle a eu lieu une modification ou une suppression. *(Par exemple, dans le cas de la figure 2.1, si un client est supprimé et que celui-ci a des commandes, le numéro de client sur ses commandes aura la valeur « client supprimé »).*

Parmi ces différentes stratégies, la stratégie « No Action » est celle qui est sélectionnée pour la mise en place du module de gestion des clés étrangères présenté dans ce mémoire. La stratégie « No Action » est relativement simple à mettre en place par rapport aux deux autres qui sont plus complexes à déployer dans la mesure où l'utilisateur peut intervenir pour prendre une décision dans certains cas *(répercussion dans l'ensemble ou seulement dans une partie des tables qui font référence, choix de la valeur par défaut, ...).*

2.5. Comparatif avec d'autres SGBD

Le comparatif qui va suivre est extrait du site : http://www.mysql.com/information/crash-me.php et a été complété avec quelques critères supplémentaires. Sur ce site, il est possible de comparer MySQL avec d'autres SGBD. Le résultat fourni est neutre et très complet.

Pour les besoins de ce comparatif, seuls les principaux systèmes de gestion de bases de données et leurs caractéristiques générales ont été répertoriés. La norme SQL-92 figure également dans ce comparatif car cette norme définit le standard et c'est la version de la norme SQL qui est la plus utilisée.

Légende :

★ : Fonctionnalité supportée
⊗ : Fonctionnalité non supportée

	MySQL 4.0	Oracle 8	Access 2000	Microsoft SQL-server 2000	Norme SQL-2
OS	Linux, Windows, MacOS X et autres Unix	Linux, Windows, MacOS X et autres Unix	Environnement Windows	Environnement Windows NT	
Utilisation - besoins	Bases de données simples, gérées par un seul utilisateur en mise à jour, utilisées essentiellement pour des applications de consultation (génération de pages web dynamiques). Le coût est faible et pas besoin de formation spécifique.	Bases de données complexes, utilisées pour contenir des données sensibles dans de grandes organisations. Les bases de données Oracle supportent la restauration suite à un accident et permettent à de nombreux utilisateurs les mises à jour simultanées.	Ensemble d'outils suffisamment riches pour tout développeur de base de données expérimenté, tout en restant accessible aux utilisateurs débutants.	Solution complète de base de données et d'analyse conçue pour le développement de la prochaine génération de solutions de « data warehouse », d'applications métier et de commerce électronique.	Tous les systèmes de gestion de données utilisent SQL pour l'accès aux données ou pour communiquer avec un serveur de données. SQL (Structured Query Language), défini d'abord chez IBM au début des années 1970, a été formalisé par les standards SQL/86, SQL/89, SQL/92 **(SQL-2)** et SQL/99.
Philosophie	Logiciel Libre	Logiciel Propriétaire	Logiciel Propriétaire	Logiciel Propriétaire	Norme
Types de colonnes principaux					
char	★	★	★	★	★
date	★	★	★	⊗	★
float	★	★	★	★	★
int	★	★	★	★	★
varchar	★	★	★	★	★
bigint	★	⊗	⊗	★	★
tinyint	★	⊗	⊗	★	★
datetime	★	⊗	★	★	★
Autres types de colonnes					
blob	★	★	⊗	⊗	★
enum	★	⊗	⊗	⊗	⊗
set	★	⊗	⊗	⊗	⊗
long	⊗	★	★	⊗	⊗
text	★	⊗	★	★	★
varchar2	⊗	★	⊗	⊗	⊗

	MySQL 4.0	Oracle 8	Access 2000	Microsoft SQL-server 2000	Norme SQL-2
contraintes					
check	⊗ uniquement syntaxe	★	⊗	⊗	★
primary key	★	★	★	★	★
unique	★	★	★	★	★
foreign key	⊗ uniquement syntaxe	★	★	★	★
sous-requêtes	⊗	★	★	★	★
rename table	★	★	⊗	⊗	★
index	★	★	★	★	⊗
select avec limite	★	⊗	⊗	⊗	★
create table if not exist	★	⊗	⊗	⊗	★
drop table if exist	★	⊗	⊗	⊗	★
tables temporaires	★	⊗	⊗	⊗	★
select sans FROM	★	⊗	★	★	⊗
intersect	⊗	★	⊗	⊗	★
minus	⊗	★	⊗	★	★
union	★	★	★	★	★
procédures stockées	⊗	★	★	★	★
déclencheurs	⊗	★	⊗	★	⊗
vues	⊗	★	⊗	★	★
synonyme	⊗	★	⊗	⊗	★
destruction d'une colonne	★	★	★	★	★
renommage de colonne	★	★	★	★	★
modification type d'une col.	★	★	★	★	★
ajout contrainte not null	★	★	★	★	★
espace de stockage	⊗	★	⊗	⊗	⊗

Chapitre 3 : Méthodologie de développement de bases de données

Afin de pouvoir décrire le processus de conception d'une base de données dans un système spécifique de gestion de bases de données, ce chapitre va présenter la méthodologie générale de conception de bases de données qui consiste en une succession de plusieurs étapes qui vont être rappelées et décrites brièvement. Le texte est, en grande partie, extrait du cours d'ingénierie de bases de données exposé aux étudiants de licence en informatique des Facultés Universitaires Notre-Dame de la Paix à Namur [Hainaut, 2001].

La figure ci-dessous présente de manière visuelle le processus d'élaboration d'une base de données.

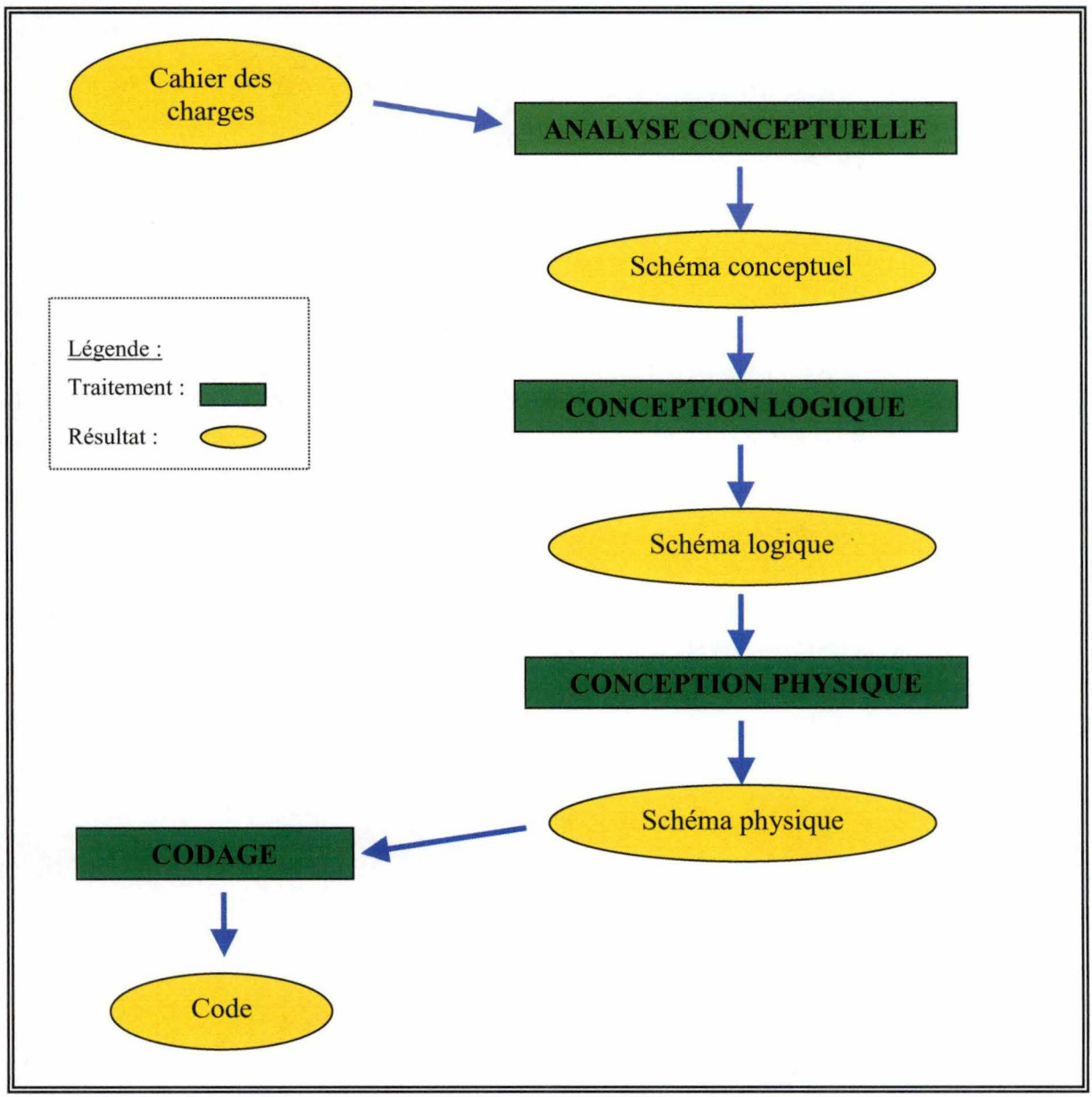


Figure 3.1. : Méthodologie de développement de bases de données

3.1. Point de départ

Comme illustré sur la figure 3.1, le point de départ est la rédaction du cahier des charges résultant de l'analyse des besoins exprimés par l'utilisateur. L'étude des besoins permet de délimiter le domaine d'application et d'élaborer un document dans lequel sont renseignés les services que le système doit offrir et la spécification des caractéristiques d'utilisation.

3.2. L'analyse conceptuelle

La phase d'analyse conceptuelle a pour but de produire une spécification formelle, complète, non ambiguë et non technique des informations. Le formalisme exploité est indépendant des technologies. Le résultat produit est le schéma conceptuel qui implémente le modèle Entité-Association détaillé dans le chapitre 4. Le schéma conceptuel sert de lien entre l'utilisateur et le concepteur de la base de données car le formalisme permet à l'utilisateur final de bien cibler le domaine d'application autrement que via le cahier des charges et permet, en même temps, à l'analyste de représenter de manière claire et rigoureuse les éléments à prendre en compte pour la suite du développement de la base de données.
Dans la phase d'analyse conceptuelle, le concepteur est amené à définir des entités caractérisant les objets du domaine d'application dignes d'intérêt et des associations définissant les liens unissant les entités entre elles. Des regroupements en types d'entités ou en types d'associations pourront être effectués s'il est nécessaire de représenter les objets de même type dans un même ensemble.
Dans le modèle Entité-Association se retrouvent également les concepts suivants :

- Les attributs représentant les propriétés des types d'entités.
- Les identifiants : un identifiant d'un type d'entités est un attribut ou un ensemble d'attributs dont la valeur permet de caractériser une et une seule instance de ce type d'entités.
- Les contraintes d'intégrité : ce sont les règles de gestion s'appliquant au domaine d'application et qui définissent les propriétés des objets, les configurations et les comportements. Les contraintes d'intégrité représentent également les propriétés formelles que les données doivent respecter à tout instant *(contraintes dynamiques)*, ou à des moments déterminés *(contraintes statiques)*.

Il est important de signaler qu'en plus d'être sémantiquement correct, le schéma conceptuel doit aussi respecter les critères de forme suivants :

- Minimalité : le schéma ne contient que des constructions pertinentes, à la mesure des concepts qu'il exprime.
- Simplicité et lisibilité : les concepts apparaissent clairement et naturellement dans le schéma.
- Expressivité : les constructions adéquates sont utilisées pour exprimer les concepts du domaine d'application.
- Elimination des redondances : chaque concept est représenté une seule fois. Il en est de même pour chaque occurrence des concepts.
- Conformité aux standards de l'organisation.

En respectant les conditions énoncées ci-dessus, le schéma conceptuel sera qualifié de « schéma conceptuel normalisé ».

3.3. La conception logique

L'étape suivant l'analyse conceptuelle est la conception logique qui a pour rôle de décrire les structures d'une base de données selon le modèle d'une famille de systèmes de gestion de bases de données mais sans évoquer de détails techniques propres à un SGBD précis. Les familles de systèmes de gestion de bases de données les plus courantes sont de type :

- Relationnels
- Orientés Objet
- Hiérarchiques
- Réseaux
- Fichiers

Cette phase aboutit à la mise en place du schéma logique. Celui-ci est équivalent au schéma conceptuel du point de vue du contenu informationnel mais les structures mises en place sont opérationnelles et conformes au modèle logique de l'environnement d'exploitation choisi. Lorsqu'on parle d'équivalence, cela signifie que le résultat est obtenu avec l'utilisation de transformations réversibles. Le schéma conceptuel subit donc un ensemble de transformations dans le but d'obtenir le schéma logique. Le modèle d'exploitation du SGBD MySQL est le modèle logique Relationnel.

Le modèle générique d'un schéma logique relationnel contient les éléments suivants :

- Les enregistrements ; agrégats logiques de données stockés et lus en une fois dans la base de données.
- Les types d'enregistrements définissant la composition de tous les enregistrements de ces types.
- Les champs ou les colonnes qui représentent les attributs définis dans l'analyse conceptuelle avec leurs caractéristiques.
- Les identifiants de types d'enregistrements représentant un groupe de champs tel qu'il ne peut exister, à tout instant, plus d'un enregistrement de ce type possédant les mêmes valeurs des champs du groupe.
- Les champs de référence de types d'enregistrements qui permettent de représenter les champs du type d'enregistrements vers lequel une association a été définie.

Il est important de signaler que la notion de schéma conforme à un modèle peut être de deux types en fonction du modèle logique choisi :

3.3.1. Le modèle strict

Un schéma logique est conforme au modèle strict s'il reprend des constructions étant directement et explicitement exprimables dans le langage de définition de données du SGBD qui mettra en œuvre le schéma.

3.3.2. Le modèle étendu

Un schéma logique est conforme au modèle étendu s'il reprend, en plus des constructions du modèle strict, des mécanismes annexes, appartenant ou non au SGBD.

3.4. La conception physique

La conception physique aboutit à la spécification des caractéristiques techniques d'implémentation de la base de données. Par rapport au schéma logique, le schéma physique engendré est enrichi de paramètres et constructions techniques offerts par le système de gestion de données.
Durant ce processus, le concepteur doit tenir compte de l'efficacité vis-à-vis des traitements effectués sur les données de la base.

Pour lui permettre de mener à bien cette tâche, les concepts suivants sont disponibles :

- La sélection des clés d'accès et des index. Ces notions favorisent les opérations d'accès et de mise à jour les plus fréquentes et les plus coûteuses.

- Le choix des espaces et des modes de stockages qui permettent de minimiser les accès physiques par la localisation des données dans les mêmes emplacements physiques.

Il est à noter que les techniques décrites ci-dessus se limitent aux bases de données relationnelles.

3.5. Le codage

La dernière étape à accomplir avant l'implantation et l'exploitation est le codage qui consiste à produire du code exécutable permettant de construire et de gérer une base de données d'un SGBD donné.

Le code peut être divisé en deux catégories complémentaires :

- Le langage de définition de données *(en anglais, Data Définition Language - DDL)* permet de déclarer explicitement certaines constructions du schéma physique.

- Les constructions additionnelles, quant à elles, traduisent des structures et des contraintes non déclarées de manière explicite via le DDL. Pour la mise en place de ces modules, le programmeur dispose de ces outils internes ou externes aux SGBD Relationnels :

    - Les prédicats de contraintes SQL *(check, assertions)*
    - Les procédures déclenchées *(triggers)*
    - Les vues filtrantes SQL
    - Les procédures stockées
    - Les modules d'accès
    - Le code distribué dans les programmes d'application
    - Les procédures de validation dans les écrans de saisie
    - Les programmes de validation avant/après chargement dans la base de données

Chapitre 4 : Conception d'une base de données MySQL

Ce chapitre va décrire le processus complet de conception d'une base de données MySQL en tenant compte des particularités de ce SGBD présentées dans le chapitre 2 et en appliquant la méthodologie de développement d'une base de données exposée dans le chapitre 3.
La première partie est relative à l'expression générique du modèle de données MySQL et les parties suivantes vont exposer les phases de conception logique, conception physique et de codage spécifiques à ce SGBD.

4.1. Expression du modèle de données en GER (notion de schéma conforme)

L'expression générique du modèle de données se fait de manière progressive. Le premier modèle à analyser est le modèle conceptuel. Par après, le modèle logique et plus particulièrement le modèle logique relationnel va être détaillé. Le dernier modèle de données est, quant à lui, le modèle physique propre à chaque SGDB. Dans cette étude, c'est donc le modèle physique relatif à MySQL qui sera exprimé.
Le cheminement à effectuer se fait de manière « verticale » car l'étude commence par le modèle le plus abstrait pour descendre vers des modèles de plus en plus concrets et également de façon « horizontale » vu qu'un choix doit être effectué lors des étapes plus concrètes. La progression horizontale est illustrée par la sélection du modèle relationnel lors de la deuxième phase et le choix du modèle propre à MySQL pour le modèle physique.

4.1.1. Le schéma conceptuel

Etant donné que l'analyse conceptuelle ne fait intervenir aucune technologie bien précise, le schéma conceptuel n'est donc contraint à aucune condition de représentation. Les concepts conformes pour représenter le domaine d'application sont cités dans cette liste :

- Des types d'entités. *(TE)*
- Des types d'associations. *(TA)*
- Des attributs avec leurs caractéristiques :
  - atomique : qui ne peut être fragmenté en composants significatifs.
  - décomposable : qui est constitué de valeurs plus élémentaires, ayant chacune une signification précise dans le domaine d'application.
  - monovalué : si à chaque occurrence de son parent ne peut être associée qu'une seule valeur.
  - multivalué : si plusieurs valeurs peuvent être associées à son parent.
  - obligatoire : si une valeur doit être obligatoirement associée à son parent.
  - facultatif : si le parent peut être démuni de valeurs.
- Des identifiants. Un identifiant peut être primaire lorsqu'il est constitué de composant(s) obligatoire(s). Il est secondaire dans le cas où au moins un de ses composants est facultatif.

- Des contraintes d'intégrité et plus spécialement :
  - Les contraintes de domaine qui exigent que les attributs respectent la définition de leur domaine de valeurs.
  - Les contraintes d'existence qui définissent les conditions de présence de composants d'une entité. Les quatre types de contraintes d'existence sont : coexistence, au moins un, exclusivité et exactement un.
  - Les contraintes de cardinalité qui représentent les bornes minimale et maximale que peut avoir un attribut ou un type d'associations.
  - Les contraintes d'inclusion ou d'exclusion de rôle.
  - Les contraintes de hiérarchie, étant également appelées « Relations IS-A ». Elles représentent différentes situations de spécialisations à l'intérieur d'un type d'entités plus général. Par exemple, un type d'entités « Personne » peut être spécialisé en deux sous-types d'entités « Homme » et « Femme ».

Exemple illustratif du schéma conceptuel

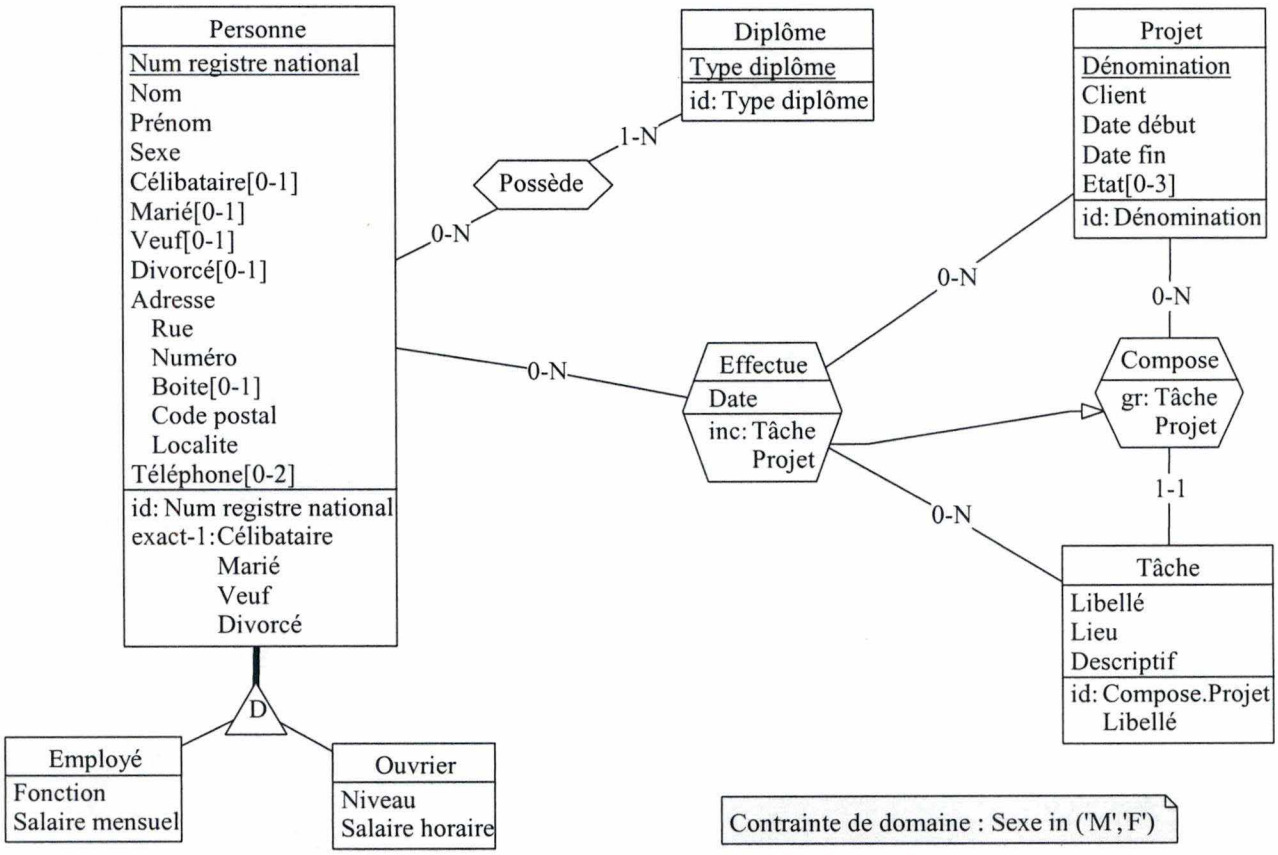


Figure 4.1. : Schéma conceptuel d'une base de données

Sur cette figure 4.1, Personne, Diplôme, Employé, Ouvrier, Projet et Tâche sont des types d'entités *(TE)* qui vont permettre de gérer une base de données relative à la mise en place de projets et à la réalisation des tâches qui le composent. Un TE possède des caractéristiques représentées à l'aide d'attributs de différents types.

Par exemple, pour le type d'entités « Personne », se retrouvent :

- des attributs atomiques : Num registre national, Nom, Prénom, Sexe, Célibataire, Marié, Veuf, Divorcé, Rue, Boîte, Code postal, Localité et Téléphone
- un attribut composé : Adresse
- des attributs facultatifs : Célibataire, Marié, Veuf, Divorcé et Boîte
- un attribut multivalué : Téléphone

Les attributs Num registre national, Nom, Prénom, Sexe, Rue, Code postal et Localité sont également des attributs monovalués et obligatoires.

Sur la figure 4.1, certains attributs sont aussi des identifiants. Par exemple, le numéro de registre national est unique pour chaque personne ou encore, la dénomination est unique pour chaque projet.

Dans ce schéma conceptuel, trois types d'associations *(TA)* relient des types d'entités. Le TA « Possède » relie les TE « Personne » et « Diplôme ». Les cardinalités de la relation expriment qu'une personne peut avoir un ou plusieurs diplômes et qu'un type de diplômes peut être attribué à une ou plusieurs personnes *(c'est une relation de type « plusieurs à plusieurs »)*. Le TA « Compose » relie le TE « Projet » avec le TE « Tâche ». Les cardinalités de la relation expriment qu'un projet est composé de 0 à plusieurs tâches et qu'une tâche appartient à un et un seul projet *(c'est une relation de type « un à plusieurs »)*. Le TA « Effectue » relie les trois TE « Personne », « Projet » et « tâche ». La relation est de type ternaire et contient aussi des informations spécifiques représentées sous forme d'attributs *(dans ce cas-ci, la date de réalisation de la tâche d'un projet par une personne est conservée)*.

Les contraintes d'intégrité évoquées précédemment sont illustrées également sur la figure 4.1 :

- Contrainte de domaine : représentée pour l'attribut « Sexe » par la définition d'un ensemble de valeurs autorisées *(« M » ou « F » et seulement ces deux valeurs)*.
- Contrainte d'existence : elle est représentée par le fait qu'une personne n'ait qu'une seule information relative à son état civil. La contrainte de type « exact-1 » signifie qu'une personne est soit célibataire, mariée, veuve ou divorcée.
- Contrainte de cardinalité : elle est représentée par le fait qu'une personne puisse avoir 0, 1 ou 2 numéros de téléphone, qu'une tâche appartienne à un et un seul projet ou encore qu'une adresse puisse contenir 0 ou 1 boîte.
- Contrainte d'inclusion de groupe : représentée par le fait qu'une tâche d'un projet effectuée par une personne doit être une tâche qui compose le projet.
- Contrainte de sous-type et de hiérarchie : entre les TE « Personne » et « Employé » et entre les TE « Personne » et « Ouvrier ». Cette contrainte signifie que tout employé et tout ouvrier est une personne et qu'une personne peut être soit un employé ou un ouvrier. Le « D » dans la hiérarchie représente une disjonction, ce qui signifie qu'une personne peut être soit un employé ou un ouvrier mais pas les deux en même temps. Il existe d'autres types de contraintes de sous-type comme par exemple, la partition, le recouvrement total et le recouvrement partiel.

4.1.2. Le schéma logique

MySQL fait partie de la famille des SGBD de type « Relationnels ». Le schéma logique conforme au modèle strict est donc représentatif de cette catégorie et inclut les objets suivants :

- Des tables : obtenues par spécialisation des types d'entités définis dans l'analyse conceptuelle
- Des champs ou colonnes : obtenus également par spécialisation des attributs des types d'entités du schéma conceptuel
- Des clés primaires ou uniques : représentant les identifiants primaires ou secondaires du schéma initial

Le schéma logique conforme au modèle étendu contient quant à lui, les concepts additionnels suivants :

- Des clés étrangères : équivalentes aux champs de référence utilisés par les types d'associations pour relier les types d'entités
- Des prédicats de type check : équivalents à la déclaration des divers types de contraintes

Exemple illustratif du schéma logique

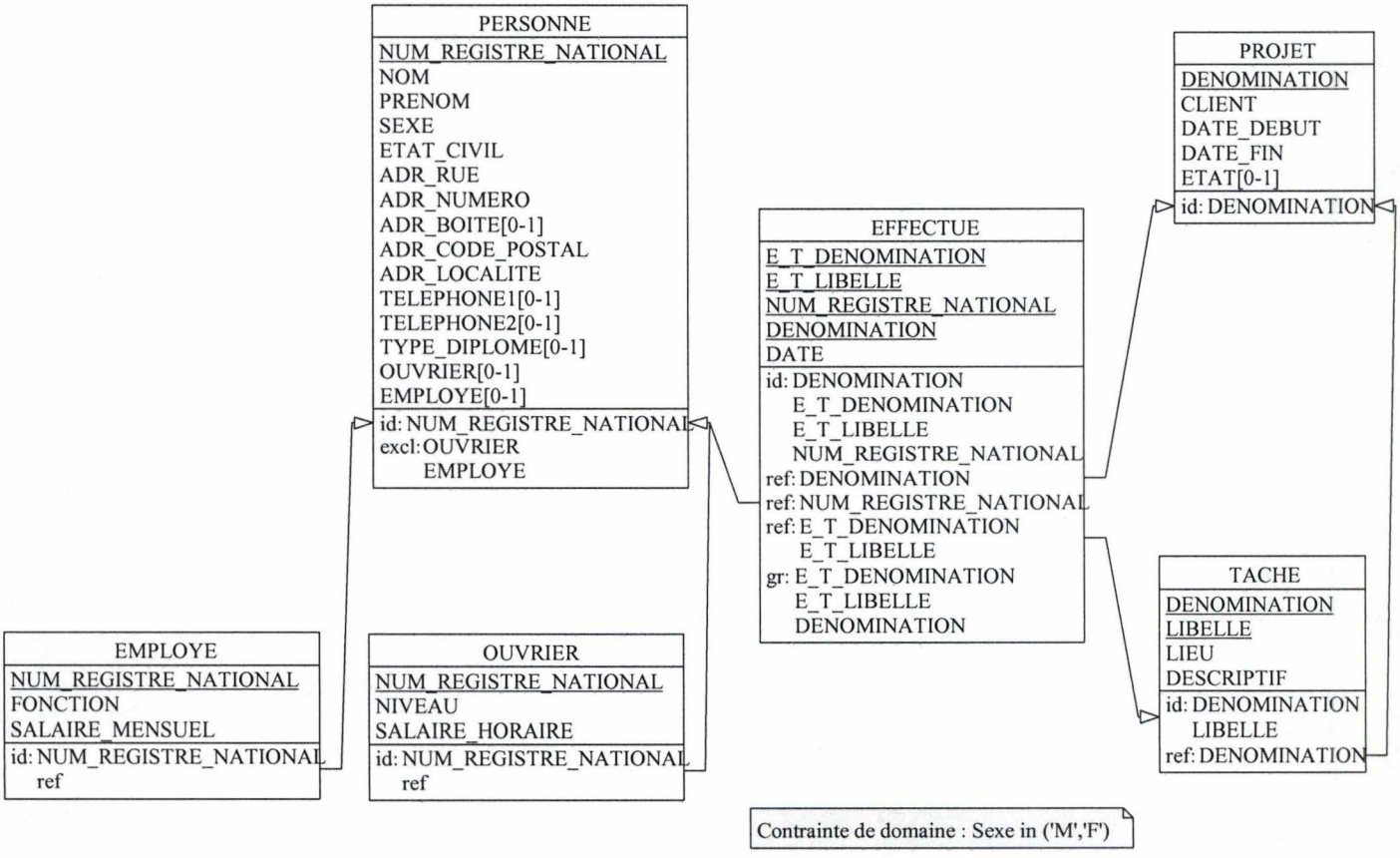


Figure 4.2. : Schéma logique relationnel d'une base de données

Les tables PERSONNE, EMPLOYE, OUVRIER, PROJET, TACHE et EFFECTUE se retrouvent à présent sur la figure 4.2. Cette figure est la représentation relationnelle de la figure 4.1. Les types d'entités, précédemment définis dans le schéma conceptuel, ont été spécialisés en table de même que le type d'associations complexe « Effectue ». Les attributs sont maintenant appelés des champs ou colonnes. L'attribut décomposable « Adresse » est désagrégé en un ensemble de champs atomiques et l'attribut multivalué « Téléphone » a été transformé en deux champs distincts et facultatifs. Les clés primaires *(NUM_REGISTRE_NATIONAL, DENOMINATION)* représentent, quant à elles, les identifiants définis dans le schéma conceptuel.

Il est à noter également que les différentes valeurs de l'état civil d'une personne qui étaient spécifiées chacune séparément sur la figure 4.1 sont maintenant regroupées dans un seul champ appelé « ETAT_CIVIL » et ce champ est de type « ENUM ». Il en est de même pour le champ mutlivalué « ETAT » d'un projet qui est transformé sur la figure 4.2 en un champ de type « SET » car, dans le domaine d'application de cet exemple, un projet ne comporte qu'un nombre limité d'états qui sont les suivants : Prospection, Développement et Maintenance. Enfin, le TE « Diplôme » représenté sur la figure 4.1 n'est plus repris sur la figure 4.2 mais il a été remplacé par un champ *(TYPE_DIPLOME)* de type « SET » dans la table PERSONNE sur la figure 4.2. Les transformations vers ces deux types de données spécifiques à MySQL sont exposées dans la seconde partie de ce chapitre.

Concernant la partie logique relative au modèle étendu, la figure 4.2 illustre les clés étrangères reliant les tables. Ces clés étrangères sont représentatives des types d'associations « un à plusieurs » définis auparavant.
Les contraintes de type « Check » sont également présentes. Celle concernant le domaine de valeurs de la colonne « Sexe » est représentée par une annotation et la contrainte d'exclusivité entre Employé et Ouvrier est définie dans la table PERSONNE.

Le détail complet du plan de transformation permettant d'aboutir à ce schéma logique relationnel à partir du schéma conceptuel et le détail des transformations exploitant le potentiel des types de données « SET » et « ENUM » dans MySQL sont décrits dans la partie « Conception Logique » de ce chapitre.

4.1.3. Le schéma physique

La conformité du schéma physique est propre à l'environnement de MySQL. Par rapport au schéma logique, il sera enrichi de constructions permettant d'optimiser les accès et les performances.

Pour cela, MySQL propose :

- Des index. Un index contient une entrée pour chaque ligne de la table et les valeurs de l'index sont triées. Cette technique permet de gagner du temps en évitant de devoir parcourir l'ensemble de la table lors de la recherche d'une valeur.

- Plusieurs types de stockage permettant de stocker les données à des emplacements bien spécifiques. Les données peuvent être stockées en mémoire ou sur le disque dur.

Exemple illustratif du schéma physique

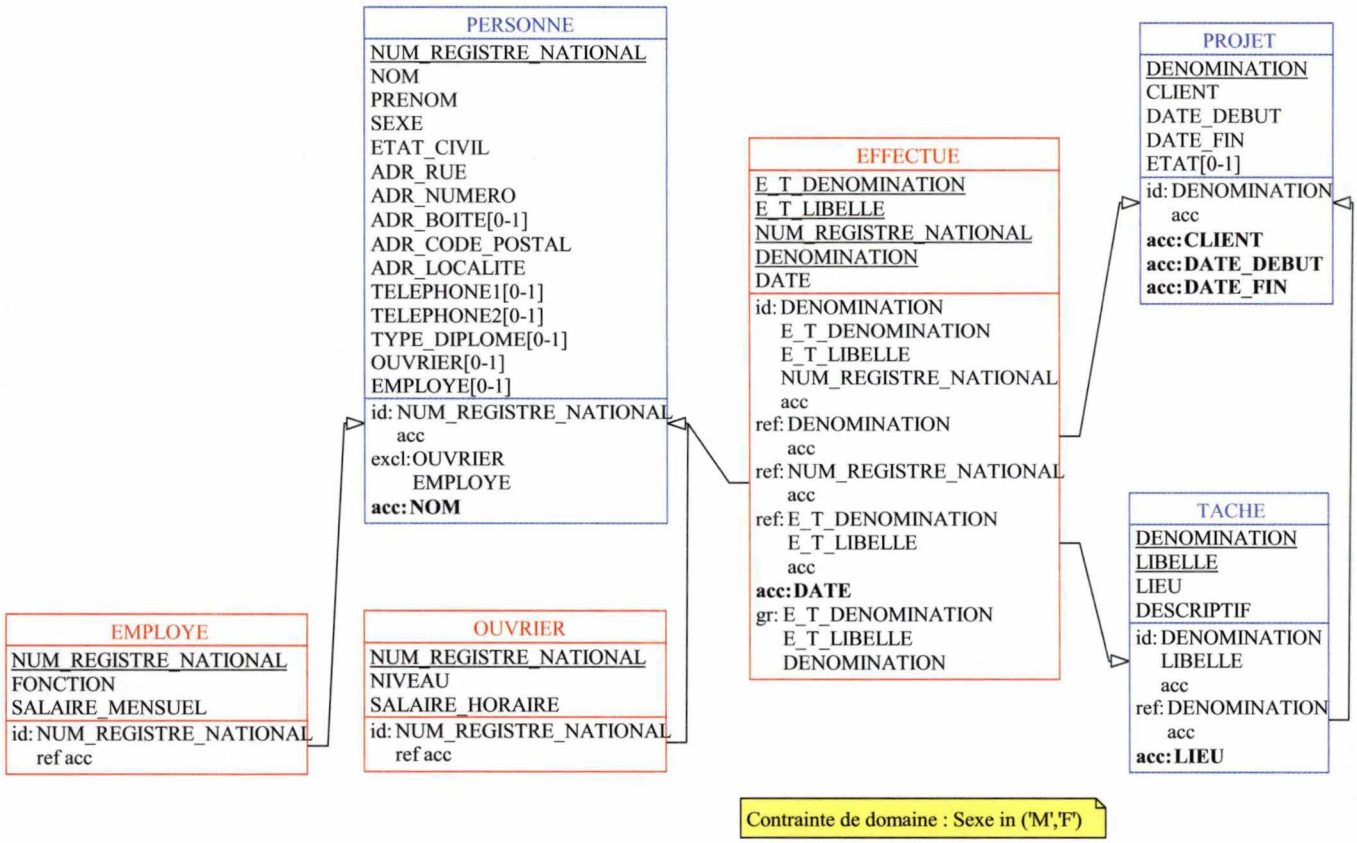


Figure 4.3. : Schéma physique MySQL d'une base de données

La figure 4.3 représente le schéma physique. Ce schéma est le schéma logique auquel sont ajoutés les concepts d'index et de types de stockage. Les index définis sur le schéma sont repérables par leur mention « acc ». Les index définis pour des raisons de performances lors de recherches ou de mises à jour sont représentés en gras sur la figure 4.3. En effet, il n'est pas rare que dans la base de données de gestion de projets, il faille effectuer des recherches sur une personne en utilisant comme paramètre son nom ou encore, effectuer des statistiques sur les tâches effectuées sur une journée ou sur base d'un lieu bien précis.
Les types de stockage sont représentés sur la figure 4.3 par l'utilisation de couleurs. Les tables bleues sont les tables de type « MyISAM » qui stockeront les données sur le disque dur et les tables rouges sont les tables qui contiendront leurs données en mémoire *(ce sont les tables de type « Heap »)*.
Les règles, heuristiques et les avantages de ces notions physiques sont mentionnées dans la partie « Conception physique » de ce chapitre 4.

4.2. Conception logique (plan de transformation)

Pour rappel, la conception logique consiste à produire un schéma logique en faisant appel à un ensemble de transformations réversibles appliquées au schéma conceptuel.

4.2.1. Transformations

Une transformation consiste à manipuler un schéma pour changer la représentation de l'information tout en veillant à ne pas modifier le contenu. Les transformations effectuées pour obtenir le schéma logique sont réversibles, ce qui signifie qu'il est toujours possible de retrouver le schéma conceptuel compte tenu que la sémantique reste constante.
Le plan de transformation permettant le passage du schéma conceptuel vers un schéma logique relationnel comporte les opérations suivantes :

1. Transformer les relations IS-A en types d'associations « un à un » et ajouter les contraintes éventuelles.
2. Transformer les types d'associations complexes, c'est-à-dire les types d'associations avec des attributs ou ayant des cardinalités de type « plusieurs à plusieurs » en types d'entités. Cela a pour conséquence de produire des types d'associations « un à un » ou « un à plusieurs » entre les types d'entités.
3. Transformer les attributs composés monovalués de niveau 1, les attributs composés sont désagrégés.
4. Transformer les attributs multivalués de niveau 1, les attributs multivalués peuvent être transformés en types d'entités ou alors, ils peuvent être transformés en un attribut de type « SET » à condition que le domaine de valeurs associé à cet attribut soit relativement restreint et qu'il soit identique pour chaque attribut. Un exemple de cette transformation est repris sur la figure 4.2 *(l'état d'un projet)* et la description détaillée de cette transformation est reprise dans la section suivante.
5. Tant qu'il existe des attributs composés ou multivalués, réitérer les opérations 3 et 4.
6. Transformer les listes d'attributs facultatifs monovalués booléens en un attribut de type « SET » ou « ENUM » en fonction de la nature de la contrainte qui les relie. Une illustration de cette transformation est reprise sur la figure 4.2 *(l'état civil d'une personne)* et le détail de cette transformation se retrouve dans la section suivante.
7. Transformer les types d'entités jouant un rôle de cardinalité 1-N et contenant un seul attribut ayant les particularités suivantes :
   - obligatoire,
   - monovalué,
   - de type « chaîne de caractères »,
   - identifiant,
   - ayant un domaine énuméré,

   en un attribut de type « SET » ou « ENUM » en fonction des cardinalités de la relation unissant ce type d'entités à un autre. Un exemple de cette transformation est repris sur la figure 4.2 *(les diplômes associés à une personne)* et la description détaillée de cette transformation est reprise dans la section suivante.

8. Transformer les types d'associations « un à un » ou « un à plusieurs » en clés étrangères.
9. Si nécessaire, ajouter des identifiants techniques et réitérer l'opération 8, car il se peut que certains types d'associations ne puissent être transformés parce que les types d'entités cibles n'ont pas d'identifiant.
10. Transformation des identifiants facultatifs, chacun d'eux est extrait pour former un type d'entités autonome relié au type d'entités source par un type d'associations « un à un ». Ce dernier est ensuite transformé en clé étrangère.
11. Finalisation, mise en conformité des noms au standard du modèle *(syntaxe, mots réservés, ...)*, domaine de valeurs des colonnes, ...

L'exemple illustrant l'emploi de ce plan de transformation pour générer le schéma logique relationnel à partir du schéma conceptuel est repris sur les figures 4.1 et 4.2.

4.2.2. Transformations logiques propres à MySQL

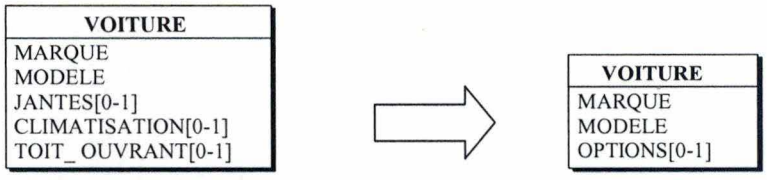
Etant donné que MySQL propose des types de domaines particuliers tels que les types « SET » et « ENUM », il est possible de profiter du potentiel de ces types de données lors de la phase de conception logique. Les transformations qui vont être détaillées dans cette section ont été intégrées dans le plan de transformation repris dans la section précédente.

1) Transformation d'une liste d'attributs en un seul attribut de type « SET » ou « ENUM »

Une table peut contenir une collection d'attributs qui forment ensemble une liste de choix possibles. Ces attributs sont de type booléen, sont facultatifs et sont également monovalués. En fonction de la contrainte d'existence qui relie ces différents attributs, cet ensemble peut alors être converti soit en un attribut de type « SET » contenant à lui seul cette liste de valeurs et dont il est possible d'en choisir une ou plusieurs ou soit en un attribut de type « ENUM » contenant également à lui seul cette liste de valeurs mais qui n'autorise la sélection que d'une seule valeur.

Exemple 1.1.

Le type d'entités A de la figure 4.4 contient trois attributs facultatifs, monovalués et de type booléen qui ne sont contraints à aucune condition d'existence. Ces trois éléments peuvent être regroupés en un seul attribut de type « SET facultatif » qui est appelé « OPTIONS » dans le type d'entités B. Le champ « OPTIONS » contiendra l'ensemble de valeurs des options possibles lors de sa déclaration. Pour chaque enregistrement de la table, le champ « OPTIONS » contiendra les valeurs spécifiques à cet enregistrement.

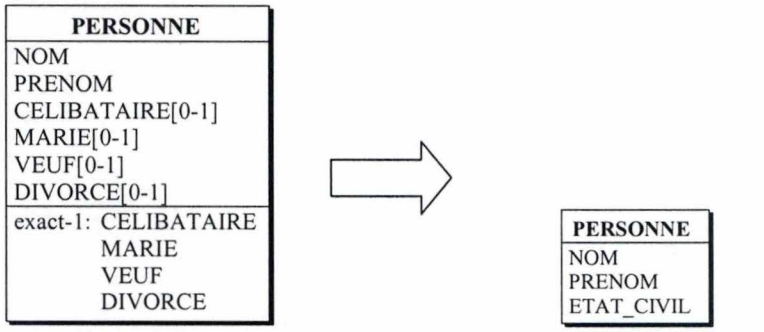


Type d'entités A — Type d'entités B

Figure 4.4. : Transformation d'une liste d'attributs en un seul attribut de type « SET »

Exemple 1.2

Le type d'entités A de la figure 4.5 contient quatre attributs facultatifs, monovalués et de type booléen qui sont unis par une contrainte de type « exactement un ». Ces attributs peuvent être fusionnés en un seul attribut de type « ENUM obligatoire » qui est appelé « ETAT_CIVIL » dans le type d'entités B. Le champ « ETAT_CIVIL » contiendra l'ensemble de valeurs des états civils possibles lors de sa déclaration. Pour chaque enregistrement de la table, le champ « ETAT_CIVIL » contiendra la valeur spécifique à cet enregistrement.



Type d'entités A Type d'entités B

Figure 4.5. : Transformation d'une liste d'attributs en un seul attribut de type « ENUM »

Le tableau ci-dessous va synthétiser les transformations de listes d'attributs reliés ou non par une contrainte d'existence en un attribut de type « SET » ou « ENUM ». Ces contraintes définissent également la présence facultative ou obligatoire qu'il faudra prendre en compte lors de la déclaration de ce nouvel attribut.

La pré-condition est la suivante : un groupe d'attributs monovalués, facultatifs et de type booléen est inclus dans un type d'entités.

La post-condition est la suivante : le groupe d'attributs a été transformé en un seul attribut de type « SET » ou « ENUM », obligatoire ou facultatif en fonction de la contrainte définie sur le groupe d'attributs.

Contrainte définie sur le groupe	**Attribut généré par la transformation**
Pas de contrainte	SET facultatif
« Au moins un »	SET obligatoire
« Exclusivité »	ENUM facultatif
« Exactement un »	ENUM obligatoire

2) Transformation d'un type d'entités en un attribut de type « SET » ou « ENUM »

Si un type d'entités joue un rôle de cardinalité 1-N et ne contient qu'un seul attribut ayant les caractéristiques suivantes :

- obligatoire,
- monovalué,
- de type « chaîne de caractères »,

- identifiant,
- ayant un domaine énuméré,

ce type d'entités peut alors être remplacé par un attribut dans le type d'entités qui référence cet ensemble de valeurs. En fonction du rôle de la cardinalité joué par le type d'entités qui recevra l'attribut, le type du nouveau champ est « SET » ou « ENUM » et est facultatif ou obligatoire.
Cette transformation a pour avantage d'obtenir une expressivité adéquate pour le type de bases de données MySQL.

Exemple 2.1.

La représentation A de la figure 4.6 contient deux types d'entités reliés par une relation de type « plusieurs à plusieurs ». Dans cet exemple, un produit peut avoir de 0 à plusieurs labels. Le type d'entités « LABEL » est conforme aux critères énoncés dans le paragraphe précédent. Il est donc possible de le transformer en un attribut dans le type d'entités « PRODUIT ». Dans la représentation B, cet attribut est déclaré de type « SET facultatif» et contient l'ensemble des labels attribués à un produit.

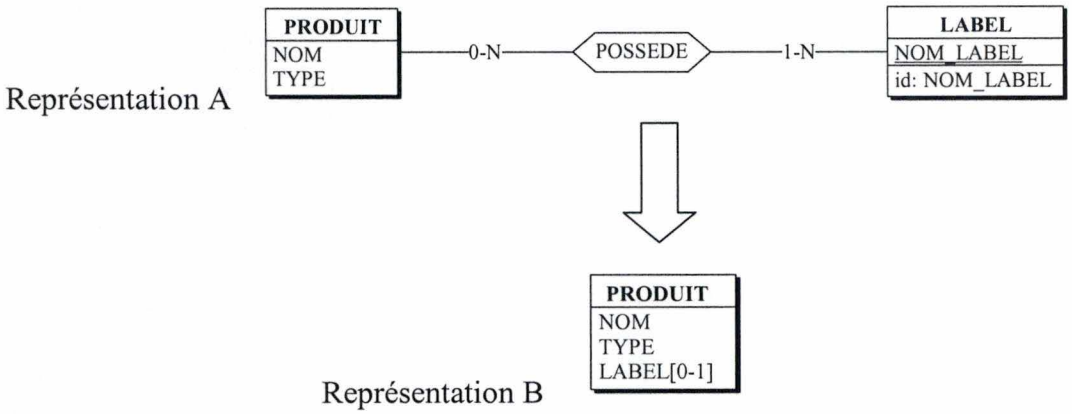


Figure 4.6. : Transformation d'un type d'entités en un attribut de type « SET »

Exemple 2.2.

La représentation A de la figure 4.7 contient 2 types d'entités reliés par une relation de type « un à plusieurs ». Dans cet exemple, une voiture consomme un et un seul type de carburant. Le type d'entités « CARBURANT » est conforme aux critères permettant la transformation d'un type d'entités en un attribut. Il est donc possible de le transformer en un attribut dans le type d'entités « VOITURE ». Dans la représentation B, cet attribut est déclaré de type « ENUM obligatoire ». Avec un attribut de ce type, il n'est possible de choisir qu'une valeur de carburant par voiture.

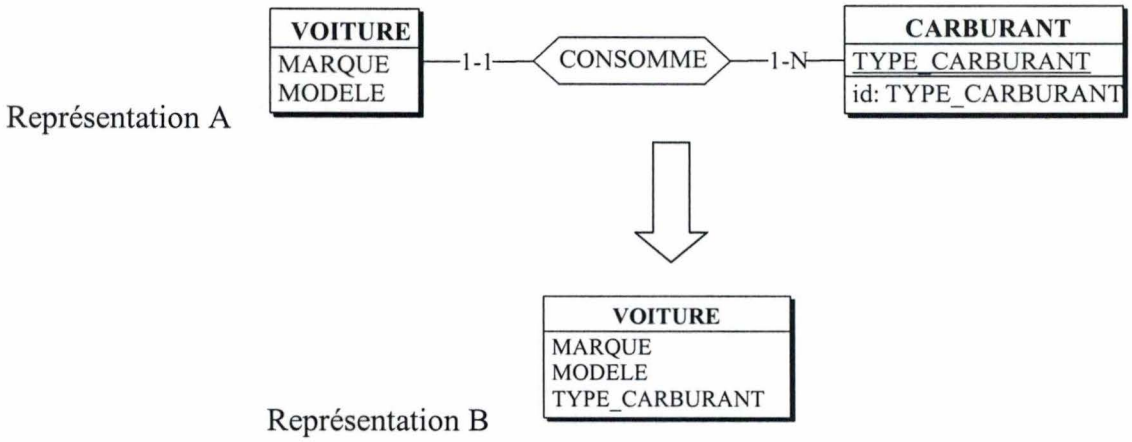


Figure 4.7. : Transformation d'un type d'entités en un attribut de type « ENUM »

Le tableau ci-dessous va synthétiser les transformations de types d'entités en attribut de type « SET » ou « ENUM ». En fonction du rôle de la cardinalité joué par le type d'entités dans lequel sera inséré l'attribut, l'attribut généré est soit de type « SET » ou « ENUM » et est facultatif ou obligatoire.

Les pré-conditions sont les suivantes : un type d'entités joue un rôle de cardinalité 1-N et ne contient qu'un seul attribut ayant les caractéristiques suivantes :

- obligatoire,
- monovalué,
- de type « chaîne de caractères »,
- identifiant,
- ayant un domaine énuméré.

La post-condition est la suivante : le type d'entités de la pré-condition est transformé en un attribut. En fonction du rôle de la cardinalité joué par le type d'entités dans lequel l'attribut sera inséré, le type du nouveau champ est « ENUM » ou « SET » et est facultatif ou obligatoire.

Rôle de la cardinalité joué par le type d'entités dans lequel l'attribut sera inséré	**Attribut généré par la transformation**
0-N	SET facultatif
1-N	SET obligatoire
0-1	ENUM facultatif
1-1	ENUM obligatoire

3) Transformation d'un attribut multivalué en un attribut de type « SET »

Si un attribut est multivalué et que son domaine de valeurs est restreint et identique pour chaque enregistrement, il est possible alors de le transformer en un attribut de type « SET ». L'exemple repris sur les figures 4.1 et 4.2 est relatif à l'état d'un projet et illustre cette transformation. En effet, le domaine d'application de l'exemple stipule qu'un projet peut être caractérisé par une succession d'états. Un projet est d'abord en cycle de prospection, ensuite, il entre en phase de développement et enfin, il passe en maintenance. Tous les projets de cette base de données ont donc tous le même

ensemble de valeurs d'états. Cette situation permet donc de transformer cet attribut multivalué en un attribut de type « SET ». Lors de la déclaration du champ « ETAT » dans la phase de codage de la base de données, le domaine de valeurs *(« Prospection », « Développement » et « Maintenance »)* sera associé au type « SET ».
En fonction de la cardinalité minimale de l'attribut multivalué, le résultat de cette transformation est soit un attribut de type « SET obligatoire » dans le cas où l'attribut multivalué est de cardinalité 1-N, soit un attribut de type « SET facultatif » si l'attribut multivalué est de cardinalité 0-N.

Le tableau ci-dessous va synthétiser les transformations d'un attribut multivalué en un attribut de type « SET ». En fonction de la borne minimale de la cardinalité, l'attribut est facultatif ou obligatoire.

Les pré-conditions sont les suivantes : un attribut dans un type d'entités est multivalué et son domaine de valeurs est limité et identique pour chaque enregistrement.

La post-condition est la suivante : l'attribut multivalué est transformé en un attribut de type « SET » et est facultatif ou obligatoire en fonction de la borne minimale de l'attribut.

Bornes de l'attribut multivalué	**Attribut généré par la transformation**
0-N	SET facultatif
1-N	SET obligatoire

4.2.3. Conventions des noms

MySQL élargit les conventions concernant les noms de bases de données, de tables, de colonnes et d'index. Les conventions sont présentées ci-après :

- Les noms doivent comporter un maximum de 64 caractères
- Il est possible d'utiliser n'importe quel caractère alphabétique ou numérique, y compris le dollar ($) et le caractère souligné (_)
- Un nom peut commencer par un caractère numérique mais ne peut être composé uniquement de caractères numériques
- Les noms ne peuvent pas être des mots réservés de MySQL. il est à noter que la liste des mots réservés de MySQL est moins exhaustive que celle définie par la norme ANSI SQL
- L'acceptation de certains caractères *(les accents par exemple)* est dépendante du système d'exploitation utilisé

Pour des raisons de portabilité vers des environnements différents *(systèmes d'exploitation ou SGBD)*, il est déconseillé d'utiliser des noms réservés de la norme ANSI SQL et d'utiliser des caractères spéciaux comme par exemple, les accents.

4.3. Conception physique (règles et heuristiques)

MySQL propose plusieurs techniques afin d'optimiser l'efficacité des traitements qui seront effectués sur la base de données.
Un exemple illustrant l'utilisation de ces techniques est présenté sur la figure 4.3.

4.3.1. Les index et les clés d'accès

Les règles concernant les clés primaires et secondaires sont les suivantes :

- Lors de la déclaration d'une clé primaire, un index ayant pour nom « PRIMARY KEY » est créé.
- Lors de la déclaration d'une clé secondaire, un index ayant pour nom « UNIQUE » est créé.
- Seules les colonnes des index PRIMARY KEY doivent obligatoirement être déclarées NOT NULL.

Compte tenu de la création implicite des clés primaires et secondaires en index lors de leur déclaration, le concepteur devra uniquement se préoccuper d'indexer les colonnes présentant un intérêt pour les opérations de sélections, de groupements, ou encore d'accès lors d'une insertion ou d'une mise à jour. Toutefois, il doit veiller à ne pas déclarer une clé d'accès qui serait le préfixe d'une autre clé d'accès.

4.3.2. Les types de stockage

Le choix du mode de stockage s'effectue en fonction de la nature des besoins. Ces derniers peuvent être relatifs à la rapidité ou à la sécurité d'une table. Une table de type « HEAP », dont les données sont uniquement stockées en mémoire, est très rapide mais a un niveau de sécurité très faible. Par exemple, lors d'un accident ou d'un redémarrage du serveur, les données seront perdues. Par contre, une table de type « InnoDB » apporte la sécurité des données en offrant, par exemple, le support transactionnel. Il est évident qu'un tel type de tables est consommateur de plus de ressources. Le type de tables par défaut proposé par MySQL est le type MyISAM qui offre un niveau intermédiaire entre sécurité et rapidité.

4.4. Codage

4.4.1. Les constructions déclarées

Le code DDL de MySQL reprend en grande partie la syntaxe définie par la norme SQL Standard. Il est donc possible de créer explicitement les éléments suivants :

- La base de données
- Les tables
- Les colonnes avec leurs spécificités
- Les clés primaires
- Les index

En plus de ces concepts, il est également possible de déclarer dans le DDL de MySQL des concepts typiques tels que le type de stockage de la table.

Ci-dessous, quelques fragments de code générés à partir de la figure 4.3 :

Création de la base de données
L'instruction permettant de créer la base de données relative à la figure 4.3 est :

```
CREATE DATABASE GESTION_DE_PROJETS ;
```

Création des tables

```
CREATE TABLE PERSONNE
      (NUM_REGISTRE_NATIONAL    char(10) NOT NULL,
       NOM                      char(20) NOT NULL,
       PRENOM                   char(20) NOT NULL,
       SEXE                     char(1) NOT NULL,
       ETAT_CIVIL               enum ('Célibataire','Marié',
                                      'Veuf','Divorcé'),
       ADR_RUE                  char(20) NOT NULL,
       ADR_NUMERO               char(4) NOT NULL,
       ADR_BOITE                char(4),
       ADR_CODE_POSTAL          char(6) NOT NULL,
       ...
       PRIMARY KEY (NUM_REGISTRE_NATIONAL)) ;


CREATE TABLE EMPLOYE
      (NUM_REGISTRE_NATIONAL    char(10) NOT NULL,
       FONCTION                 char(10) NOT NULL,
       SALAIRE_MENSUEL          int(5) NOT NULL,
       PRIMARY KEY (NUM_REGISTRE_NATIONAL))
       Type = HEAP ;
```

Les champs obligatoires sont déclarés « NOT NULL » et la clé primaire est directement déclarée lors de la création de la table *(PRIMARY KEY)*.
Le type par défaut d'une table MySQL est le type MYISAM et il n'est pas obligatoire de le déclarer. Par contre, lorsque le type d'une table n'est pas celui par défaut, il est nécessaire de le déclarer explicitement *(dans la table EMPLOYE, il faut la déclarer de type HEAP)*.

Création des index
Le code permettant de créer les index de la figure 4.3 est le suivant

```
CREATE INDEX INDEX_1 ON PERSONNE (NOM) ;
CREATE INDEX INDEX_2 ON EFFECTUE (DATE) ;
...
```

Il est à signaler que les identifiants dans un SGBD MySQL sont implicitement des index et il n'est pas nécessaire de les redéfinir en tant que tel une seconde fois.

Déclaration des clés étrangères

La syntaxe de l'instruction permettant de déclarer une clé étrangère de la table TACHE vers la table PROJET de la figure 4.3 est la suivante :

```
ALTER TABLE TACHE ADD CONSTRAINT FKCOMPOSE
        FOREIGN KEY (DENOMINATION)
        REFERENCES PROJET;
```

Déclaration des contraintes de type « check »

Par exemple, pour le champ SEXE, étant donné que son domaine de valeurs est limité, il faudra ajouter cette instruction :

```
ALTER TABLE PERSONNE ADD CONSTRAINT CK_PERSONNE
        CHECK SEXE IN ('M','F') ;
```

Il est important de signaler que les déclarations des clés étrangères et des contraintes de type « check » sont exprimables dans le code DDL de MySQL. Cependant, ces contraintes exprimées ne seront pas gérées dans les versions actuelles de MySQL.

4.4.2. Constructions exprimées de manière procédurale

Les constructions exprimées de manière procédurale permettent de mettre en oeuvre tout ce que MySQL ne permet pas de gérer nativement.
Parmi les constructions additionnelles qui peuvent améliorer la qualité de la base de données, se retrouvent :

- La mise en place de la gestion des clés étrangères *(qui va être proposée dans les chapitres suivants de ce mémoire)*
- La validation des contraintes de type « check »
- La mise en application des règles de gestion

Les outils proposés par MySQL pour réaliser ces procédures sont actuellement peu nombreux. En effet, MySQL n'offre pas les possibilités de procédures stockées, de déclencheurs et de vues. Actuellement, le programmeur devra plutôt employer des programmes externes au SGBD pour construire ces nouvelles fonctionnalités.

Chapitre 5 : Développement d'une base de données MySQL avec l'atelier de génie logiciel DB-MAIN

Ce chapitre est destiné aux personnes impliquées dans la conception et le développement des bases de données et va illustrer la mise en œuvre complète d'une base de données MySQL présentée dans le chapitre 4 avec l'atelier logiciel DB-MAIN. Les différentes parties détaillées dans ce qui va suivre sont relatives à la description générale de DB-MAIN et à son utilisation lors des différentes phases de développement.

Les deux premières sections de ce chapitre sont extraites du tutorial de présentation et du manuel d'utilisation de l'atelier logiciel DB-MAIN [DB-MAIN 2003].

5.1. Description de l'atelier de génie logiciel DB-MAIN

5.1.1. Présentation générale

DB-MAIN est un environnement de génie logiciel, c'est-à-dire, un ensemble de logiciels destinés à l'ingénierie des bases de données et organisés autour d'un composant centralisé dans lequel se trouvent les données partagées et que l'on appelle « référentiel ». DB-MAIN a pour objectif d'apporter une aide dans les principaux processus de développement et de maintenance de bases de données tels que :

- Le processus de développement (top-down) : analyse des besoins, analyse conceptuelle, normalisation, intégration de schémas, conception logique, conception physique, optimisation de schémas et génération de code.
- Les transformations : transformations de schémas, transformations de modèles.
- La rétro-ingénierie et la compréhension de programmes (bottom-up) : analyse de schémas, analyse de code, rétro-ingénierie de données.
- La maintenance, l'évolution et l'intégration : migration de données, évolution de bases de données, intégration et fédération de bases de données, conception et génération de wrappers de données.
- Les bases de données spéciales : conception de bases de données spatiales, évaluation et génération de bases de données actives.
- L'ingénierie XML.

5.1.2. Particularités

- DB-MAIN offre un modèle de données à large spectre. Ce modèle reprend les concepts des principaux modèles conceptuels *(ERA, UML, NIAM)*, les modèles logiques modernes *(relationnel, Orienté Objet, XML)* et anciens *(CODASYL, IMS, fichiers COBOL)* et les modèles physiques correspondants.
- DB-MAIN est méthodologiquement neutre. Il admet plusieurs méthodes de travail sans imposer de contraintes sur les processus ni sur les produits.
- DB-MAIN dispose d'un moteur méthodologique qui aide les utilisateurs à suivre des méthodes d'ingénierie tout en maintenant un historique des opérations.
- DB-MAIN offre des modèles et des outils de rétro-ingénierie permettant de retrouver les spécifications de la plupart des bases de données réelles, depuis les fichiers COBOL jusqu'aux bases de données relationnelles et XML.

- DB-MAIN offre une collection d'assistants destinés à aider les développeurs dans les tâches complexes ou fastidieuses *(transformations de schémas, conception physique, analyse de code, ...)*.
- DB-MAIN est programmable. Il est possible pour l'utilisateur de construire de nouveaux processeurs ou des méthodes avec les langages de programmation Voyager ou Java. Cela est dû à l'ouverture de DB-MAIN permettant d'accéder au référentiel de données. Grâce à ces possibilités de développement, des extensions peuvent être développées au sein de l'équipe du laboratoire d'ingénierie des bases de données mais aussi par les utilisateurs eux-mêmes.

5.2. Description du langage de programmation « Voyager »

Dans le cadre de ce mémoire, le langage de programmation Voyager va être utilisé pour l'élaboration des modules d'extension de DB-MAIN. La description qui va suivre permet de mettre en évidence les particularités de ce langage de programmation.
Le langage de programmation Voyager est un langage permettant d'exécuter des tâches dynamiques sur le référentiel de l'atelier logiciel DB-MAIN dans lequel se trouvent les données persistantes et les données volatiles.
Le processus de mise en place d'un script développé en Voyager se fait en respectant les étapes suivantes :

- Ecriture du code dans un fichier ayant l'extension « .V2 »
- Compilation
- Le module généré ayant l'extension « .OXO » est utilisable dans l'atelier DB-MAIN via la commande « Execute Voyager ».

Voyager est un langage de programmation impératif de type Pascal / C. Il permet les actions suivantes :

- La gestion de listes
- L'analyse lexicale
- Les requêtes prédicatives du référentiel
- L'extension dynamique du référentiel
- Les librairies dynamiques
- L'interaction avec DB-MAIN

Les apports du langage Voyager pour DB-MAIN sont :

- L'extension fonctionnelle de l'atelier
- Le paramétrage sophistiqué des assistants
- Les fonctions d'importation et d'exportation
- Les librairies génériques

De plus amples informations concernant le langage de programmation Voyager peuvent être obtenues à la référence suivante : [Englebert 1999].

5.3. Analyse conceptuelle

La phase d'analyse conceptuelle consiste à produire un schéma qui est conforme au modèle Entité-Association et qui est représentatif du domaine d'application décrit dans le cahier des charges. Comme cela est expliqué dans les chapitres précédents, le schéma Entité-Association généré à la fin de cette phase contient des types d'entités, des types d'associations, des attributs et diverses contraintes.

5.3.1. Création du schéma conceptuel

Dans l'atelier logiciel DB-MAIN, l'analyste a devant lui tous les utilitaires lui permettant de mettre en place un schéma conceptuel.

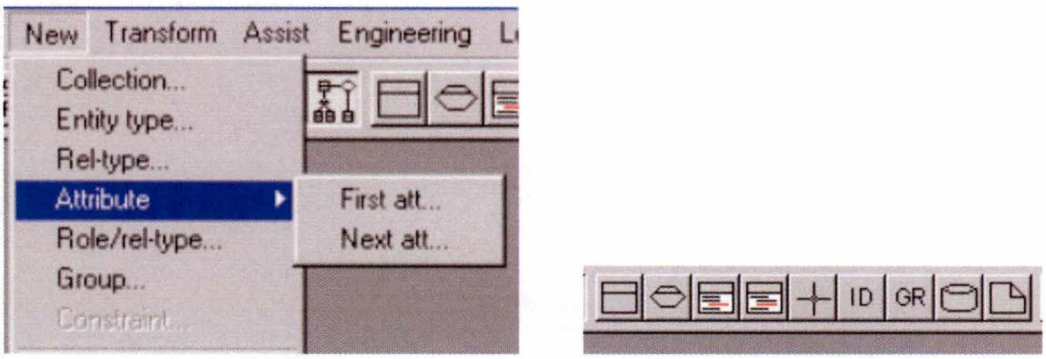


Figure 5.1. : Menu et palette d'outils permettant la phase d'analyse conceptuelle dans DB-MAIN

Comme le montre la figure 5.1, il est possible de représenter le domaine d'application à l'aide d'un menu ou d'une palette d'outils mis à disposition. Cet ensemble de fonctionnalités permet de définir d'abord les types d'entités et ensuite, d'y insérer les différents attributs avec leurs diverses caractéristiques.
Par après, pour relier les types d'entités, le concepteur de la base de données doit faire appel à la fonctionnalité permettant de déclarer les types d'associations et doit préciser leurs cardinalités.
Lorsque les types d'entités, les attributs et les types d'associations sont déterminés, il est possible d'ajouter dans le schéma conceptuel les notions suivantes :

- Les identifiants primaires ou secondaires. Pour les définir, il suffit de sélectionner le ou les attributs faisant partie de l'identifiant et d'utiliser ensuite le bouton « ID » de la barre d'outils pour la validation.

- Les contraintes d'intégrité. Elles sont matérialisées sous forme de propriétés spécifiques à un type d'entités, à un type d'attributs ou à un type d'associations en fonction de la nature de la contrainte. Si, malgré tout, il n'est pas possible de déclarer de façon explicite une contrainte, il reste toujours l'éventualité de faire appel à la fonction d'annotation dans laquelle sera définie textuellement la contrainte à prendre en considération.

Un exemple complet de schéma conceptuel réalisé avec DB-MAIN est illustré sur la figure 4.1 du chapitre 4.

5.4. Conception logique

La conception logique a pour rôle de mettre en place un schéma dans lequel est élaborée la structure d'une base de données selon le modèle d'une famille de SGBD.
Pour obtenir un tel schéma, il est nécessaire de faire appel à un ensemble de transformations. Celles permettant d'obtenir un schéma logique relationnel sont présentées dans le chapitre 4. Une transformation est une opération qui consiste à remplacer un ensemble d'objets du schéma conceptuel par un ensemble d'objets conduisant à un schéma conforme, ici, en l'occurrence, à un schéma logique relationnel. La représentation change mais le contenu informationnel reste identique.
Pour exécuter ces transformations, le concepteur de la base de données dispose de plusieurs assistants. Parmi ceux-ci, le premier propose une méthode générale de conception d'un schéma logique relationnel en ne tenant compte d'aucune optimisation, le second permet de résoudre les problèmes structurels généraux et le dernier permet, en plus, d'affiner la portée de certaines transformations grâce à des procédures de contrôle.
Etant donné que l'atelier logiciel DB-MAIN est méthodologiquement neutre, rien n'empêche également de mettre en place un schéma logique de manière manuelle.
Dans les sections qui vont suivre, la mise en place des types « SET » et « ENUM » de MYSQL est décrite de même que les différents assistants de transformations et l'assistant de traitement des noms.

La façon de procéder pour l'obtention d'un schéma logique relationnel d'une base de données MySQL en utilisant les outils présentés dans cette partie est exposée dans la section 8.1.3 de l'étude de cas dans le chapitre 8.

5.4.1. Paramétrisation des nouveaux types « SET » et « ENUM » dans DB-MAIN

Création des domaines de valeurs « SET » et « ENUM »

La définition des nouveaux types « SET » et « ENUM » de MySQL dans DB-MAIN se réalise en utilisant un type de données spécifique de format « User Defined Domain ». Pour mettre en place ces deux types, le concepteur de la base de données doit ajouter, dans la fenêtre de définition de domaines personnalisés, les deux types « SET » et « ENUM » comme illustré sur la figure 5.2. Il est à noter qu'il est possible de créer une procédure écrite avec le langage de programmation Voyager qui permettrait de paramétrer ces types de données de façon automatique.

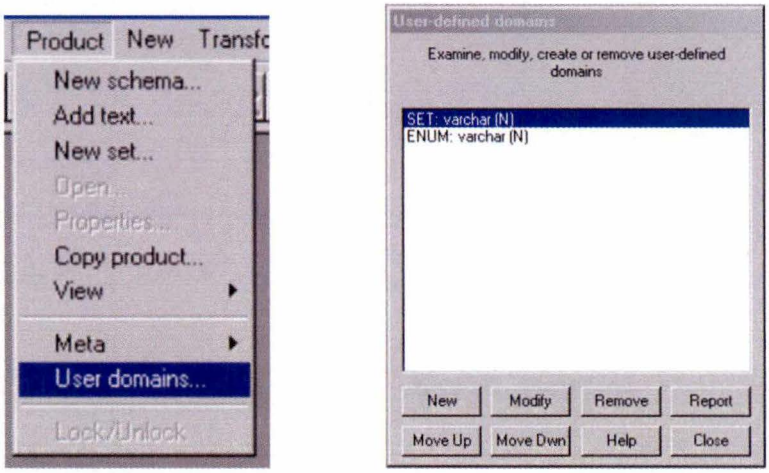


Figure 5.2. : Création des domaines de valeurs « SET » et « ENUM »

Déclaration de colonnes de type « SET » et « ENUM » dans DB-MAIN

Dans l'atelier logiciel DB-MAIN, les opérations suivantes doivent être accomplies pour déclarer un de ces deux types à une colonne :

- Dans un premier temps, sélectionner le type « User-defined » pour le champ considéré et choisir le type « SET » ou « ENUM » comme représenté sur la figure 5.3.

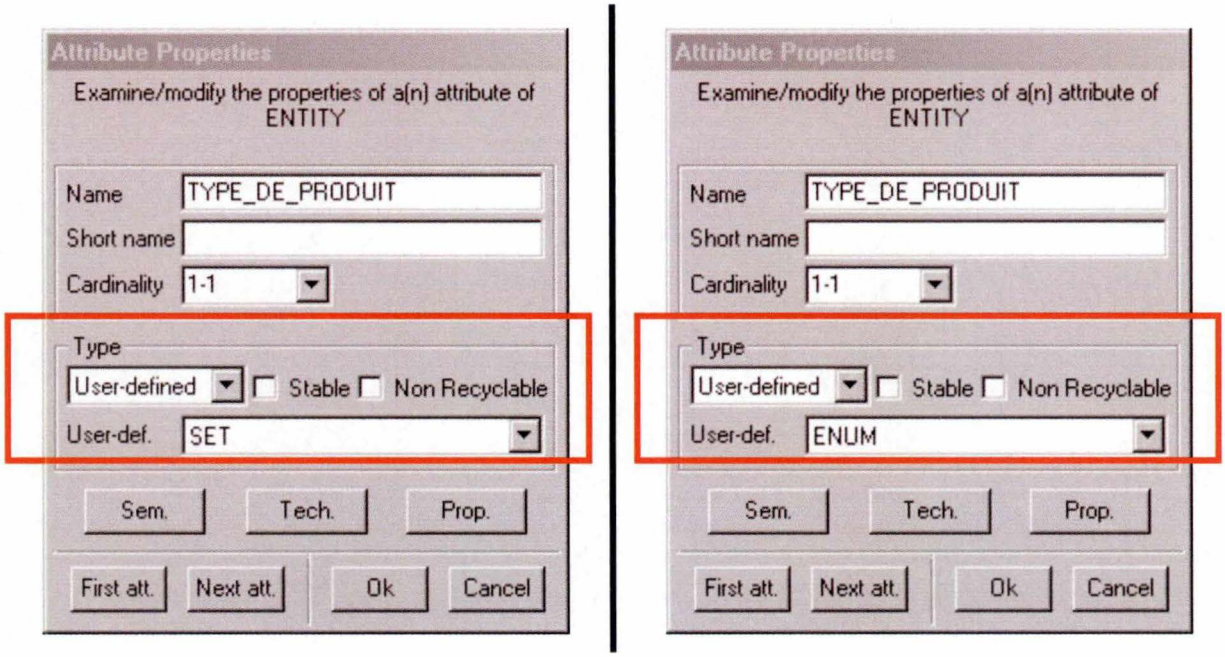


Figure 5.3. : Déclaration du type « SET » et « ENUM»

- Ensuite, éditer la fenêtre de propriétés dans laquelle il est possible de modifier les propriétés dynamiques de l'attribut *(le bouton « Prop. » sur la figure 5.3)*. Pour la définition des valeurs associées aux types « SET » et « ENUM », la propriété dynamique dénommée « Value constraint » est utilisée et les informations relatives à cette propriété contiennent donc l'ensemble des valeurs qui lui sont associées. Le format représenté sur les figures 5.4 et 5.5 est le suivant :

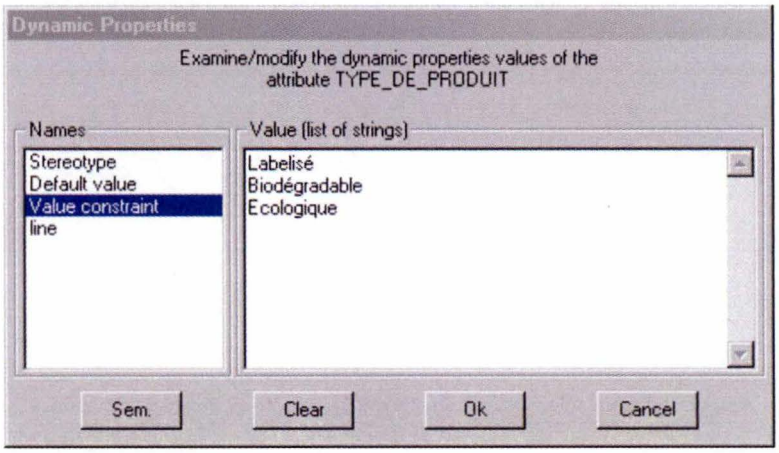


Figure 5.4. : Création de la liste de valeurs d'un champ de type « SET »

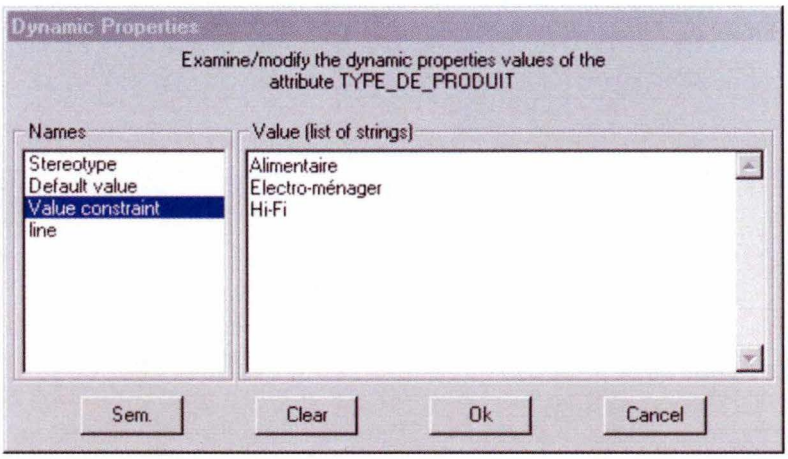


Figure 5.5. : Création de la liste de valeurs d'un champ de type « ENUM »

Les trois sections qui vont suivre vont présenter, par ordre d'intérêts, les différents outils proposés dans l'atelier logiciel DB-MAIN qui permettent au concepteur d'une base de données d'effectuer plus efficacement la phase de conception logique.

5.4.2. Fonctionnalité de génération automatique du modèle relationnel

Cette fonctionnalité est utilisée pour remplacer le schéma conceptuel courant par une version logique relationnelle sans effectuer d'optimisations. L'appel de cette fonction s'effectue via l'instruction « Relational model » reprise dans le Menu « Transform » de DB-MAIN *(illustré sur la figure 5.6)*. Etant donné que MySQL fait partie des SGDB de type « Relationnels », le concepteur de la base de données peut avoir recours à cette fonctionnalité et obtenir de la sorte, un schéma logique relationnel. Cependant, ce schéma logique est simple et non optimisé. Cet outil est utile par exemple, lors des phases de prototypages.

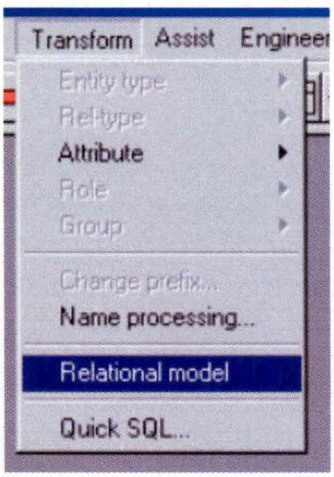


Figure 5.6. : Fonctionnalité « Relational model » du menu « Transform »

5.4.3. Assistant de transformations globales

L'assistant de transformations globales a pour objectif d'aider le concepteur à résoudre des problèmes de nature structurelle en utilisant les notions de transformations. Pour se faire, cet assistant est utilisé afin de repérer ces problèmes sur un schéma ou alors, est utilisé dans le but d'effectuer une ou plusieurs transformations qui remplacent les structures problématiques par une construction équivalente.
La figure 5.7 montre les différentes fonctionnalités de l'assistant de transformations globales. A droite de l'écran, se trouve un catalogue de transformations standard dans lequel le concepteur de la base de données choisit les transformations qu'il juge nécessaires pour la conception logique. Il a la possibilité de regrouper toutes les transformations qu'il souhaite réaliser dans un script qui se constitue à gauche de l'écran. Quand le concepteur a mis en place son plan de transformation, il peut rendre effectives ses transformations et il a la possibilité de sauvegarder son script pour une réutilisation ultérieure sur d'autres schémas.
Il est à noter que DB-MAIN propose des scripts de transformations prédéfinis pour faciliter cette tâche de composition du schéma logique.

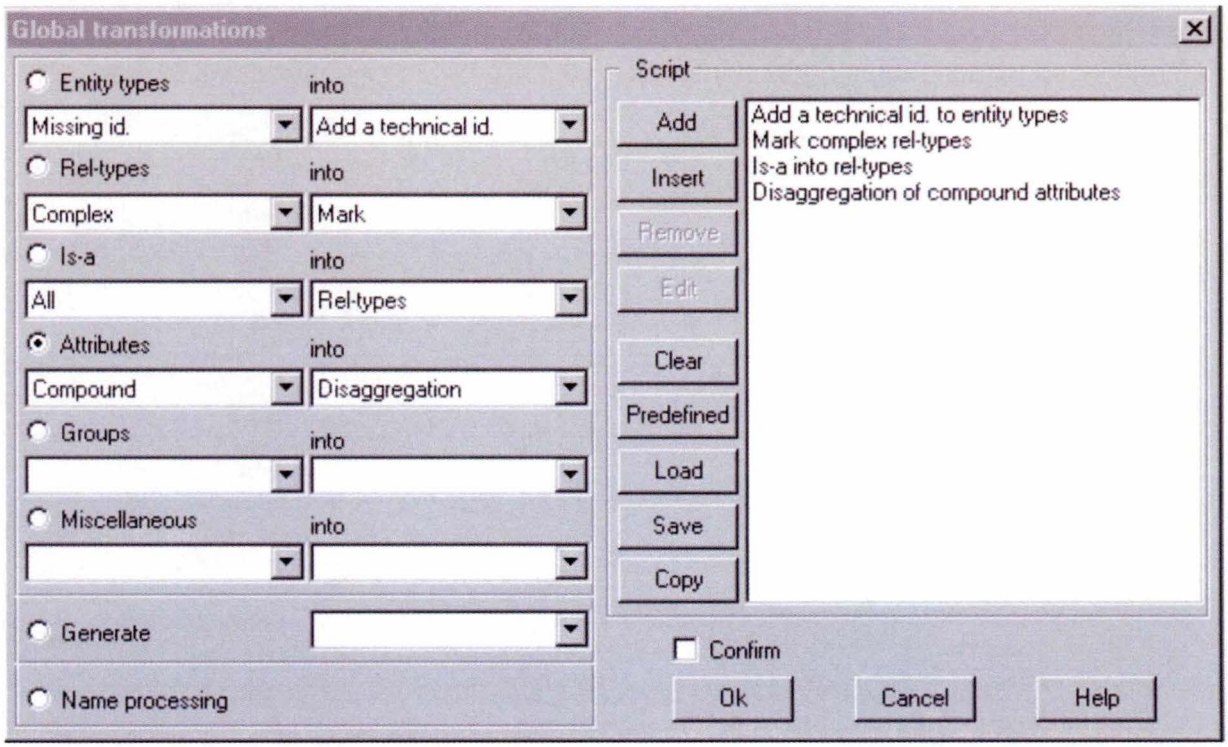


Figure 5.7. : L'assistant de transformations globales

Remarque : il n'est pas possible d'insérer des transformations développées en Voyager dans un script mis en place avec cet assistant. L'assistant de transformations globales ne gère que les transformations qui se trouvent dans son catalogue. Pour ajouter des transformations particulières développées en Voyager, le concepteur devra alors utiliser l'assistant de transformations globales avancées qui est décrit dans la section suivante.

5.4.4. Assistant de transformations globales avancées

L'assistant de transformations globales avancées est une version plus sophistiquée de l'assistant de transformations globales présenté précédemment. Il permet, en effet, plus de flexibilité et plus de puissance dans le développement de scripts grâce aux structures de contrôle et à la possibilité de réduire la portée d'une transformation via la mise en place de règles d'analyse de schéma.
De plus, un répertoire de transformations globales avancées peut être élaboré et réutilisé dans la définition de nouvelles transformations. Il est également possible d'importer, dans cet assistant, des transformations externes personnalisées écrites en Voyager.

Les figures 5.8 et 5.9 exposent les deux fenêtres relatives à cet assistant.

La figure 5.8 représente la partie où le script de transformations globales avancées peut être rédigé. A gauche de cet écran, se retrouvent le catalogue des primitives de transformations, les différentes structures de contrôles et la librairie des transformations avancées définies par l'utilisateur.
La figure 5.9 représente, quant à elle, la fenêtre d'édition de règles permettant de spécifier la portée de la transformation définie. Dans l'exemple illustré *(figure 5.9),* la transformation d'un attribut multivalué en un type d'entités s'effectuera en respectant la règle définissant la portée de la transformation aux attributs ayant une cardinalité supérieure à deux éléments.
Lorsque le script de transformations globales avancées est complet, le concepteur peut le sauvegarder pour une réutilisation ultérieure.

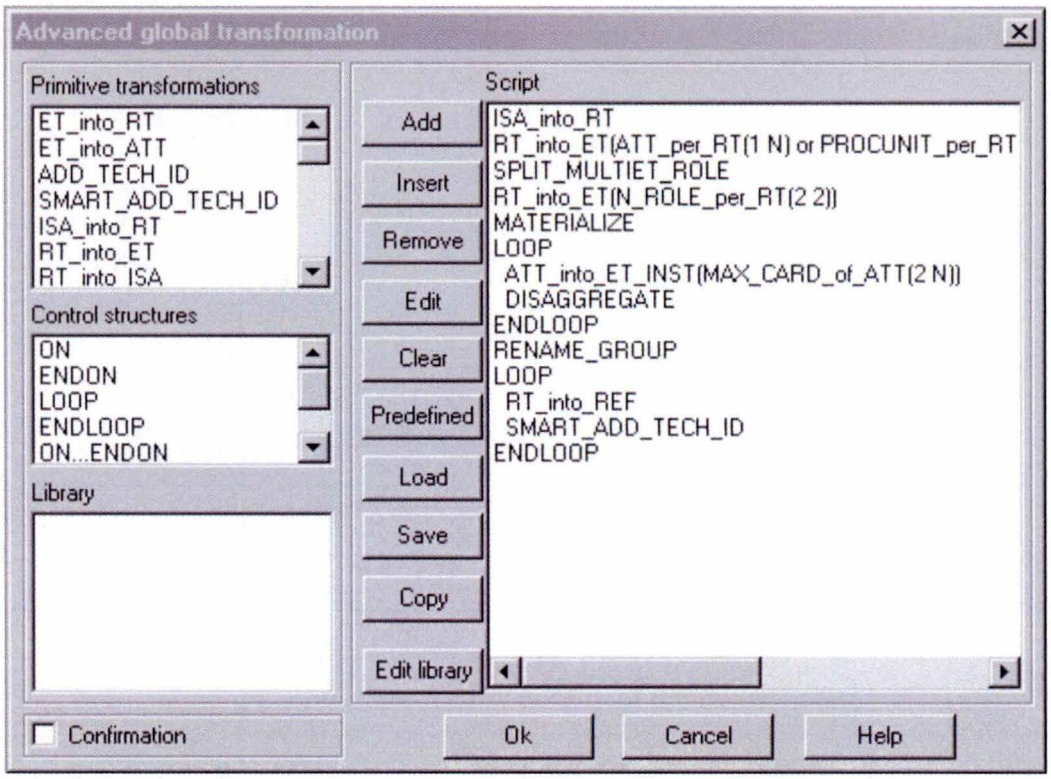


Figure 5.8. : L'assistant de transformations globales avancées

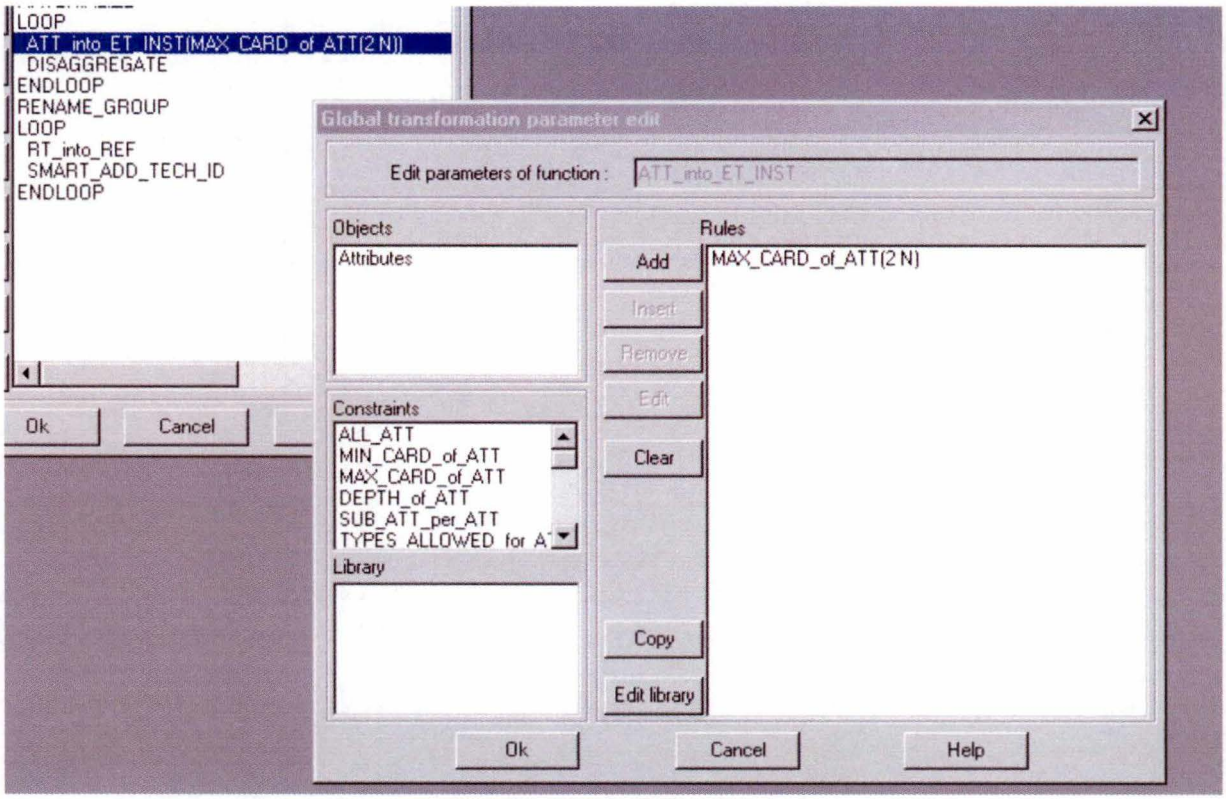


Figure 5.9. : Fenêtre de définition de règles d'analyse d'objets

Concernant la mise en place du schéma logique d'une base de données MySQL, lorsque le concepteur se réfère aux transformations particulières présentées dans la section 4.2.2, l'assistant de transformations globales avancées est employé pour importer les trois transformations de schémas qui permettent d'exploiter le potentiel des types de données « SET » et « ENUM ». Ces transformations sont rédigées avec le langage de programmation Voyager et sont les suivantes :

- La première transforme un attribut multivalué en un type d'entités ou en un attribut de type « SET » si cet attribut répond aux critères définis dans la section 4.2.2.
- La seconde substitue un groupe d'attributs facultatifs booléens reliés ou non par une contrainte en un attribut de type « SET » ou « ENUM » en fonction des contraintes exposées dans la section 4.2.2.
- La troisième transforme un type d'entités jouant un rôle de cardinalité 1-N et contenant un seul attribut identifiant obligatoire de type « chaîne de caractères » en un attribut de type « SET » ou « ENUM » en fonction du rôle de cardinalité joué par le type d'entités dans lequel sera inséré l'attribut. Cette transformation est également décrite dans la section 4.2.2.

La figure 5.10 illustre le plan de transformation du chapitre 4 inséré dans l'assistant de transformations globales avancées et illustre également la marche à suivre pour importer une fonction d'un script généré avec le langage de programmation Voyager. Pour cela, il suffit d'ajouter une primitive de type « EXTERN » et de renseigner le fichier dans lequel se trouve la fonction écrite en Voyager, le nom de la fonction qui doit être prise en compte et le filtre éventuel réduisant le champ d'action de la transformation.

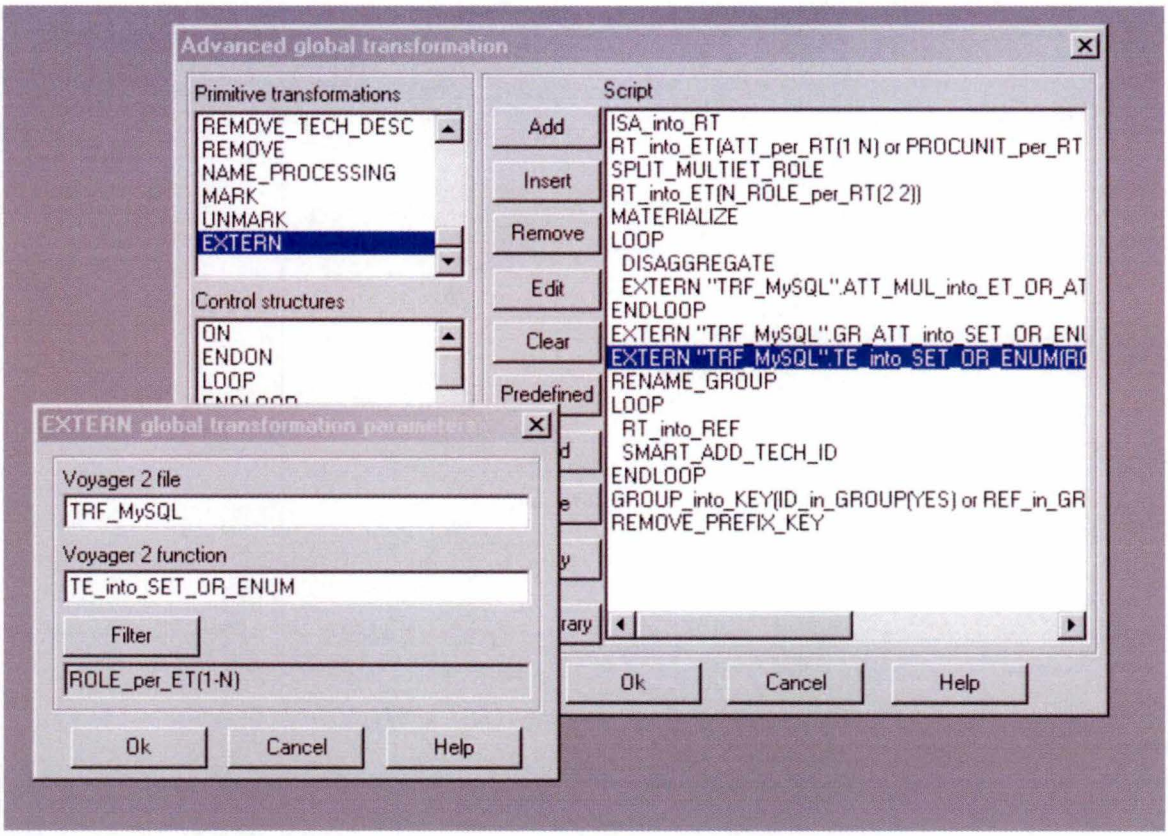


Figure 5.10. : Fenêtre de définition de fonctions mises en place par l'utilisateur

5.4.5. Traitement des noms

Dans le but d'obtenir une syntaxe conforme à la convention des noms définie par le SGBD, cette fonctionnalité permet de formater le nom des objets se trouvant dans le schéma. Par exemple, grâce à cette possibilité, il est possible de mettre en lettres capitales tous les noms des champs des tables, de retirer les caractères accentués ou encore de limiter la taille. L'ensemble des opérations proposées est repris sur la figure 5.11.

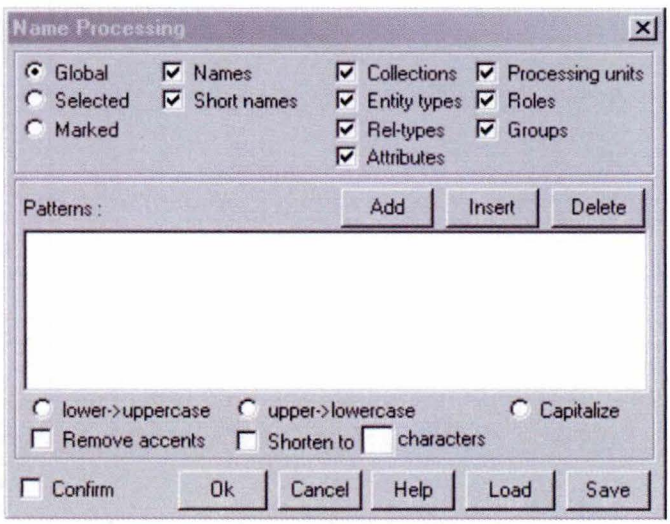


Figure 5.11. : Ecran de traitement des noms

5.5. Conception physique : spécification des paramètres physiques

La conception physique permet de spécifier des caractéristiques techniques d'implémentation de la base de données en fonction des constructions techniques offertes par le SGBD MySQL.

5.5.1. Définition des clés d'accès

La définition des clés d'accès s'effectue sur un composant de type « groupe » contenant un ou plusieurs attributs. Comme illustré sur la figure 5.12, la déclaration de la clé d'accès est prise en compte lorsque le concepteur de la base de données sélectionne la fonction « Access key » sur la fenêtre des propriétés. Pour la définition des clés d'accès, il est possible également d'utiliser les assistants présentés dans les sections précédentes pour une définition effectuée de façon automatisée.

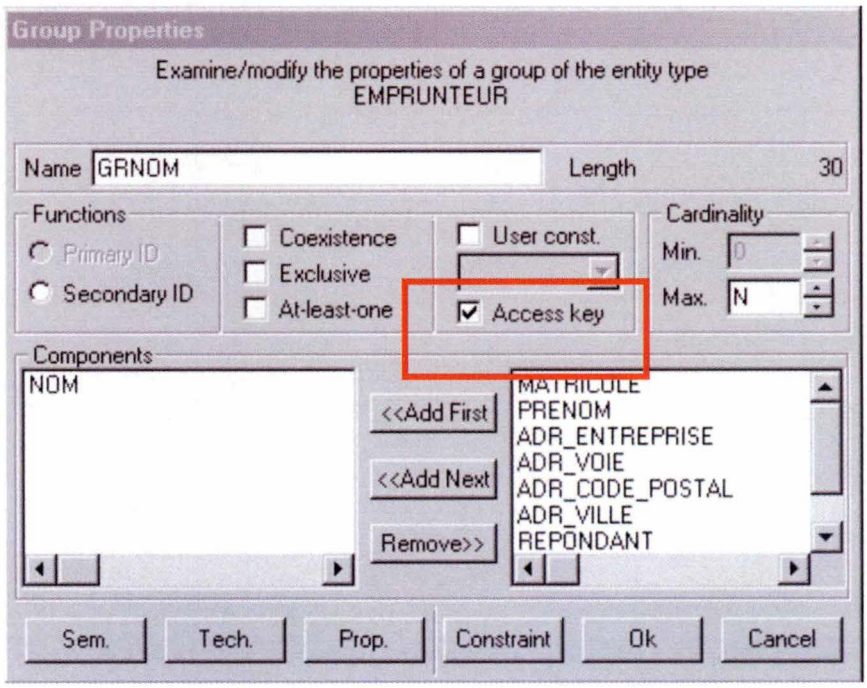


Figure 5.12. : Définition des clés d'accès

5.5.2. Paramétrisation du type de stockage des tables

Création de la propriété dynamique relative aux types de stockage des tables MySQL dans DB-MAIN

Pour implémenter la possibilité de définir un type de stockage particulier à MySQL dans DB-MAIN, le concepteur doit employer les méta-propriétés relatives aux types d'entités *(illustrées sur les figures 5.13 et 5.14)*. La première démarche à accomplir est de créer une nouvelle propriété dynamique ayant pour nom « type-table ». Cette nouvelle propriété doit être de type « modifiable et monovalué » et doit également contenir une liste de valeurs prédéfinies dans laquelle est mentionné l'ensemble des valeurs possibles des modes de stockage *(ISAM, MyISAM, MERGE, INNODB, ...) (voir figure 5.15)*. Cette liste peut être actualisée à tout moment en fonction de l'apparition ou de la disparition des types de stockage de MySQL.

Il est à noter qu'il est possible d'automatiser la création de la propriété dynamique relative aux types de stockage des tables à l'aide d'un script écrit avec le langage de programmation Voyager.

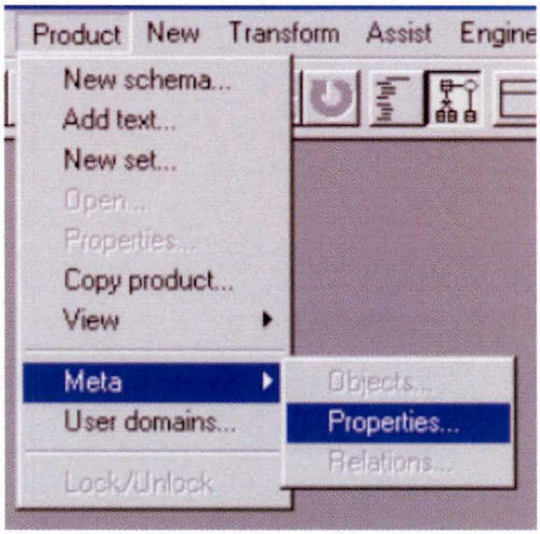


Figure 5.13. : Sélection de la fenêtre des Méta-propriétés

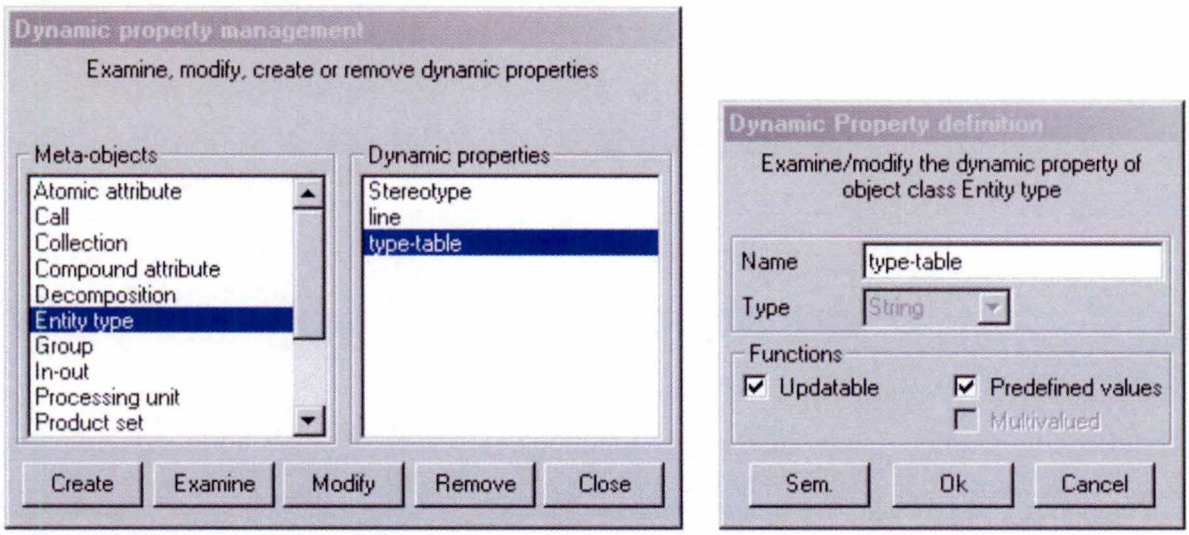


Figure 5.14. : Création de la propriété dynamique « type-table »

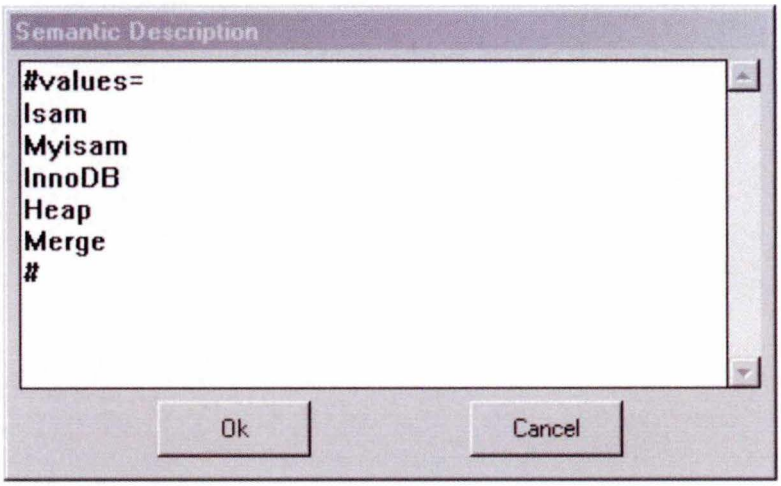


Figure 5.15. : Déclaration de la liste des valeurs prédéfinies reprenant les différents types de stockage des tables MySQL

Déclaration d'un type de stockage à une table MySQL définie dans le schéma physique avec DB-MAIN

Le type de stockage d'une table peut être invoqué en utilisant les propriétés dynamiques de cette table. Il suffit de sélectionner la propriété dynamique « type-table » et de choisir le type de stockage désiré en l'ajoutant dans la liste de valeur correspondante *(voir illustration sur la figure 5.16)*. Il n'est pas nécessaire de spécifier cette propriété pour chaque table du schéma physique. Le script de génération de code DDL n'en tiendra pas compte dans le cas où rien n'est renseigné et le moteur du SGBD MySQL spécifiera le type de stockage par défaut suivant la version utilisée.

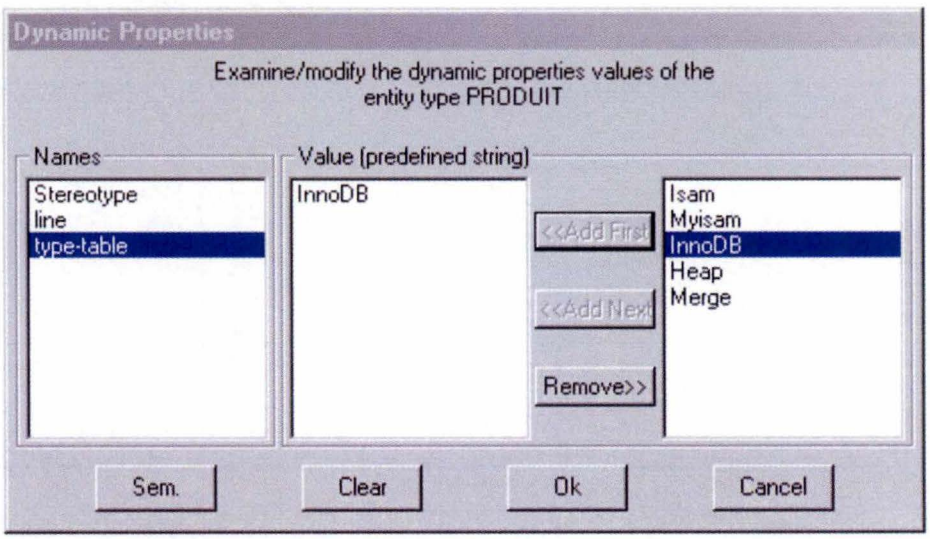


Figure 5.16. : Déclaration d'un type de stockage à une table du schéma physique

5.6. Codage

Cette phase consiste à produire du code exécutable qui va permettre de construire la base de données définie dans le schéma physique.

5.6.1. Génération du code DDL

La génération du code DDL représentatif du schéma physique défini dans DB-MAIN se fait à l'aide d'un programme développé en Voyager et spécialement conçu pour générer du code conforme à la syntaxe du SGBD MySQL. Pour exécuter cette manipulation, il suffit de lancer l'instruction « Execute Voyager » et de choisir le générateur adéquat qui se chargera d'élaborer un fichier de sortie dans lequel sera inséré le code DDL généré à partir du schéma physique. Le générateur de code DDL de MySQL est dénommé « mysql.oxo » et la description technique de ce module est reprise dans le chapitre 7. L'appel de ce générateur est illustré par la figure 5.17.

Le fichier obtenu en résultat et contenant les instructions de création de la base de données est utilisable directement dans l'environnement de MySQL.

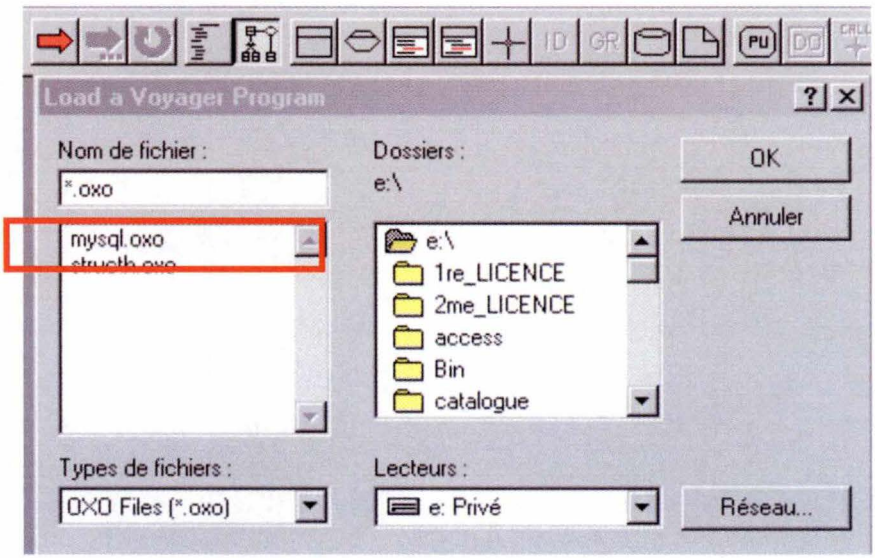


Figure 5.17. : Appel du générateur de code DDL de MySQL

5.6.2. Génération du fichier représentant les clés étrangères de la base de données

Pour le fonctionnement correct du module de gestion des clés étrangères, externe à DB-MAIN, il est nécessaire d'avoir une représentation des clés reliant les tables de la base de données. Cette représentation sera utile lors de la sollicitation des tables via les différentes interactions qu'il est possible d'effectuer *(insert, update et delete)*.
Le lien entre le schéma physique obtenu dans l'atelier logiciel DB-MAIN et ce module de gestion de clés étrangères se fait via l'intermédiaire d'un fichier dans lequel se retrouvent les informations concernant les clés reliant les tables.
Pour la génération de ce fichier, il suffit d'utiliser le module développé en Voyager et dénommé « structh.oxo ». Le résultat produit est un fichier ayant pour extension « structbd.h ». La description technique de ce module figure dans le chapitre 7. L'appel de ce générateur est illustré par la figure 5.18.

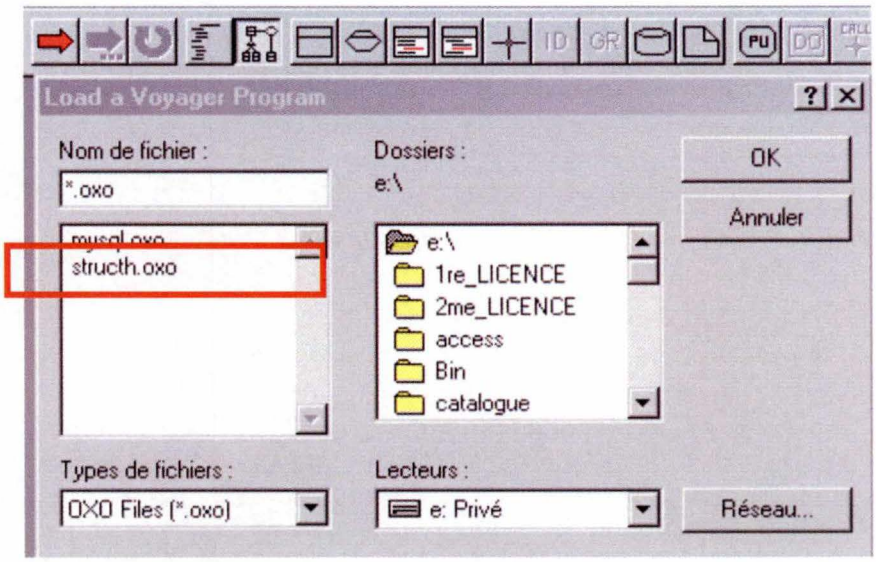


Figure 5.18. : Appel du générateur du fichier représentatif des clés étrangères de la BD

Chapitre 6 : Environnement de programmation MySQL

Ce chapitre est destiné à l'attention des programmeurs qui ont pour mission de mettre en place des applications dans lesquelles s'effectuent des interactions avec une base de données MySQL. Dans ce qui va suivre, une introduction est suivie d'une brève présentation des principaux paramètres à prendre en considération avant le développement. Les sections 6.1 et 6.2 sont extraites du chapitre 5 de l'ouvrage rédigé par P. Dubois sur MySQL [Dubois, 2000]. Par après, l'environnement dans lequel a été programmé le module de gestion des clés étrangères est décrit avec ses points forts. Pour terminer cette partie, la structure et le mode opératoire d'un programme utilisant le module de gestion des clés étrangères sont détaillés.

6.1. Introduction à la programmation avec MySQL

Bon nombre de programmes utilisent les services d'une base de données. Par exemple, les applications de gestion dans les entreprises ou encore les applications Web à partir desquelles il est possible de consulter et de gérer un ensemble de données.
Pour élaborer de telles applications, il est plus efficace de dialoguer directement avec le serveur de bases de données. Pour simplifier le développement de ces applications, une bibliothèque cliente est fournie par MySQL et permet l'accès aux bases de données depuis les programmes mis en place. La bibliothèque « client » implémente une interface de programmation d'applications *(API)* qui définit la façon dont les programmes clients établissent et gèrent les communications avec le serveur.
Chaque liaison de langage définit sa propre interface en spécifiant ses règles d'accès à MySQL.
Les trois interfaces de programmation d'applications *(API)* de MySQL les plus populaires sont :

- L'API « C » : interface de programmation principale avec MySQL.
- L'API « PHP » : PHP est un langage de script permettant d'intégrer des programmes dans les pages Web de façon particulièrement performante.
- L'API « DBI » pour le langage « Perl » : DBI est disponible sous la forme d'un module Perl qui joue le rôle d'interface avec d'autres modules au niveau du pilote de la base de données.

Il est à signaler qu'il existe également d'autres méthodes plus générales d'interactions entre un programme et une base de données. Une d'entre elles est d'ailleurs utilisée par le module de gestion des clés étrangères et est présentée dans ce qui va suivre.

6.2. Paramètres de développement à prendre en considération

Avant de commencer le développement d'une application, il est important de tenir compte des paramètres suivants :

- L'environnement d'exécution cible, c'est-à-dire le contexte dans lequel l'application sera utilisée.
- La portabilité : critère permettant de faciliter le processus de changement de SGDB interagissant avec l'application développée.
- Les performances, en tenant compte du nombre d'utilisateurs, de la fréquence d'utilisation, de la charge du réseau, ...
- Les facilités de développement.

6.3. Composants de l'environnement du module de gestion des clés étrangères

Le module de gestion des clés étrangères fait appel à des composants et à des techniques de programmation particulières. Les paramètres pris en considération pour le développement de ce module ont été principalement d'obtenir un outil dont la portabilité au niveau du SGBD soit la plus générale et dont l'environnement d'exécution soit le plus large possible. Cela explique la raison pour laquelle le module de gestion des clés étrangères fait appel aux techniques suivantes :

- Module généré sous la forme d'une librairie de fonctions *(DLL)*
- Interaction avec la base de données MySQL via la technologie « ODBC »
- Utilisation d'un fichier de type « header file »
- Développement - langage de programmation C/C++

6.3.1. Librairie de fonctions (DLL)

Une librairie de fonctions *(DLL)* est une librairie compilée, c'est-à-dire un fichier contenant plusieurs fonctions d'un programme. Une librairie de fonctions est prête à être utilisée et est chargée dynamiquement en mémoire lors de l'appel d'un programme. Une DLL n'est pas compilée en même temps que le programme qui l'utilise et est indépendante de ces derniers. De plus, une librairie de fonctions peut être utilisée par plusieurs applications simultanément. Cela a pour avantages de ne plus charger inutilement la mémoire et de permettre à chaque application de ne contenir que la référence de la fonction appelée. L'intérêt des librairies de fonctions réside également, par exemple, dans le cas d'une mise à jour. En effet, lors d'une amélioration ou d'une correction dans une fonction, il n'est pas nécessaire de modifier et de recompiler le ou les programmes « clients » entièrement. Il suffit simplement de recompiler la DLL modifiée. Un dernier avantage réside également dans le partage des fonctions avec d'autres programmeurs tout en gardant le code source confidentiel. En effet, seuls le nom et l'ensemble des paramètres des fonctions sont disponibles. Des informations supplémentaires au sujet des librairies de fonctions peuvent être obtenues en utilisant la référence suivante : [Laplanche 2002].

6.3.2. Technologie ODBC

ODBC signifie « Open DataBase Connectivity ». Il s'agit d'un format défini par Microsoft permettant la communication entre les SGBD du marché et des clients « bases de données ». Le gestionnaire ODBC est présent sur les systèmes d'exploitation Windows mais il existe aussi des implémentations sous d'autres plates-formes telles que Unix et Linux.
La technologie ODBC a pour avantage de permettre d'interfacer de façon standard une application à n'importe quel serveur de bases de données, pour peu que celui-ci possède un « pilote ODBC ». Il est à noter que la quasi totalité des SGBD, dont les principaux SGBD du marché, possèdent un tel pilote. De plus amples informations concernant la technologie ODBC peuvent être obtenues en utilisant les références suivantes : [Barbier 2002], [ENST 2000] et [Pillou 2003].

6.3.3. Utilisation d'un fichier de type « header file »

Le fichier « header file », dont l'extension est « .h », joue le rôle d'un composant « dynamique » dans lequel se trouvent les informations relatives à la structure des clés reliant les tables contenues dans le schéma physique. Ce fichier fait office de lien entre

le schéma physique de la base de données, le module de gestion des clés étrangères et la base de données MySQL existante. Les informations, contenues de manière structurée dans ce fichier dynamique, sont fournies à partir du schéma physique mis en place par le concepteur de la base de données. Ensuite, ces données sont consultées par le module de gestion des clés étrangères qui coopère avec la base de données MySQL. La génération de ce fichier « header file » se fait à l'aide d'un module présenté et décrit dans les chapitres 5 et 7.

6.3.4. Développement - langage de programmation C/C++

Le langage de programmation C/C++ a été utilisé pour la mise en œuvre de la librairie de fonctions permettant la gestion des clés étrangères. Les principales raisons qui ont motivé ce choix sont les suivantes :

- Langage de programmation largement répandu dans la communauté des programmeurs.
- Le développement d'applications en C/C++ peut être réalisé à partir de nombreux systèmes d'exploitation.
- Langage permettant d'inclure facilement un fichier externe et dynamique de type « header file ».
- Langage intégrant facilement la technologie ODBC permettant l'interaction avec les bases de données via l'utilisation des pointeurs.

La figure 6.1 illustre les concepts de l'environnement de programmation MySQL présentés dans la section 6.3. Les flèches signifient que l'élément de départ interagit avec l'élément qui lui est relié.

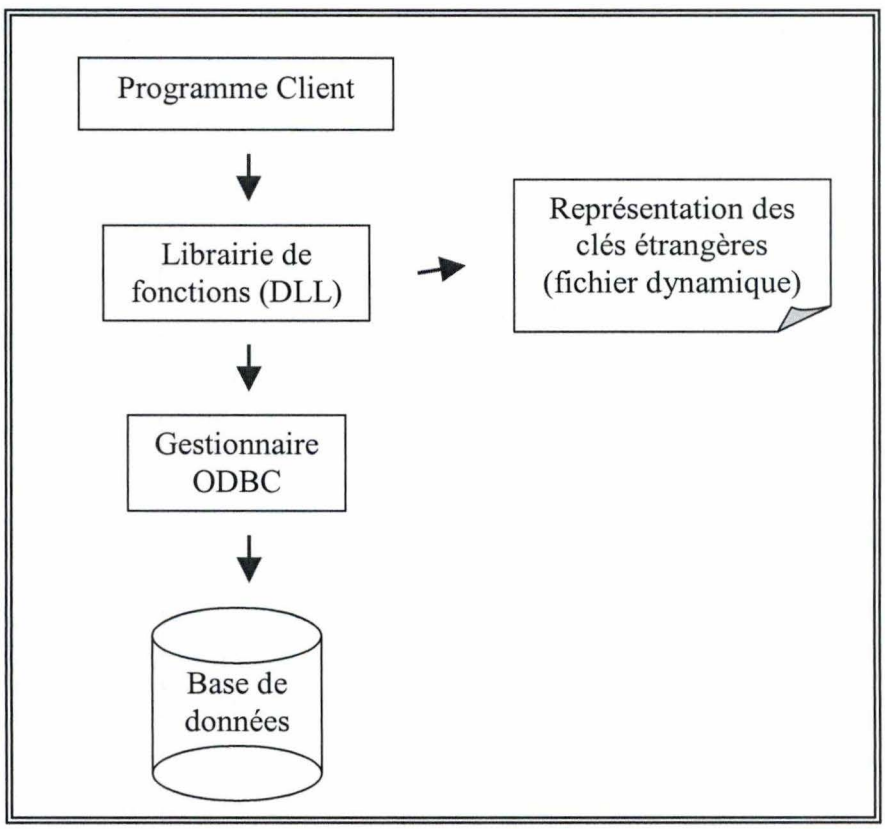


Figure 6.1. : L'environnement de programmation MySQL

6.4. Structure d'un programme utilisant le module de gestion des clés étrangères

Cette section va illustrer la structure d'un programme faisant appel au module de gestion des clés étrangères. Le mode opératoire reprend les actions suivantes :

- Définir un lien ODBC vers la base de données MySQL
- Compiler la librairie de fonctions et l'ajouter dans le programme
- Déclarer les fonctions et les procédures importées de la librairie de fonctions
- Appeler les fonctions et procédures de la librairie de fonctions

6.4.1. Définition du lien ODBC

Pour pouvoir mettre en place un lien ODBC vers une base de données MySQL, il faut avoir installé au préalable l'utilitaire dénommé « MyODBC » qui déploie sur le système d'exploitation le pilote ODBC pour MySQL. La figure 6.2 montre le pilote ODBC de MySQL installé sur le système d'exploitation. Cette procédure n'est à réaliser qu'une seule fois.

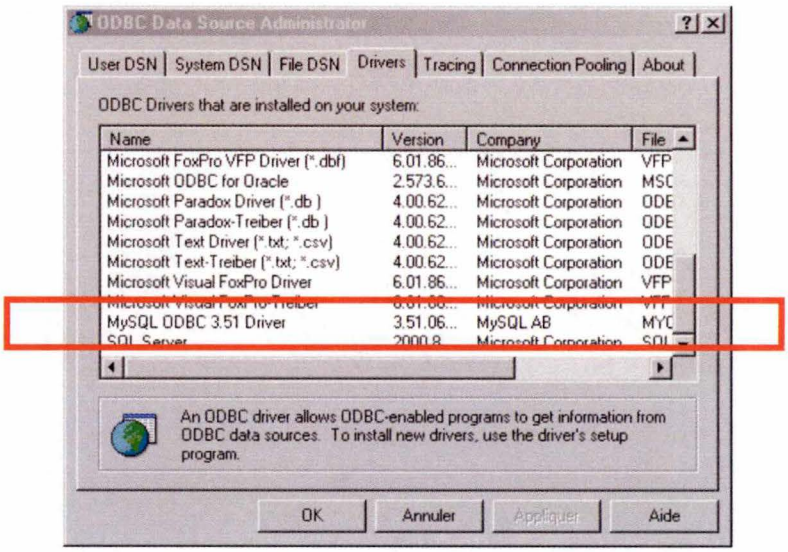


Figure 6.2. : Pilote ODBC pour MySQL installé sur le système d'exploitation

Par après, il suffit de créer un nouveau lien « utilisateur » dans lequel sont renseignés le nom du lien, l'adresse du serveur et le nom de la base de données MySQL que l'utilisateur souhaite gérer. D'autres informations, comme par exemple, l'identifiant de l'utilisateur et son mot de passe, peuvent également être renseignées dans cette partie. La figure 6.3 montre la fenêtre de création d'un lien ODBC vers une base de données MySQL.

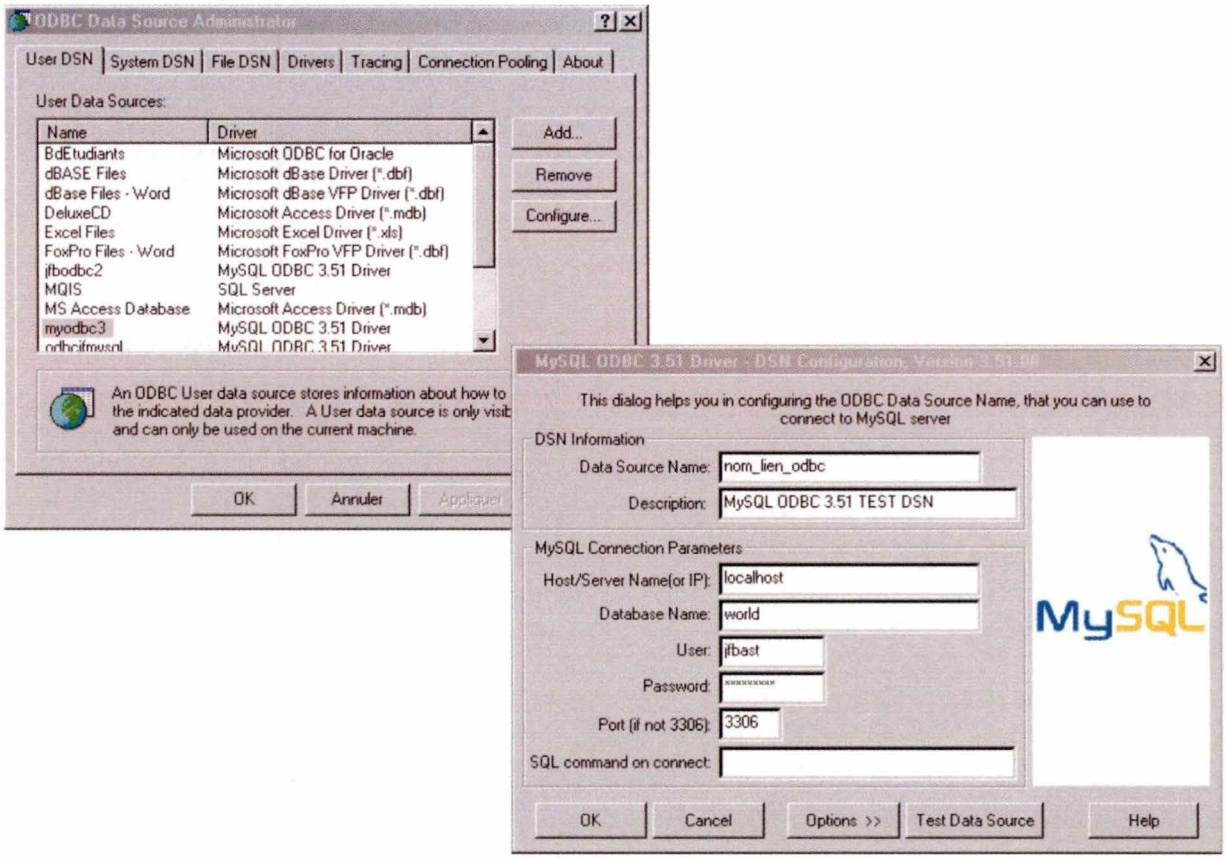


Figure 6.3. : Création d'un lien ODBC vers une base de données MySQL

6.4.2. Compilation de la librairie de fonctions et ajout dans le programme

Avant de faire appel à la librairie de fonctions, il est nécessaire d'y intégrer le fichier dynamique reprenant la structure des clés étrangères *(le fichier « header file » généré à partir du schéma physique dans DB-MAIN)*. Pour se faire, il suffit d'abord de déposer le fichier « .h » à l'emplacement requis et ensuite, de compiler la librairie de fonctions. La compilation aboutit à la création de deux fichiers dont l'extension est « .dll » et « .lib ». Le fichier « .dll » est le fichier exécutable et le fichier « .lib » fait office d'interface avec la librairie de fonctions car c'est dans ce fichier que sont reprises les informations relatives aux fonctions et aux procédures contenues dans la librairie de fonctions. Pour terminer, afin d'intégrer la librairie de fonctions, il suffit d'ajouter le fichier « .lib » dans l'environnement du programme client.

6.4.3. Déclaration des fonctions et des procédures importées

Lors de l'élaboration du programme, il est nécessaire de déclarer les fonctions et les procédures de la librairie de fonctions de gestion des clés étrangères qu'il est possible d'importer et d'utiliser. Cette phase de déclaration se fait en ajoutant les instructions suivantes dans le corps du programme :

```
Extern « C » __declspec (dllimport) void definir_nom_lien_odbc (char* nom_lien) ;
Extern « C » __declspec (dllimport) int validation (char* ligne_commande) ;
```

Concernant la librairie de gestion des clés étrangères, les deux importations permettent les actions suivantes :

- « definir_nom_lien_odbc » : il s'agit d'une procédure qui enregistre le nom du lien ODBC qui est utilisé. Cela a pour conséquence de spécifier en même temps la base de données MySQL qui sera utilisée.
- « validation » : il s'agit d'une fonction validant l'instruction reprise en paramètre d'entrée. La validation se fait suivant les règles de gestion des clés étrangères décrites dans le chapitre 7. Lorsque l'instruction est refusée, il n'y a pas d'interaction avec la base de données. La fonction renvoie un entier permettant de vérifier le comportement de la fonction. Les valeurs renvoyées et leur signification sont :

  1 : L'instruction s'est bien effectuée.
  2 : Le module a accepté l'instruction mais le SGBD MySQL la refuse. Cette situation peut être la conséquence de la duplication d'une clé primaire, d'une mauvaise valeur associée à une colonne, ...
  3 : La fonction « validation » refuse l'instruction car celle-ci altère l'intégrité des données.

6.4.4. Appel des fonctions et des procédures importées

Pour appeler une fonction ou une procédure importée et donc, extérieure au programme, il suffit simplement d'ajouter dans le code du programme le nom correspondant et d'y indiquer le nombre exact de paramètres avec le type correspondant. Par exemple, pour appeler la fonction de validation d'une ligne de commande, il suffit de déclarer un champ de type « char » et un autre de type « int », de remplir le champ de type « char » avec la syntaxe de l'instruction et de le passer en paramètre lors de l'appel. La syntaxe sera la suivante :

```
char commande[2000] ;
int ctrl ;
...
gets (commande) ;
...
ctrl=validation(commande) ;
```

La technique est semblable pour l'appel de la procédure de définition d'un lien ODBC vers une base de données MySQL. Toutefois, lors de l'appel d'une procédure, il n'y a pas de paramètre en retour.

Chapitre 7 : Extension de l'atelier DB-MAIN

Ce chapitre va analyser le comportement et le code des modules suivants :

- Les générateurs ajoutés dans l'atelier logiciel DB-MAIN permettant la création de la base de données MySQL et la gestion des clés étrangères définies entre les tables de cette base.

- La librairie de fonctions *(DLL)* relatives à la gestion des clés étrangères vérifiant et validant les instructions d'interaction avec la base de données.

Cette analyse permettra de comprendre les principes des traitements effectués et de les modifier selon les besoins.

7.1. Le générateur de code DDL de MySQL

7.1.1. Module de départ

Le module à partir duquel le générateur de code DDL de MySQL a été développé est le module permettant la génération de code DDL de type SQL Standard. Ce module est disponible dans l'atelier logiciel DB-MAIN.
Pour modifier du code source écrit en Voyager, il faut d'abord éditer le fichier ayant l'extension de type « .v2 ». Ensuite, il faut compiler le nouveau code pour obtenir un exécutable pour le logiciel DB-MAIN.
La compilation du fichier « .v2 » s'effectue en lançant cette instruction :

```
comp_V2 fichier.V2
```

Par après, un module ayant une extension de type « .oxo » est utilisable dans DB-MAIN.

7.1.2. Objectifs

Le générateur de code DDL de MySQL est employé pour générer, à partir du schéma physique défini dans l'atelier logiciel DB-MAIN, un fichier dans lequel se retrouvent toutes les instructions nécessaires à la création de la base de données. Le fichier produit en résultat est directement exploitable dans l'environnement de MySQL. En effet, dans l'environnement de MySQL, il suffit de lancer le fichier généré pour obtenir la création de la base de données correspondante. Le code complet de ce générateur se trouve dans la partie « Annexes ».

7.1.3. Architecture du programme

La structure principale

La structure principale englobe les parties suivantes :

- Création et ouverture d'un fichier pour y générer le résultat exploitable directement à partir de l'interface de commandes de MySQL
  [effectué via la fonction « OuvreFichier »]

- Sélection du schéma physique courant comme base de travail
  [effectué via un ensemble d'instructions]

- Génération du script DDL de MySQL correspondant à la base de travail dans le fichier créé auparavant
  [effectué via la procédure « GenererMySql »]

- Fermeture du fichier [effectué via une instruction]

La procédure « GenererMySql »

Cette procédure ne nécessite pas de paramètres en entrée. Le corps de cette procédure effectue :

- La création de la base de données
  [effectué via un ensemble d'instructions]

- Pour chaque table du schéma considéré :

  - La création de la structure de la table
    [effectué via la procédure « GenererTable »]
  - La création des contraintes de la table
    [effectué via la procédure « GenererContrainte »]
  - La création des index de la table
    [effectué via la procédure « GenererIndex »]

La procédure « GenererTable »

Cette procédure a besoin d'une table comme paramètre d'entrée. Les traitements effectués sont :

- La création de la table
  [effectué via un ensemble d'instructions]

- Et pour chacun des attributs de la table :

  - La conversion du domaine de valeurs
    [effectué via la fonction « FindType »]
  - La génération de ses caractéristiques
    [effectué via un ensemble d'instructions]

- La détermination du type de la table
  [effectué via un ensemble d'instructions]

La procédure « GenererContrainte »

Cette procédure a besoin d'une table comme paramètre d'entrée. Les traitements effectués sont :

- La déclaration des clés étrangères
  [effectué via la fonction « GenererForeignKey »]

- La déclaration des contraintes de type « Check »
  [effectué via la fonction « GenererCheck »]

La procédure « GenererForeignKey »

Cette procédure nécessite deux paramètres en entrée : la table considérée et le groupe d'attributs de cette table dans lequel est définie la notion de clé étrangère.
Dans un premier temps, la recherche de la table et de son groupe d'attributs reliés aux deux paramètres d'entrée s'effectue. Ensuite, l'ensemble d'instructions de cette procédure a pour objectif de déclarer des instructions de modification de la table et de déclarer la syntaxe de la clé étrangère correspondante. Pour cela, la procédure fait appel à deux autres fonctions [« NameOf » et « GetListOfComponents »] et à une autre procédure [« PrintComponents »] pour la recherche et la déclaration des noms des éléments repris dans le groupe.

La procédure « GenererCheck »

Cette procédure a besoin d'un paramètre en entrée : le groupe dans lequel est définie la notion de contrainte de coexistence, d'exclusivité ou de type « at-least-one ».
Cette procédure a pour mission de déclarer ces contraintes de type « check » via une instruction de modification de la table, les traitements sont effectués en fonction du type de contrainte à générer.
Cette procédure fait aussi appel aux fonctions [« NameOf »] et [« GetListOfComponents »] pour la recherche des composants du groupe passé en paramètre.

La procédure « GenererIndex »

Cette procédure reçoit une table comme paramètre d'entrée. Le traitement effectué est :

- La création des index définis dans le schéma pour la table considérée. Cette procédure fait appel à la procédure [« PrintComponents »] pour la déclaration des éléments constituant l'index.

La fonction « OuvreFichier »

Cette fonction n'a pas besoin d'argument en entrée. Elle renvoie comme résultat un entier permettant de contrôler le bon déroulement de l'opération. Les instructions reprises dans cette fonction permettent de définir un nom de fichier dans lequel est généré le code DDL de MySQL et qui sera utilisable dans l'environnement MySQL. Pour terminer, la fonction ouvre le fichier en écriture.

La fonction « FindType »

Cette fonction nécessite comme argument une occurrence d'un attribut d'un type de tables. Le résultat de cette fonction est une chaîne de caractères contenant le type de l'attribut passé en entrée.
Cette fonction a donc pour mission de convertir le type défini dans DB-MAIN en un type conforme à MySQL.

La fonction « NameOf »

Cette fonction a comme argument d'entrée un composant d'un groupe et retourne une chaîne de caractères contenant le nom de ce composant.

La fonction « GetListOfComponents »

Cette fonction donne la liste des composants d'un groupe passé en argument d'entrée de cette fonction.

La procédure « PrintComponents »

Cette procédure nécessite comme paramètre d'entrée une occurrence d'un groupe. Le traitement de cette procédure génère la liste des composants de ce groupe.

====

7.1.4. Gestion des types « SET » et « ENUM »

Dans le générateur de code DDL de MySQL, la gestion de ces deux types particuliers se fait dans la fonction « FindType » permettant d'obtenir le domaine d'un attribut. Si ce dernier a un domaine de valeurs de type « User-defined », le programme va déterminer s'il s'agit d'un attribut de type « SET » ou « ENUM » et va consulter la propriété « value constraint » pour obtenir la liste des valeurs définies.
Le retour du type se fait donc en deux étapes :

- La première étape consiste à déterminer le type exact « SET » ou « ENUM »

- La seconde étape consiste à lire et formater, tant qu'il en existe, les valeurs associées à ce type d'attribut. Le formatage consiste à emballer les valeurs entre des apostrophes et de les séparer par une virgule.

7.1.5. Gestion du type de stockage d'une table

Dans la procédure de création d'une table [« GenererTable »], à la fin de la déclaration de la structure, le type de stockage de la table est ajouté si une information a été spécifiée pour la propriété dynamique « type-table ».

7.1.6. Gestion des clés étrangères et des contraintes de type « check »

Par souci de portabilité et d'anticipation pour les versions futures, le script reprend la déclaration des clés étrangères et la déclaration des contraintes de types « check » [à l'exception des contraintes de type « check (exist ...) »]
MySQL acceptera leur déclaration mais n'en tiendra pas compte dans les versions actuelles. Un message signalera d'ailleurs cette information à l'utilisateur via la console permettant de contrôler le bon déroulement de la génération de code.

7.2. Le générateur du fichier contenant la représentation des clés étrangères

7.2.1. Objectif

Le générateur a pour mission de produire un fichier, à partir de l'atelier logiciel DB-MAIN, dans lequel sont reprises les informations relatives aux clés étrangères d'un schéma physique d'une base de données. Ce fichier est utilisé pour la déclaration dynamique des clés étrangères et est exploité par le module permettant la gestion de ces dernières. Il y a une distinction entre les deux types de clés contenues dans ce fichier : Les clés partant d'une table vers une table de référence, appelées clés de type « TO » et les clés d'une table vers une autre table lui faisant référence, appelées clés de type « FROM ». Le code complet de ce générateur se trouve dans la partie « Annexes ».

7.2.2. Architecture du programme

La structure principale

La structure principale englobe les parties suivantes :

- Création et ouverture d'un fichier pour y générer le résultat exploitable par le module externe à MySQL gérant les clés étrangères. Le fichier généré est un fichier ayant un nom constant « structbd.h ». L'extension de type « header file » est utilisable dans l'environnement de développement d'applications en C/C++
  [effectué via la fonction « OuvreFichier »]

- Sélection du schéma courant comme base de travail
  [effectué via un ensemble d'instructions]

- Génération de l'entête dans le fichier
  [effectué via la procédure « GenererEntete »]

- Génération des clés étrangères de type « TO »
  [effectué via la procédure « GenererForeignKeyTO »]

- Génération des clés étrangères de type « FROM »
  [effectué via la procédure « GenererForeignKeyFROM »]

- Fermeture du fichier
  [effectué via une instruction]

La procédure « GenererEntete »

Cette procédure ne nécessite pas de paramètres en entrée. Le traitement effectué est :

- La déclaration des structures dans lesquelles sont stockées les informations concernant les clés étrangères. Il y a deux types de structures. La première contient les clés étrangères de type « TO », c'est-à-dire, la représentation des clés partant d'une table origine vers

une table cible. La seconde structure, quant à elle, contient les clés étrangères de type « FROM », c'est-à-dire, la représentation des clés d'une table cible vers une table origine.

La procédure « GenererForeignKeyTO »

Cette procédure n'a pas besoin de paramètres en entrée. Le corps de cette procédure effectue :

- Pour chaque table contenue dans le schéma courant :
  - La génération de l'ensemble de ses clés étrangères de type « TO ». [effectué via la procédure « GenererForeignKeyTO_Table »]

La procédure « GenererForeignKeyFROM »

Cette procédure n'a pas besoin de paramètres en entrée. Le corps de cette procédure effectue :

- Pour chaque table contenue dans le schéma courant :
  - La génération de l'ensemble de ses clés étrangères de type « FROM ». [effectué via la procédure « GenererForeignKeyFROM_Table »]

La procédure « GenererForeignKeyTO_Table »

Cette procédure reçoit comme argument la table pour laquelle va être déclaré l'ensemble des clés étrangères de type « TO » qu'elle contient. C'est-à-dire, les clés étrangères dont la table en argument est la table d'origine. Le traitement s'effectue comme suit :

- Pour chaque groupe d'attributs représentant une clé étrangère dans la table :
  - Génération de la clé étrangère de type « TO » de ce groupe d'attributs.

  [effectué via la procédure « GenererForeignKeyTO_Groupe_Table »]

La procédure « GenererForeignKeyFROM_Table »

Cette procédure reçoit comme argument la table pour laquelle va être déclaré l'ensemble des clés étrangères de type « FROM » qu'elle contient. C'est-à-dire, les clés étrangères dont la table en argument est la table cible. Le traitement s'effectue comme suit :

- Pour chaque groupe d'attributs représentant une clé étrangère dans la table :

- Génération de la clé étrangère de type « FROM » de ce groupe d'attributs.

[effectué via la procédure «GenererForeignKeyFROM_Groupe_Table»]

La procédure « GenererForeignKeyTO_Groupe_Table »

Cette procédure nécessite deux paramètres en entrée : la table considérée et le groupe d'attributs de cette table dans lequel est définie la notion de clé étrangère de type « TO ».
Dans un premier temps, la recherche de la table et de son groupe d'attributs reliés aux deux paramètres d'entrée s'effectue. Ensuite, l'ensemble d'instructions de cette procédure a pour objectif de générer de façon structurée dans le fichier de sortie la clé étrangère que le groupe passé en argument représente. Pour cela, cette procédure fait appel à la procédure [« PrintComponentsofKeyTO »].

La procédure « GenererForeignKeyFROM_Groupe_Table »

Cette procédure nécessite deux paramètres en entrée : la table considérée et le groupe d'attributs de cette table dans lequel est définie la notion de clé étrangère de type « FROM ».
Dans un premier temps, la recherche de la table et de son groupe d'attributs reliés aux deux paramètres d'entrée s'effectue. Ensuite, l'ensemble d'instructions de cette procédure a pour objectif de générer de façon structurée dans le fichier de sortie la clé étrangère que le groupe passé en argument représente. Pour cela, cette procédure fait appel à la procédure [« PrintComponentsofKeyFROM »].

La procédure « PrintComponentsofKeyTO »

Cette procédure a besoin de quatre paramètres : les deux tables qui sont reliées entre elles via la clé étrangère et leur groupe d'attributs correspondants. La procédure génère dans le fichier de sortie les informations structurées relatives à cette clé. La structure est la suivante :

nom table origine @ nom colonne table d'origine # nom table cible @ nom colonne table cible

Toutes les clés sont séparées par un point virgule.

Cette représentation de l'information est utilisée par le module de gestion des clés étrangères pour les opérations de recherches. La procédure fait également appel à deux autres fonctions [« NameOf » et « GetListOfComponents »] pour la recherche des éléments repris dans les groupes passés en paramètres. Pour la génération des noms de colonnes de la table origine et de la table cible, la procédure fait appel à la procédure [« PrintComponents »].

La procédure « PrintComponentsofKeyFROM »

Cette procédure a besoin de quatre paramètres : les deux tables qui sont reliées entre elles via la clé étrangère et leur groupe d'attributs correspondants. La procédure génère dans le fichier de sortie les informations structurées relatives à cette clé. La structure est la suivante :

nom table cible @ nom colonne table cible # nom table origine @ nom colonne table d'origine

Toutes les clés sont séparées par un point virgule.

Cette représentation de l'information est utilisée par le module de gestion des clés étrangères pour les opérations de recherches. La procédure fait également appel à deux autres fonctions [« NameOf » et « GetListOfComponents »] pour la recherche des éléments repris dans les groupes passés en paramètres. Pour la génération des noms de colonnes de la table origine et de la table cible, la procédure fait appel à la procédure [« PrintComponents »].

La fonction « OuvreFichier »

Cette fonction n'a pas besoin d'argument en entrée. Elle renvoie comme résultat un entier permettant de contrôler le bon déroulement de l'opération. Les instructions reprises dans cette fonction définissent le nom « structbd.h » au fichier. Pour terminer, la fonction ouvre le fichier en écriture.

La fonction « GetListOfComponents »

Cette fonction donne la liste des composants d'un groupe passé en argument d'entrée de cette fonction.

La procédure « PrintComponents »

Cette procédure nécessite comme paramètre d'entrée une occurrence d'un groupe. Le traitement de cette procédure génère la liste des composants de ce groupe.

====

7.2.3. Exemples de formats de clés générés

Les différentes sortes de clés étrangères sont représentées ci-dessous avec leur format correspondant dans le fichier généré :

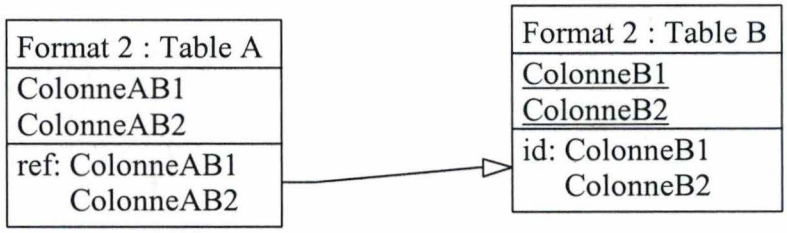
Format 1

Clé de type « TO » : TableA@ColonneAB#TableB@ColonneB;
Clé de type « FROM » : TableB@ColonneB#TableA@ColonneAB;



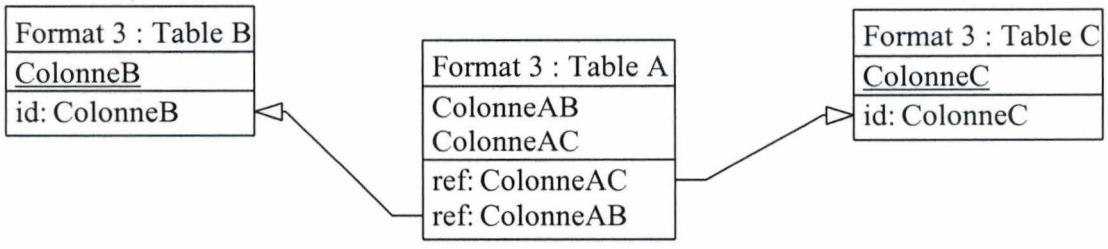
Format 2

Clé de type « TO » :
TableA@ColonneAB1,ColonneAB2#TableB@ColonneB1,ColoneB2;
Clé de type « FROM » :
TableB@ColonneB1,ColonneB2#TableA@ColonneAB1,ColonneAB2;



Format 3

Clé de type « TO » :
TableA@ColonneAB#TableB@ColonneB;
TableA@ColonneAC#TableC@ColonneC;
Clé de type « FROM » :
TableB@ColonneB#TableA@ColonneAB;
TableC@ColonneC#TableA@ColonneAC;



7.3. La librairie de fonctions relatives à la gestion des clés étrangères

7.3.1. Objectifs

Le module de gestion des clés étrangères consiste en une librairie de fonctions pouvant être utilisées via des applications clientes. La déclaration et l'appel des fonctionnalités du module sont présentés dans les sections 6.3 et 6.4.
Dans cette librairie, l'objectif principal est la vérification de la cohérence et la validation d'une instruction avant l'enregistrement effectif dans une table d'une base de données MySQL. Pour se faire, un contrôle s'effectue en fonction du type de l'instruction reçue en argument d'entrée. Les trois types possibles sont l'insertion, la suppression et la modification d'enregistrements. Les diagrammes d'actions des contrôles effectués pour ces actions sont détaillés par après.
La deuxième fonctionnalité de la librairie permet de paramétrer le nom du lien ODBC dans lequel sont renseignées les informations relatives à la base de données désignée.
Le code complet de cette librairie de fonctions se trouve dans la partie « Annexes ».

7.3.2. Traitement « Insertion »

Lors de l'insertion d'une ligne dans une table, si cette dernière est reliée à une ou plusieurs autres tables et si, dans la ligne insérée, une ou plusieurs colonnes font référence à des enregistrements contenus dans d'autres tables, un contrôle vérifie l'intégrité des données avant la réalisation effective de l'insertion.
Le module agit en fonction des paramètres représentés dans le diagramme d'actions suivant :

Le module recherche toutes les clés reliant la table dans laquelle il va y avoir une insertion avec les tables de référence.

S'il existe des clés de ce type :

Pour chacune des clés, le module génère une instruction de sélection afin de vérifier que la valeur de la ou des colonnes est bien reprise dans la table de référence.

Si le résultat est positif, l'information insérée est cohérente et l'insertion est validée.

Sinon, le résultat est égal à 0, cela signifie qu'il n'existe pas de données de référence et l'insertion est refusée.

Sinon, il n'existe pas de clés et l'insertion est acceptée sans contrôle d'intégrité des données.

7.3.3. Traitement « Suppression »

Dans le module de gestion des clés étrangères, l'application de la suppression s'effectue en utilisant la stratégie « No Action ». Cela signifie que, lors de la validation de l'intégrité des données, s'il existe des lignes référençant la ou les lignes à supprimer, la suppression sera refusée. Autrement dit, la stratégie « No Action » interdit de supprimer des lignes lorsque celles-ci sont référencées.

Lors de la suppression d'une ligne contenue dans une table qui est référencée par une ou plusieurs autres, un contrôle se réalise avant la validation.

Le contrôle de cohérence des données effectue les actions représentées dans le diagramme d'actions suivant :

Le module recherche toutes les clés reliant la table dans laquelle une ligne va être supprimée avec la ou les tables qui lui font référence.

S'il existe des clés de ce type,

Le module génère, pour chacune des clés, une requête de sélection de type « jointure » afin de vérifier si la ligne à supprimer est référencée par d'autres tables.

Si le résultat des sélections est égal à 0, alors la ligne n'est pas référencée et la suppression est validée.

Sinon, le résultat est strictement supérieur à 0, cela signifie qu'il existe des enregistrements qui référencent la ligne. La suppression est alors refusée.

Sinon, il n'existe pas de clés et la suppression est acceptée sans contrôle d'intégrité des données.

7.3.4. Traitement « Modification – Mise à jour »

Lors d'une mise à jour d'une ligne dans une table, deux types de vérifications vont être effectués.
Premièrement, si la table est reliée à une ou plusieurs autres tables via les clés étrangères, et si, dans cette ligne modifiée, une ou plusieurs colonnes font référence à des enregistrements contenus dans d'autres tables, alors, un contrôle vérifie l'intégrité des données avant la réalisation effective de la modification.
Deuxièmement, lors de la modification d'une ligne référencée par une ou plusieurs autres tables, un contrôle se réalise avant la validation. L'application de la modification s'effectue en utilisant également la stratégie « No Action ». Cela signifie que lors de la validation de l'intégrité des données, s'il existe une ligne référençant la ou les lignes à modifier, l'opération de mise à jour sera refusée.

Le module agit en fonction des paramètres représentés dans le diagramme d'actions suivant :

Premier contrôle :
Le module recherche toutes les clés reliant la table dans laquelle il va y avoir une mise à jour avec les tables de référence.

S'il existe des clés de ce type :

Pour chacune des clés, le module génère une instruction de sélection afin de vérifier que la valeur de la ou des colonnes est bien reprise dans la table de référence.

Si le résultat est positif, l'information modifiée est cohérente et la mise à jour est acceptée.

Sinon, le résultat est égal à 0, cela signifie qu'il n'existe pas de données de référence et la mise à jour est refusée.

Sinon, il n'existe pas de clés et la mise à jour est acceptée sans contrôle d'intégrité des données.

Deuxième contrôle :
Le module recherche toutes les clés reliant la table dans laquelle une ligne va être modifiée avec la ou les tables qui lui font référence.

S'il existe des clés de ce type,

Le module génère, pour chacune des clés, une requête de sélection de type « jointure » afin de vérifier si la ligne à modifier est référencée par d'autres tables.

Si le résultat de la sélection est égal à 0, alors la ligne n'est pas référencée et la mise à jour est acceptée.

Sinon, le résultat est strictement supérieur à 0, cela signifie qu'il existe des enregistrements qui référencent la ligne. La mise à jour est alors refusée.

Sinon, il n'existe pas de clés et la mise à jour est acceptée sans contrôle d'intégrité des données.

Si la mise à jour a été **acceptée dans tous** les contrôles, l'instruction est validée et la mise à jour est effective dans la base de données.

Si la mise à jour a été **refusée dans un seul contrôle**, l'instruction n'est pas validée est la mise à jour ne sera pas exécutée.

7.3.5. Architecture

La structure principale de la fonction de gestion des clés étrangères

La fonction, dénommée « validation » dans le script de la librairie de fonctions, reçoit comme argument d'entrée une chaîne de caractères correspondant à une ligne de commande et renvoie en retour un entier permettant de déterminer le comportement de la fonction. Le traitement rassemble les parties suivantes :

- Réception du paramètre provenant de l'application cliente. Le paramètre reçu représente une ligne de commande d'interaction avec une base de données MySQL.
  [effectué via un ensemble d'instructions]

- Analyse du type de traitement à effectuer. Les trois traitements acceptés sont les insertions *(INSERT)*, les suppressions *(DELETE)* et les mises à jour *(UPDATE)*.
  [effectué via un ensemble d'instructions]

- Réalisation des procédures de vérification et de validation en fonction de la ligne de commande reçue en paramètre.

  - Si la ligne de commande est relative à une insertion :

    Recherche du nom de la table affectée
    [effectué via un ensemble d'instructions]
    Formatage des colonnes et des valeurs à insérer
    [via la procédure « formatage_colonnes_insert »]
    Recherche des clés de type « TO »
    [via la fonction « rechercher_cle_TO »]
    Préparation et vérification de l'intégrité des données
    [via la procédure « verification_integrite »]
    Validation dans la base de données
    [via la procédure « traitement_BD »]

  - Si la ligne de commande est relative à une suppression :

    Recherche du nom de la table affectée
    [effectué via un ensemble d'instructions]
    Formatage des colonnes et leur valeur reprises dans la condition
    [via la procédure « formatage_condition_delete_update »]
    Recherche des clés de type « FROM »
    [via la fonction « rechercher_cle_FROM »]
    Préparation et vérification de l'intégrité des données
    [via la procédure « verification_integrite »]
    Validation dans la base de données
    [via la procédure « traitement_BD »]

- Si la ligne de commande est relative à une mise à jour :

  Recherche du nom de la table affectée
  [effectué via un ensemble d'instructions]
  Formatage des colonnes et des valeurs à modifier
  [via la procédure « formatage_colonnes_update »]
  Formatage des colonnes et leur valeur reprises dans la condition
  [via la procédure « formatage_condition_delete_update »]
  Recherche des clés de type « FROM »
  [via la fonction « rechercher_cle_FROM »]
  Préparation et vérification de l'intégrité des données
  [via la procédure « verification_intégrite »]
  Recherche des clés de type « TO »
  [via la fonction « rechercher_cle_TO »]
  Préparation et vérification de l'intégrité des données
  [via la procédure « verification_intégrite »]
  Validation dans la base de données
  [via la procédure « traitement_BD »]

- Retour du paramètre de contrôle et réinitialisation des données pour le prochain appel. [effectué via un ensemble d'instructions]

La procédure « formatage_colonnes_insert »

Cette procédure ne nécessite pas de paramètres en entrée. Elle a pour but de générer dans une zone de type « structure » située en mémoire, les noms des colonnes et leur valeur correspondante qui vont être insérées dans la table. Le traitement effectué est le suivant :

- Initialisation de la structure recevant les informations concernant les colonnes à insérer avec leur valeur.
  [effectué via un ensemble d'instructions]

- Remplissage dans la structure des noms de colonnes à insérer.
  [effectué via un ensemble d'instructions]

- Remplissage dans la structure des valeurs de colonnes à insérer.
  [effectué via un ensemble d'instructions]

La procédure « formatage_colonnes_update »

Cette procédure ne nécessite pas de paramètres en entrée. Le traitement effectué est similaire au traitement effectué par la procédure « formatage_colonnes_insert » mais les zones situées en mémoire seront relatives aux noms et valeurs des colonnes qui vont être modifiées.

La procédure « formatage_condition_delete_update »

Cette procédure ne nécessite pas de paramètres en entrée. Elle a pour but de générer une zone de caractères située en mémoire et dans laquelle se trouvent de façon correctement formatée les conditions émises dans la ligne de commande dans le cas de suppressions ou de mises à jour. Le traitement consiste à spécifier dans cette zone de caractères le nom de la table et la colonne correspondante afin d'éviter toutes confusions lors de la phase de vérification de la cohérence des données. En effet, la requête de sélection permettant les vérifications fait appel à la technique de jointure et pour éviter une ambiguïté au niveau des noms de colonnes, il paraît nécessaire de spécifier le nom de la table qui lui est associée.

La fonction « rechercher_cle_TO »

Cette fonction ne nécessite pas de paramètres en entrée et a pour objectif de renvoyer l'ensemble des clés de type « TO » relatives à la table renseignée dans la ligne de commande. Une clé de type « TO » correspond à une clé partant d'une table vers une table de référence.
Le traitement effectué est le suivant :

- Remplissage de la structure contenant l'ensemble complet des clés étrangères du schéma de la base de données.
  [effectué via la procédure « generer_key » qui se trouve dans le fichier dynamique généré par DB-MAIN à partir du schéma physique]

- Recherche et renvoi de l'ensemble des clés de type « TO » correspondant à la table renseignée dans la ligne de commande.
  [effectué via un ensemble d'instructions]

La fonction « rechercher_cle_FROM »

Cette fonction ne nécessite pas de paramètres en entrée et a pour objectif de renvoyer l'ensemble des clés de type « FROM » relatives à la table renseignée dans la ligne de commande. Une clé de type « FROM » correspond à une clé d'une table de référence vers une autre table.
Le traitement effectué est le suivant :

- Remplissage de la structure contenant l'ensemble complet des clés étrangères du schéma de la base de données.
  [effectué via la procédure « generer_key » qui se trouve dans le fichier dynamique généré par DB-MAIN à partir du schéma physique]

- Recherche et renvoi de l'ensemble des clés de type « FROM » correspondant à la table renseignée dans la ligne de commande.
  [effectué via un ensemble d'instructions]

La procédure «verification_integrite »

Cette procédure ne nécessite pas de paramètres en entrée et a pour objectif d'effectuer une série de contrôles en fonction du type d'interaction avec la base de données. A la suite de ces contrôles, la ligne de commande est validée ou refusée. Les traitements sont les suivants :

En fonction du type d'interaction, les actions suivantes sont réalisées pour chacune des clés de type « TO » ou de type « FROM » de la table considérée dans la ligne de commande :

- A partir de la représentation de la clé étrangère de type « TO » ou de type « FROM » qui est prise en compte, extraction du nom des tables et de l'ensemble de leurs colonnes. Ensuite, mise en mémoire du nom des tables. En mémoire, il y a le nom de la table « origine » *(celle de la ligne de commande)* et le nom de la table « cible » *(celle qui est reliée via la clé étrangère).*
  [effectué via un ensemble d'instructions]

- Mise en mémoire dans une zone de type « structure » du nom de chacune des colonnes reprises dans la clé de type « TO » ou de type « FROM » qui est prise en compte lors de la vérification. Cette structure permet de faire la concordance entre les noms des colonnes des deux tables reliées par la clé étrangère. Cela est nécessaire lors de la création de la requête de sélection.
  [effectué via la procédure « remplir_struct_colonne »]

- Contrôle de la cohérence des données :

  - Si la ligne de commande est relative à une insertion :

    - Mise en place de la requête de sélection permettant de vérifier si la valeur des colonnes à insérer correspond bien à des enregistrements contenus dans la table de référence.
      [effectué via un ensemble d'instructions]
    - Exécution de la requête.
      [effectué via la procédure « requete_select »]
    - Gestion du résultat de la requête. Si le résultat est égal à 0, l'insertion dans la base de données est refusée. Si le résultat est strictement positif, l'insertion est acceptée.
      [effectué via un ensemble d'instructions]

    Il est à signaler que dans le cas où les colonnes reprises dans l'insertion ne font pas référence à des enregistrements contenus dans d'autres tables et, dans le cas ou les valeurs des colonnes insérées sont nulles, la requête valide d'office l'insertion. Un contrôle est également réalisé afin de vérifier le nombre exact de colonnes de référence reprises dans la clé.

- Si la ligne de commande est relative à une suppression :

  - Mise en place de la requête de sélection de type « jointure » permettant de vérifier si la valeur de la (des) colonne(s) à supprimer est contenue dans des enregistrements étant déjà référencés. [effectué via un ensemble d'instructions]
  - Exécution de la requête.
    [effectué via la procédure « requete_select »]
  - Gestion du résultat de la requête. Si le résultat est strictement positif, cela signifie que la suppression affecte des enregistrements référencés et l'instruction est alors refusée. Par contre, si le résultat est égal à 0, la suppression dans la base de données est acceptée.
    [effectué via un ensemble d'instructions]

- Si la ligne de commande est relative à une mise à jour :

  Première partie :
  - Mise en place de la requête de sélection permettant de vérifier si la valeur des colonnes à modifier correspond bien à des enregistrements contenus dans la table de référence.
    [effectué via un ensemble d'instructions]
  - Exécution de la requête.
    [effectué via la procédure « requete_select »]
  - Gestion du résultat de la requête. S'il est égal à 0, la mise à jour dans la base de données est refusée. Si le résultat est strictement positif, la mise à jour est acceptée.
    [effectué via un ensemble d'instructions]

  Deuxième partie :
  - Mise en place de la requête de sélection de type « jointure » permettant de vérifier si la valeur de la (des) colonne(s) à modifier est contenue dans des enregistrements étant déjà référencés.
    [effectué via un ensemble d'instructions]
  - Exécution de la requête.
    [effectué via la procédure « requete_select »]
  - Gestion du résultat de la requête. Si le résultat est strictement positif, cela signifie que la modification affecte des enregistrements référencés et l'instruction est alors refusée. Si le résultat est égal à 0, la mise à jour dans la base de données est acceptée.
    [effectué via un ensemble d'instructions]

  Il est à signaler que dans le cas où les colonnes reprises dans la mise à jour ne font pas référence à des enregistrements contenus dans d'autres tables et, dans le cas ou les valeurs des colonnes mises à jour sont nulles et facultatives, la requête valide la deuxième partie du traitement. Un contrôle est également réalisé afin de vérifier le nombre exact de colonnes de référence reprises dans la clé.

La procédure «remplir_struct_colonnes »

Cette procédure ne nécessite pas de paramètres en entrée et a pour objectif de remplir une zone de type « structure » située en mémoire qui contient les noms des colonnes des tables qui sont reliées entre elles via la clé de type « FROM » ou de type « TO ». Ces informations sont utilisées lors de la conception de la requête de sélection permettant le contrôle de l'intégrité des données.

La procédure «requete_select »

Cette procédure a besoin en entrée de la chaîne de caractères représentant l'instruction de sélection générée dans les différents cas repris dans la procédure « verifier_integrite ». La procédure va exécuter l'instruction de sélection auprès de la base de données et va ensuite garnir une zone en mémoire dans laquelle se trouvera le résultat de la sélection.

La procédure «traitement_BD »

Cette procédure ne nécessite pas de paramètres en entrée et a pour objectif d'interagir avec la base de données pour effectuer l'instruction insérée en paramètre, si toutefois, le résultat des contrôles de cohérence des données le permet.

La procédure « definir_nom_lien_ODBC »

Cette procédure reçoit et enregistre le paramètre d'entrée contenant une chaîne de caractères dans laquelle se trouve le nom de lien ODBC défini par l'utilisateur.
Cette procédure est appelée à partir du programme client utilisant la librairie de fonctions présentée dans le chapitre 6.

La fonction « DllMain »

Cette fonction est à ajouter en tant que fonction par défaut pour la création d'une librairie de fonctions *(DLL)*.

Chapitre 8 : Etude de cas

Ce chapitre va illustrer, via la présentation d'un exemple concret, les différents aspects du SGBD MySQL abordés dans ce mémoire. Dans un premier temps, les phases de conception, de codage et de déploiement d'une base de données sont illustrées. Ensuite, un exemple illustratif d'un programme n'utilisant pas la méthode proposée pour la gestion des clés étrangères est présenté de même qu'un programme équivalent faisant appel, quant à lui, aux outils conçus pour la gestion des clés étrangères. Cette comparaison est réalisée dans le but de mettre en évidence les traitements qu'il est possible d'éviter en appliquant la méthode proposée dans les chapitres précédents.

8.1. Mise en place d'une base de données MySQL

Cette partie expose un exemple de conception d'une base de données MySQL en partant de la description du domaine d'application jusqu'à l'implémentation réelle. Etant donné que l'atelier logiciel DB-MAIN est méthodologiquement neutre, il existe beaucoup de façons de mettre en place une base de données notamment en fonction de l'utilisation ou non des assistants et des scripts prédéfinis ou créés par l'utilisateur.

8.1.1. Description du domaine d'application

Le domaine dans lequel va évoluer l'exemple de cette étude de cas est relatif à la gestion des commandes dans un magasin.
Le magasin propose différents produits classés en catégories. Les trois catégories de produits proposés par le magasin sont les suivantes : « Meubles », « Electro-ménager » et « Alimentaire ». Les produits commandés proviennent de différents fournisseurs. Les informations relatives à un produit sont : un libellé et une date de fabrication. Un fournisseur possède un nom et une localisation. Une commande est relative à un seul client, contient un numéro, une date et est caractérisée par un état d'avancement *(une commande est toujours livrée, payée et clôturée)*. Un client est soit un client de type « régulier » ou de type « occasionnel » et est enregistré avec les informations suivantes : un nom, un prénom, une adresse. Enfin, la quantité d'un produit commandé chez un fournisseur est également conservée.

8.1.2. Analyse conceptuelle

La traduction du domaine d'application dans un schéma Entité-Association aboutit au schéma conceptuel représenté sur la figure 8.1. Le schéma conceptuel est généré à partir de l'atelier logiciel DB-MAIN et contient des types d'entités, des types d'associations, des attributs, des identifiants et des contraintes conformément à ce qui a été défini dans les sections 3.2 et 4.1.1.

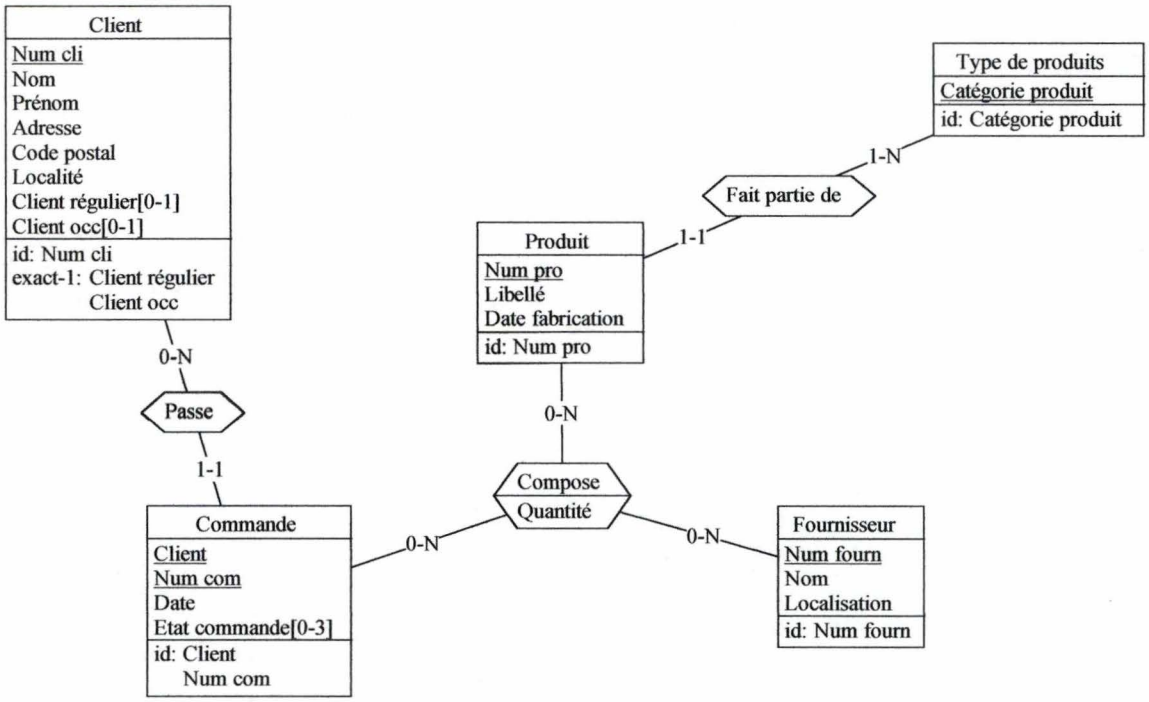


Figure 8.1. : Etude de cas : schéma conceptuel

8.1.3. Conception logique

La phase de conception logique a pour objectif de mettre en place un schéma logique relationnel en faisant subir au schéma conceptuel un ensemble de transformations.
Avant d'entamer le processus de transformations, le concepteur de la base de données crée les types spécifiques « SET » et « ENUM » dans l'atelier logiciel DB-MAIN comme cela est décrit dans la section 5.4.1.
Ensuite, plusieurs types d'assistants de transformations sont disponibles. Pour aboutir au schéma logique représenté sur la figure 8.2, le concepteur de la base de données a utilisé l'assistant de transformations globales avancées dans lequel il a chargé le script correspondant à la génération d'un schéma logique relationnel appliquant le plan de transformation décrit dans la section 4.2.1.
Comme particularités de ce plan de transformation, il est à noter que le script a fusionné les deux colonnes « Client_régulier » et « Client_occ » en une seule colonne *(« TYPE_CLIENT »)* dont le domaine est « ENUM » et dont les valeurs associées sont « régulier » et « occasionnel ». Le choix du domaine « ENUM » résulte du fait que les deux champs sont distincts, qu'ils sont booléens et facultatifs, qu'ils représentent le même type d'information et qu'une contrainte de type « exactement un » n'autorise qu'un seul type par client. Une autre colonne de type « ENUM » a été déclarée. Il s'agit de la colonne « CATEGORIE_PRODUIT » dans la table « PRODUIT ». Dans le schéma conceptuel de la figure 8.1, cette information était représentée sous la forme d'un type d'entités mais étant donné que le type d'entités « Type de produits » ne

contient qu'un seul attribut, que sa liste de valeur est limitée dans le domaine d'application de l'exemple et que chaque produit fait partie d'une et une seule catégorie de produits, le script avait devant lui un cas l'autorisant à transformer ce type d'entités en une colonne de type « ENUM ». Le script a également transformé l'attribut multivalué « ETAT_COMMANDE » en une colonne de type « SET ». Vu que le domaine d'application n'autorise qu'un ensemble de valeurs prédéfinies, stables et peu changeantes, il a été jugé opportun de définir un domaine de valeurs de type « SET » pour cet attribut. L'explication de ces trois transformations figure dans la section 4.2.2.
Comme dernières transformations effectuées par le concepteur de la base de données, il est à noter que le type d'associations complexe, appelé « Compose » dans le schéma conceptuel, a été transformé en une table et que cette table a été renommée « DETAIL » pour une meilleure représentation. Les noms des divers éléments du schéma ont été mis en lettres majuscules et les accents ont été supprimés.
Sur le schéma logique de la figure 8.2, des tables, des colonnes et des clés sont représentées comme cela a été évoqué dans les parties 3.3, 4.1.2 et 4.2 de ce mémoire.

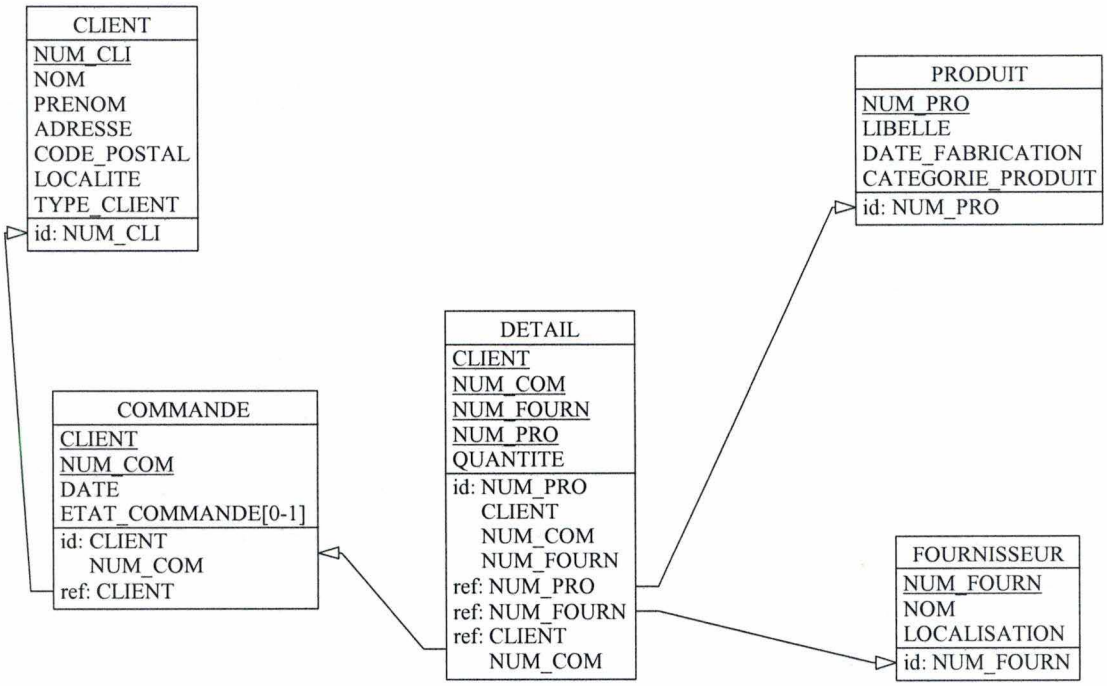


Figure 8.2. : Etude de cas : schéma logique

8.1.4. Conception physique

Lors de l'étape de conception physique, les clés d'accès et les types de stockage des tables sont définis. Avant d'entamer cette phase dans l'atelier logiciel DB-MAIN, le concepteur de la base de données doit veiller à créer la propriété dynamique correspondante aux types de stockage via la technique expliquée dans la section 5.5.2. Ensuite, il fait appel à l'assistant de transformations globales dans lequel il charge le script prédéfini lui permettant de générer les transformations physiques de base pour un schéma relationnel *(définition des clés d'accès sur les identifiants et les colonnes qui référencent une autre table)*. Après cela, il traite les colonnes pour lesquelles il juge opportun de définir une clé d'accès. Ce traitement s'effectue en fonction des besoins nécessaires pour la gestion des informations qui seront stockées dans les différentes tables. Dans le cas de l'exemple présenté dans ce chapitre, les colonnes

référençant les noms et les dates sont indexées afin de faciliter les opérations de recherches.
La mise en place des types de stockage pour chacune des tables s'effectue, quant à elle, en définissant le type approprié via les propriétés dynamiques de la table. Le choix se fait en fonction des besoins en rapidité ou en sécurité. Sur la figure 8.3, le concepteur a défini les tables en rouge *(« COMMANDE » et « DETAIL »)* de type « HEAP » pour la rapidité et les autres tables *(« CLIENT », « PRODUIT » et « FOURNISSEUR »)* sont de type « MyISAM », type par défaut de MySQL.
Le schéma physique résultant de cette phase de conception est illustré sur la figure 8.3.

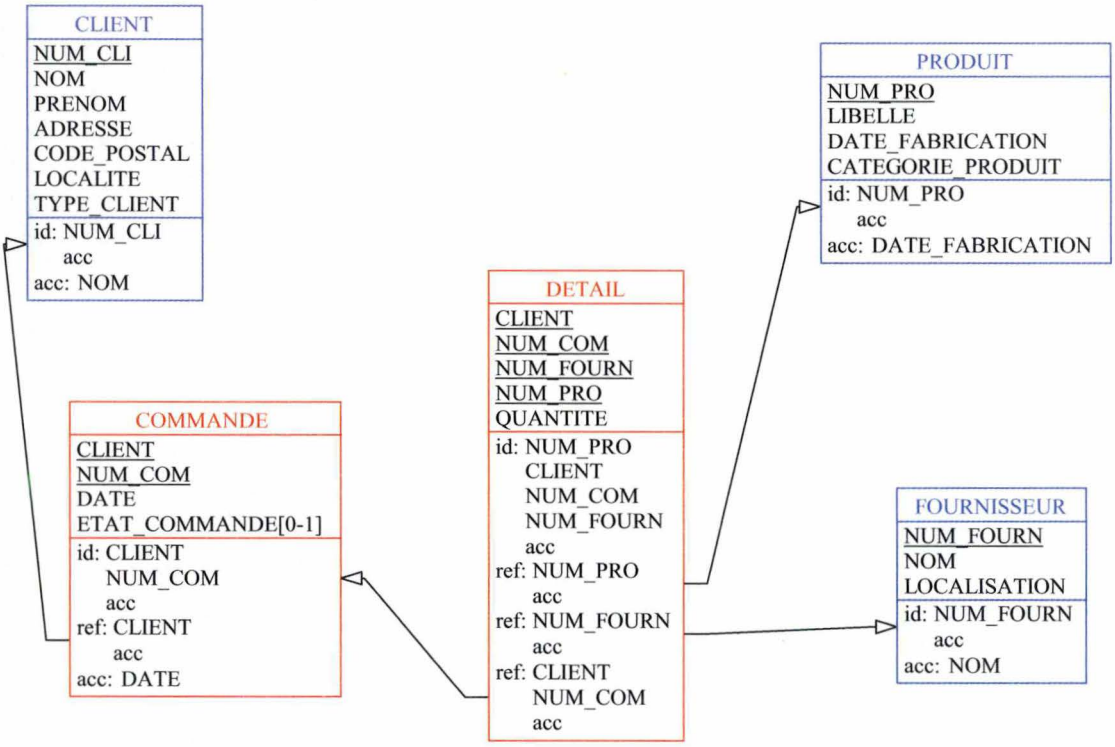


Figure 8.3. : Etude de cas : schéma physique

8.1.5. Codage

Le code de création de la base de données est généré à partir d'un module développé et accessible depuis l'atelier logiciel DB-MAIN *(le module « mysql.oxo » présenté dans les sections 5.6.1 et 7.1)*. Le concepteur de la base de données doit mentionner un nom de fichier dans lequel sont insérées les instructions de création de la base de données *(dans le cas présent, il s'agit du fichier « magasin.mysql »)*.
Le script de création de la base de données correspondant au schéma physique illustré sur la figure 8.3 est repris à la page suivante. Dans ce script, la première instruction crée la base de données appelée « Magasin » et la seconde instruction sélectionne cette base de données qui vient d'être créée. Cette sélection est nécessaire pour pouvoir définir par la suite les tables, les contraintes et les index.
Concernant les index, il est à noter également qu'ils ne sont pas déclarés si les colonnes indexées sont en même temps identifiantes. En effet, comme mentionné dans la section 2.2.7, le SGBD MySQL définit par défaut un index sur les colonnes de type « PRIMARY KEY » et « UNIQUE ».

```
-- ****************************
-- * génération de code MySQL *
-- ****************************
-- Database Section
-- _____________________
create database Magasin;
use Magasin;

-- Table Section
-- _____________
create table CLIENT (
        NUM_CLI char(5) not null,
        NOM char(20) not null,
        PRENOM char(20) not null,
        ADRESSE char(60) not null,
        CODE_POSTAL char(6) not null,
        LOCALITE char(20) not null,
        TYPE_CLIENT enum('Regulier', 'Occasionnel') not null,
        primary key (NUM_CLI))
        type=MyISAM;

create table COMMANDE (
        CLIENT char(5) not null,
        NUM_COM char(5) not null,
        DATE date not null,
        ETAT_COMMANDE set('Livree', 'Payee', 'Cloturee'),
        primary key (CLIENT, NUM_COM))
        type=HEAP;

create table FOURNISSEUR (
        NUM_FOURN char(3) not null,
        NOM char(20) not null,
        LOCALISATION char(30) not null,
        primary key (NUM_FOURN));

create table PRODUIT (
        NUM_PRO char(3) not null,
        LIBELLE char(25) not null,
        DATE_FABRICATION date not null,
        CATEGORIE_PRODUIT enum('Meubles', 'Electro-menager', 'Alimentaire')
        not null,
        primary key (NUM_PRO));

create table DETAIL (
        CLIENT char(5) not null,
        NUM_COM char(5) not null,
        NUM_FOURN char(3) not null,
        NUM_PRO char(3) not null,
        QUANTITE int not null,
```

```
        primary key (NUM_PRO, CLIENT, NUM_COM, NUM_FOURN))
        type=HEAP;

-- Constraints Section
-- _____________________
-- !!Génération des contraintes,
-- Dans MySQL, les foreign keys et les checks
-- seront DECLARES mais PAS GERES !!
-- _____________________
alter table COMMANDE add constraint FKPASSE
        foreign key (CLIENT)
        references CLIENT;

alter table DETAIL add constraint FKCOM_PRO
        foreign key (NUM_PRO)
        references PRODUIT;

alter table DETAIL add constraint FKCOM_FOU
        foreign key (NUM_FOURN)
        references FOURNISSEUR;

alter table DETAIL add constraint FKCOM_COM
        foreign key (CLIENT, NUM_COM)
        references COMMANDE;

-- Index Section
-- _____________
create index CLIENT_NOM
        on CLIENT (NOM);

create index FKPASSE
        on COMMANDE (CLIENT);

create index COMMANDE_DATE
        on COMMANDE (DATE);

create index FOURNISSEUR_NOM
        on FOURNISSEUR (NOM);

create index PRODUIT_DATE
        on PRODUIT (DATE_FABRICATION);

create index FKCOM_PRO
        on DETAIL (NUM_PRO);

create index FKCOM_FOU
        on DETAIL (NUM_FOURN);

create index FKCOM_COM
        on DETAIL (CLIENT, NUM_COM);
```

8.1.6. Déploiement de la base de données

Pour réaliser le déploiement de la base de données, il suffit de lancer l'environnement de MySQL et d'importer en même temps le fichier dans lequel sont définies les instructions de création de la base de données. Cette manipulation s'exécute en respectant la syntaxe de l'instruction dans laquelle est renseigné l'emplacement du fichier de création de la base de données. La procédure est illustrée par la figure 8.4.

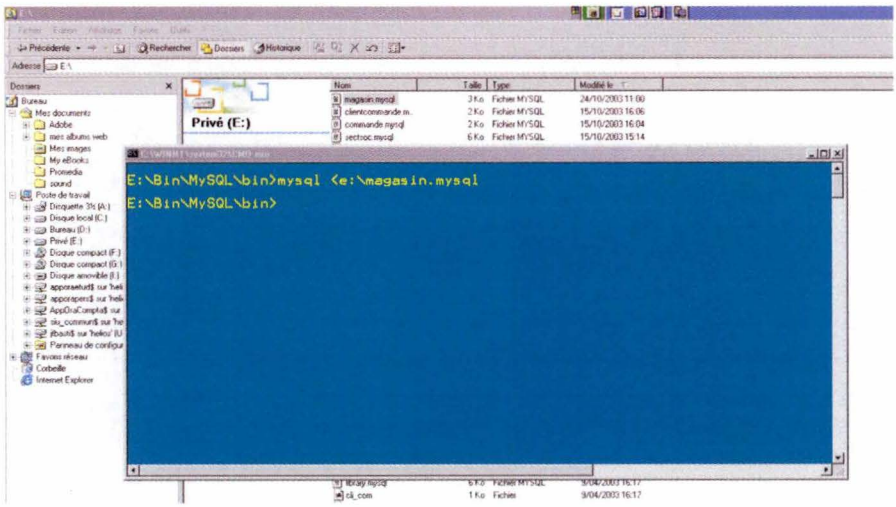


Figure 8.4. : Etude de cas : déploiement de la base de données

Après avoir déployé la base de données, il est possible de visualiser la structure et les caractéristiques des tables qui viennent d'être créées. La figure 8.5 illustre d'ailleurs la description des tables « CLIENT », « COMMANDE », « PRODUIT », « FOURNISSEUR » et « DETAIL » créées dans l'environnement de MySQL.

```
mysql> desc client;
+--------------+-----------------------------------+------+-----+----------+-------+
| Field        | Type                              | Null | Key | Default  | Extra |
+--------------+-----------------------------------+------+-----+----------+-------+
| NUM_CLI      | char(5)                           |      | PRI |          |       |
| NOM          | char(20)                          |      | MUL |          |       |
| PRENOM       | char(20)                          |      |     |          |       |
| ADRESSE      | char(60)                          |      |     |          |       |
| CODE_POSTAL  | char(6)                           |      |     |          |       |
| LOCALITE     | char(20)                          |      |     |          |       |
| TYPE_CLIENT  | enum('Regulier','Occasionnel')    |      |     | Regulier |       |
+--------------+-----------------------------------+------+-----+----------+-------+
7 rows in set (0.00 sec)

mysql> desc commande;
+---------------+-------------------------------------+------+-----+---------------------+-------+
| Field         | Type                                | Null | Key | Default             | Extra |
+---------------+-------------------------------------+------+-----+---------------------+-------+
| CLIENT        | char(5)                             |      | PRI |                     |       |
| NUM_COM       | char(5)                             |      | PRI |                     |       |
| DATE          | date                                |      | MUL | 0000-00-00          |       |
| ETAT_COMMANDE | set('Livree','Payee','Cloturee')    | YES  |     | NULL                |       |
+---------------+-------------------------------------+------+-----+---------------------+-------+
4 rows in set (0.00 sec)

mysql> desc produit;
+-------------------+-----------------------------------------------------+------+-----+---------------------+-------+
| Field             | Type                                                | Null | Key | Default             | Extra |
+-------------------+-----------------------------------------------------+------+-----+---------------------+-------+
| NUM_PRO           | char(3)                                             |      | PRI |                     |       |
| LIBELLE           | char(25)                                            |      |     |                     |       |
| DATE_FABRICATION  | date                                                |      | MUL | 0000-00-00          |       |
| CATEGORIE_PRODUIT | enum('Meuble','Electro-menager','Alimentaire')      |      |     | Meuble              |       |
+-------------------+-----------------------------------------------------+------+-----+---------------------+-------+
4 rows in set (0.00 sec)

mysql> desc fournisseur;
+--------------+----------+------+-----+---------+-------+
| Field        | Type     | Null | Key | Default | Extra |
+--------------+----------+------+-----+---------+-------+
| NUM_FOURN    | char(3)  |      | PRI |         |       |
| NOM          | char(20) |      | MUL |         |       |
| LOCALISATION | char(30) |      |     |         |       |
+--------------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc detail;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| CLIENT    | char(5) |      | PRI |         |       |
| NUM_COM   | char(5) |      | PRI |         |       |
| NUM_FOURN | char(3) |      | PRI |         |       |
| NUM_PRO   | char(3) |      | PRI |         |       |
| QUANTITE  | int(11) |      |     | 0       |       |
+-----------+---------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Figure 8.5. : Etude de cas : description des tables de la base de données dans l'environnement de MySQL

8.2. Gestion des clés étrangères dans un programme

Cette partie va illustrer les avantages apportés par le module de gestion des clés étrangères. L'utilisation et la description des fonctions contenues dans ce module sont reprises dans les sections 6.3, 6.4 et 7.3.

8.2.1. Hypothèses

Pour expliquer de façon précise les avantages de ce module de gestion de clés étrangères, le programme qui va être illustré consiste en une fonction d'insertion d'une ligne de détail pour la commande d'un client. Il s'agit dans ce cas-ci d'une procédure écrite avec le langage de programmation C/C++ mais la librairie de fonctions peut être importée dans un autre environnement de programmation. Pour simplifier le code, les hypothèses suivantes ont été émises :

- Le lien ODBC vers la base de données MySQL a été défini préalablement dans les deux cas présentés.
- Les initialisations des zones de travail sont effectuées via des procédures.
- Dans le premier exemple, les interactions avec la base de données pour les requêtes de sélections sont effectuées via une fonction dans laquelle est entrée en paramètre l'instruction de type « Select » et en retour, le nombre d'enregistrements est renvoyé.
- Les valeurs des colonnes « CLIENT », « NUM_COM », « NUM_FOURN », « NUM_PRO » et « QUANTITE » sont passées comme paramètres d'entrée de la fonction présentée.

8.2.2. Exemple d'une fonction n'utilisant pas la fonction de validation

```
// Déclaration des variables
char[256] requete_select;
char[256] ligne_insert ;
int result ;
bool insertion_acceptee ;

// Procédure d'insertion d'une ligne Detail
void insertion_detail (char[5] var_client, char[5] var_num_com, char[3] var_num_fourn, char[3] var_num_pro, int var_quantite)
{
// L'insertion est par défaut acceptée
        insertion_acceptee = TRUE ;
// Procédure qui initialise la zone requete_select
        initialisation_requete_select () ;
// Mise en place de la requête qui va vérifier si le numéro de commande pour le client existe bien dans la table
// « COMMANDE »
        strcpy (requete_select, ''SELECT COUNT(*) FROM COMMANDE WHERE '') ;
        strcat (requete_select, ''CLIENT = '') ;
        strcat (requete_select, var_client) ;
        strcat (requete_select, '' AND NUM_COM = '') ;
        strcat (requete_select, var_num_com) ;
        strcat (requete_select, '' ;\0 '') ;
// Lancement du Select via une procédure prédéfinie ... en retour, le nombre d'enregistrements
        result = interactionBD_select (requete_select) ;
// Si le résultat est égal à 0, on refuse l'insertion
        if (result ==0)
        insertion_aceptee = FALSE ;
// Procédure qui réinitialise la zone requete_select
        initialisation_requete_select () ;
// Mise en place de la requête qui va vérifier si le numéro de fournisseur existe bien dans la table « FOURNISSEUR »
        strcpy (requete_select, ''SELECT COUNT(*) FROM FOURNISSEUR WHERE '') ;
        strcat (requete_select, ''NUM_FOURN = '') ;
        strcat (requete_select, var_num_fourn) ;
        strcat (requete_select, '' ;\0'') ;
```

```
// Lancement du Select via une procédure prédéfinie ... en retour, le nombre d'enregistrements
        result = interactionBD_select (requete_select) ;
// Si le résultat est égal à 0, on refuse l'insertion
        if (result ==0)
        insertion_aceptee = FALSE ;
// Procédure qui réinitialise la zone requete_select
        initialisation_requete_select () ;
// Mise en place de la requête qui va vérifier si le numéro de produit existe bien dans la table « PRODUIT »
        strcpy (requete_select, ''SELECT COUNT(*) FROM PRODUIT WHERE '') ;
        strcat (requete_select, ''NUM_PRO = '') ;
        strcat (requete_select, var_num_pro) ;
        strcat (requete_select, '' ;\0 '') ;
// Lancement du Select via une procédure prédéfinie ... en retour, le nombre d'enregistrements
        result = interactionBD_select (requete_select) ;
// Si le résultat est égal à 0, on refuse l'insertion
        if (result ==0)
        insertion_aceptee = FALSE ;

// Si l'insertion est toujours acceptée, on crée une ligne de commande pour l'insertion dans la table
        if (insertion_acceptee == TRUE)
        {
// Procédure qui initialise la zone ligne_insert
                initialisation ligne_insert() ;
// Remplissage de la ligne d'insertion
                strcpy (ligne_insert, ''INSERT INTO DETAIL '') ;
                strcat (ligne_insert , ''(CLIENT,NUM_COM,NUM_FOURN,NUM_PRO,QUANTITE) VALUES '' ) ;
                strcat (ligne_insert, ''(var_client,var_num_com,var_num_fourn,var_num_pro,var_quantite) ;\0 '') ;
// Lancement de l'instruction dans la base de données via une procédure
                interactionBD_insert(ligne_insert) ;
        }
}
```

Le code ci-dessus illustre bien la gestion manuelle et rigoureuse qu'il faut exécuter avant de rendre effective l'insertion dans la table « DETAIL ». De plus, il est important de souligner que le programmeur doit connaître parfaitement les noms des tables et des champs pour la mise en place des requêtes de sélections. Il est à noter également que le code devra être reparcouru dans le cas de l'ajout d'une table, d'une colonne ou encore dans le cas d'un changement de nom d'une table ou d'une colonne.

8.2.3. Exemple de fonction utilisant la fonction de validation

Avant de faire appel à la fonction « validation » du module de gestion, il faut avoir généré auparavant la structure des clés étrangères sous forme d'un fichier informatique qui sera consulté lors de la validation de l'instruction d'insertion. Pour créer ce fichier, le générateur correspondant est appelé à partir du schéma physique défini dans DB-MAIN. L'appel et la description de ce générateur sont expliqués dans les sections 5.6.2 et 7.2. Le résultat correspondant à l'exemple présenté dans cette étude de cas est repris ci-dessous.

```
struct
{char liste_foreignkeyTO[20000] ;
} foreignkeyTO ;
struct
{char liste_foreignkeyFROM[20000] ;
} foreignkeyFROM ;


void generer_key()
{
strcpy (foreignkeyTO.liste_foreignkeyTO,"\0");
strcat (foreignkeyTO.liste_foreignkeyTO,"COMMANDE@CLIENT#CLIENT@NUM_CLI#;\0");
strcat (foreignkeyTO.liste_foreignkeyTO,"DETAIL@NUM_PRO#PRODUIT@NUM_PRO#;\0");
strcat (foreignkeyTO.liste_foreignkeyTO,"DETAIL@NUM_FOURN#FOURNISSEUR@NUM_FOURN#;\0");
strcat (foreignkeyTO.liste_foreignkeyTO,"DETAIL@CLIENT,NUM_COM#COMMANDE@CLIENT,NUM_COM#;\0");
```

```
strcpy (foreignkeyFROM.liste_foreignkeyFROM,"\0");
strcat (foreignkeyFROM.liste_foreignkeyFROM,"CLIENT@NUM_CLI#COMMANDE@CLIENT#;\0");
strcat (foreignkeyFROM.liste_foreignkeyFROM,"PRODUIT@NUM_PRO#DETAIL@NUM_PRO#;\0");
strcat (foreignkeyFROM.liste_foreignkeyFROM,"FOURNISSEUR@NUM_FOURN#DETAIL@NUM_FOURN#;\0");
strcat
(foreignkeyFROM.liste_foreignkeyFROM,"COMMANDE@CLIENT,NUM_COM#DETAIL@CLIENT,NUM_COM#;\0");
}
```

La fonction d'insertion d'une ligne dans la table « DETAIL » s'effectue alors comme suit :

```
// Importation de la fonction de validation qui est contenue dans la librairie de fonctions ajoutée dans l'environnement
// de l'application
extern "C" __declspec(dllimport)  int validation (char* ligne_from_ext);

// Déclaration des variables
char[256] ligne_insert ;
int resultat ;

// Procédure d'insertion d'une ligne Detail
void insertion_detail (char[5] var_client, char[5] var_num_com, char[3] var_num_fourn, char[3] var_num_pro, int var_quantite)
{
// Procédure qui initialise la zone ligne_insert
        initialisation ligne_insert() ;
// Remplissage de la ligne d'insertion
        strcpy (ligne_insert, ''INSERT INTO DETAIL '') ;
        strcat (ligne_insert , ''(CLIENT,NUM_COM,NUM_FOURN,NUM_PRO,QUANTITE) VALUES '' ) ;
        strcat (ligne_insert, ''(var_client,var_num_com,var_num_fourn,var_num_pro,var_quantite) ;\0 '') ;
// Lancement de l'instruction dans la base de données via une procédure
        resultat = validation(ligne_insert) ;
}
```

Le code ci-dessus est simplifié par rapport à la version ne faisant pas appel à cette procédure de validation. De plus, cette façon de faire est beaucoup plus souple en ce qui concerne les modifications ou évolutions de la base de données. En effet, le programme est moins dépendant de la structure de la base de données. Cela simplifie fortement le travail du programmeur d'applications.

Chapitre 9 : Conclusion

Les deux grands objectifs de ce mémoire étaient les suivants :

- Analyser le système de gestion de bases de données MySQL en décrivant le processus complet d'élaboration d'une base de données MySQL et en tenant compte des spécificités propres à ce SGBD.

- Proposer une méthode de gestion des clés étrangères dans une base de données MySQL en utilisant la stratégie « No Action ». Cette stratégie consiste à refuser une instruction d'interaction avec la base de données si cette instruction altère la cohérence et l'intégrité des données.

Ces deux objectifs ont été atteints et les solutions apportées ont été exposées tout au long des chapitres composant ce document. En effet, les principes de fonctionnement du SGBD MySQL ont d'abord été exprimés. Ensuite, un rappel théorique de la création d'une base de données a été effectué. Ces notions théoriques ont été, par après, appliquées à la conception d'une base de données MySQL. Cette analyse commence à partir du modèle le plus abstrait pour aboutir à la phase de déploiement concret en passant par les phases de transformation et d'optimisation. Pour la mise en pratique de ces différentes étapes de conception d'une base de données, l'atelier de génie logiciel DB-MAIN a été utilisé et la marche à suivre relative à l'utilisation de cet outil pour mettre en place une base de données MySQL est illustrée dans ce document.
Deux modules ont été également mis en place et sont détaillés dans ce mémoire. Le premier module est relatif à la génération d'un fichier contenant les instructions de création de la base de données et le second, génère un fichier représentatif des clés étrangères qui va être utilisé pour atteindre le deuxième objectif de ce mémoire.
Pour la résolution du deuxième objectif, une solution méthodologique de gestion des clés étrangères a été mise au point avec comme résultat concret une librairie de fonctions utilisables à partir de différents programmes d'applications. Dans ce document, les différentes techniques de programmation utilisant les services d'une base de données sont présentées et sont suivies de la description des méthodes d'importation et d'appel de l'outil mis en place. Pour terminer, la description technique de chaque module développé dans le cadre de ce mémoire et une étude de cas complète figurent également dans ce document.

Les principaux avantages des solutions apportées aux objectifs énoncés ci-dessus sont les suivants :

- Utilisation et extension d'un outil spécifiquement dédié à la mise en place et à la gestion des bases de données
- Solution souple et extensible de gestion des contraintes de type « clés étrangères »

Pour situer la solution méthodologique décrite dans ce mémoire par rapport à d'autres alternatives, un tableau va être présenté. Le comparatif repris dans ce tableau a pour but d'exposer les différentes caractéristiques des outils actuels permettant la gestion de clés étrangères dans le SGBD MySQL. Grâce à ce tableau, les concepteurs de bases de données et les programmeurs pourront choisir la solution la plus adaptée à leurs besoins.

Les solutions reprises dans le tableau sont les suivantes :

- La librairie de fonctions mise en place dans le cadre de ce mémoire.
- Les outils graphiques implémentés avec la fonctionnalité de gestion des clés étrangères. Ces outils permettent de gérer de manière visuelle une base de données.
- Les programmes d'interaction avec une base de données développés avec un langage de programmation et interagissant avec la base de données via des interfaces de programmation.
- Le moteur de stockage « InnoDB » intégré dans certaines versions du SGBD MySQL.

Les critères de comparaison sont les suivants :

- Le format sous lequel se présente la solution.
- Les types de tables MySQL supportés. Comme MySQL propose plusieurs types de tables, les solutions proposées ne sont pas toutes valables sur l'ensemble des types proposés.
- Gestion de l'intégrité des données. Celle-ci peut se faire de manière automatisée ou de manière programmée manuellement.
- Etendue de la fonctionnalité de gestion des clés étrangères. Ce critère analyse la portée de chaque solution.
- Mise en place de la solution. Ce critère analyse la façon dont la solution est déployée dans le système.
- Compatibilité avec les différentes versions de MySQL

	Librairie de fonctions présentée dans le mémoire	**Outils graphiques (avec gestion des clés étrangères)**	**Programmes externes d'interaction avec la BD**	**Moteur de stockage « InnoDB »**
Format	Librairie contenant un ensemble de fonctions et procédures qu'il est possible d'importer et d'utiliser dans les applications clientes.	Console graphique dans laquelle il faut définir, au préalable, les clés étrangères reliant les tables de la base de données MySQL. Outil permettant de travailler de façon visuelle.	Applications interagissant avec une base de données via les nombreuses API. Les types d'applications qu'il est possible de mettre en place sont multiples et variés.	Moteur de stockage relatif à un type particulier de table MySQL. Les contraintes relatives aux clés étrangères sont déclarées explicitement lors de la création de la table.
Types de tables MySQL supportés	Tous les types de tables MySQL	Tous les types de tables MySQL	Tous les types de tables MySQL	Uniquement les types de tables « InnoDB »
Gestion de l'intégrité des données	Gestion automatisée	Gestion automatisée	Gestion manuelle	Gestion automatisée
Etendue de la fonctionnalité de gestion des clés étrangères	La fonction de gestion des clés étrangères peut être importée dans tout programme client interagissant avec une base de données MySQL.	La fonction de gestion des clés étrangères est utilisée pour valider l'instruction générée par l'utilisateur dans les écrans graphiques de l'environnement uniquement.	La gestion des clés étrangères s'effectue à l'intérieur du programme mis en place. Cette gestion est personnalisée, programmée manuellement et est fortement dépendante de la structure de la base de données.	La fonction de gestion est située au niveau du SGBD MySQL. Cela permet d'avoir une étendue la plus large possible. Toutes les interactions avec la base de données *(écrans de saisie, programmes clients)* peuvent bénéficier de la fonctionnalité de gestion des clés étrangères.

	Librairie de fonctions présentée dans le mémoire	Outils graphiques (avec gestion des clés étrangères)	Programmes externes d'interaction avec la BD	Moteur de stockage « InnoDB »
Mise en place du système	La représentation dynamique des clés étrangères s'effectue de manière automatisée à partir du schéma physique mis en place dans l'atelier logiciel. Ensuite, la fonction de gestion des clés étrangères de la librairie de fonctions est importée dans un programme client afin de valider les interactions entre ce programme et la base de données.	Dans l'environnement de l'outil graphique, l'utilisateur doit déclarer manuellement les clés étrangères de la base de données avec laquelle il interagit. Par après, toutes les interactions seront validées via le mécanisme de contrôle de gestion des clés étrangères mis en place dans ces outils.	La programmation de la déclaration et de la gestion des clés étrangères s'effectue manuellement via les instructions insérées dans les programmes. Toutes les interactions avec la base de données seront analysées via les contrôles mis en place par le programmeur dans ce genre d'applications.	Le moteur de stockage est intégré au SGBD MySQL. La déclaration des clés étrangères s'effectue via les instructions du langage de définition de données *(DDL)*. Toute interaction avec la base de données sera validée via le mécanisme de gestion des clés étrangères mis en place dans ce moteur de stockage.
Compatibilité avec les différentes versions de MySQL	Compatible avec toutes les versions de MySQL.	Logiciel « abeille » [Jeta Software 2003] : compatible avec les versions 3.23 et 4.	Compatible avec toutes les versions de MySQL.	Disponible sur certaines versions de MySQL depuis la version 3.23.34. Cependant, la mise en place de la fonctionnalité de gestion des clés étrangères a débuté lors du lancement de la version 3.23.44 de MySQL.

Le tableau ci-dessus met en évidence les points forts et les points faibles de chacune des solutions. Ce tableau va être analysé dans les paragraphes qui vont suivre.

La librairie de fonctions, mise en place dans le cadre de ce mémoire, a pour avantage de supporter tous les types de tables du SGBD MySQL et cela pour l'ensemble des versions de ce SGBD. La gestion des clés étrangères est automatisée et est effectuée via une fonction de validation. Toutefois, il faudra veiller à tenir une certaine rigueur lors du déploiement de la solution *(mise en place complète de la base de données dans l'atelier logiciel, génération de la structure des clés étrangères et importation des fonctions dans les programmes clients).*

Les outils graphiques possèdent, quant à eux, l'avantage suivant : faciliter la déclaration des clés étrangères et les interactions avec la base de données grâce à l'aspect visuel. Toutefois, ces outils sont « fermés » car la fonctionnalité de gestion des clés étrangères ne peut pas être utilisée ailleurs que dans l'environnement de l'outil graphique.

Les programmes externes d'interaction avec la base de données sont actuellement les plus utilisés en association avec une base de données MySQL car ces programmes permettent de créer de façon personnalisée des applications utilisant les services d'une base de données. Ils sont compatibles avec tous les types de tables MySQL et supportent l'ensemble des versions de ce SGBD. Cependant, l'inconvénient principal de cette solution est le suivant : la gestion des clés étrangères doit être programmée de façon complètement manuelle.

Le moteur de stockage « InnoDB » présente un certain nombre d'avantages. Il est intégré au SGBD MySQL et cela élargit considérablement l'étendue de la fonctionnalité de gestion des clés étrangères. Grâce à cela, il n'est plus nécessaire de déployer cette fonctionnalité à l'extérieur du SGBD. De plus, le moteur de stockage « InnoDB » permet de déclarer les clés étrangères en utilisant la syntaxe du langage de définition de données *(DDL)* qui est normalisé. Il est à noter cependant que ce moteur de stockage n'est pas présent sur les premières versions de MySQL et qu'il n'est possible de profiter de cette fonctionnalité que sur des tables de type « InnoDB ».

Dans les versions futures du système de gestion de bases de données MySQL, il est annoncé que la déclaration explicite des clés étrangères et leur gestion seront implémentées et que cette fonctionnalité sera d'application sur l'ensemble des types de table proposés par MySQL. Cependant, la mise en place de cette fonctionnalité n'altère pas la raison d'exister de la méthodologie de gestion des contraintes décrite dans ce mémoire. Les raisons qui le prouvent sont les suivantes :

- Il existera toujours des bases de données implémentées avec des anciennes versions du SGBD MySQL.

- Il existe d'autres perspectives et améliorations à cette méthodologie. Elles sont présentées dans la section suivante.

Perspectives et améliorations

Les perspectives et améliorations qu'il est possible d'apporter en complément à ce mémoire se situent essentiellement au niveau de l'utilisation de la solution proposée pour la gestion des clés étrangères de MySQL. En effet, étant donné l'aspect générique de cette solution, il est possible d'employer les mêmes notions pour la mise en place d'autres types de contraintes dans un système de gestion de bases de données. Grâce à cette méthodologie, il serait réalisable de gérer de façon automatique des contraintes permettant de vérifier les valeurs des données ou encore des contraintes d'existence dans le SGBD MySQL.

Une autre perspective de développement consisterait à modifier la stratégie utilisée dans la librairie de fonctions. Cette modification nécessiterait l'ajout du code correspondant aux règles de gestion relatives à une autre stratégie dans la fonction de validation. Par exemple, il serait possible de substituer la stratégie « No Action » par la stratégie de type « Cascade » qui répercute la valeur des données altérées vers les colonnes qui la référencent afin de conserver la cohérence et l'intégrité. Cette substitution se ferait en ajoutant les instructions adéquates à la place des instructions de refus de la ligne de commande. Le tronc de base de la librairie *(module de connexion et procédure de sélection dans la base de données)* resterait le même.

Enfin, vu que la portabilité et environnement d'utilisation de la solution sont les plus larges possible, grâce notamment à la technologie ODBC employée, il est envisageable d'implémenter ce module de gestion de clés étrangères sur tout autre système de gestion de bases de données n'offrant pas la possibilité de gérer les clés étrangères de manière explicite.

Bibliographie

[Aquilina 2003] Aquilina Jean-Michel, *Aide mémoire MySQL*, éditions OEM/Eyrolles, Paris, France, 2003.

[Barbier 2002] Barbier Pascal, *Gestion des tables DBMS par lien ODBC*, http://www.ensg.ign.fr/Formation/Formation_Continue/Formation_interne/Supports_de_cours/PDF/Sig_Pdf/MapInfo_6.5_livret_3.pdf, Institut National des Sciences Géographiques, Marne la Vallée, France, 9 août 2002.

[Ceri, Cochrane, Widom 2000] Stefano Ceri, Roberta J. Cochrane, Jennifer Widom, *Practical Applications of Constraints and Triggers : Successes and Lingering Issues,* http://www.cs.brown.edu/courses/cs227/papers/CCW00-Practical.pdf, 12 juin 2000.

[CharonWare 2003] CharonWare, *CASE Studio 2 : basic information,* http://www.casestudio.com/download/cs2_info.pdf, République tchèque, juillet 2003.

[Datanamic 2003] Datanamic, *site de référence concernant le logiciel DeZign for Databases (Atelier de génie logiciel),* http://www.datanamic.com/dezign/index.html, Pays Bas.

[DB-MAIN 2003] DB-MAIN Team, *The Database Engineering CASE environment,* http://www.info.fundp.ac.be/~dbm/publication/2002/DB-MAIN-Reference-Manual.pdf, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgique, 25 mars 2002.

[Dubois 2000] Dubois Paul, *MySQL*, éditions CampusPress, Paris, France, 2000.

[Englebert 1999] Englebert Vincent, *Voyager 2 : Atelier de Génie Logiciel & Méta-Modélisation*, http://www.info.fundp.ac.be/~ven/, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgique, 1999.

[ENST 2000] ENST DB Team, *Didacticiel d'accès aux Bases de Données*, http://www.bd.enst.fr/~dombd/Cours/Applications/odbc/, 27 décembre 2000.

[Estier, Pigneur 2000] Thibault Estier, Yves Pigneur, *Bases de données, Règles d'intégrité, contraintes d'intégrité*, Ecole des HEC, Université de Lausanne, Suisse, 2000.

[Hainaut 2001] Hainaut Jean-Luc, *Ingénierie des bases de données (troisième édition)*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgique, 2001.

[InnoDB 2003] InnoDB, *Site de référence concernant le moteur de stockage de tables InnoDB,* http://www.innodb.com, Finlande.

[JETA Software 2003] JETA Software, *Site de référence concernant le logiciel « abeille » (logiciel graphique de gestion de bases de données avec la fonctionnalité de gestion des clés étrangères),* http://www.jetaware.com/product_feat.jsp, USA.

[Laplanche 2002] Ganaël Laplanche, *Développement d'une DLL en C++ pour visual Basic*, http://www.laboratoire-microsoft.org/articles/dev/dll/, 3 octobre 2002.

[Leblanc 2003] Hervé Leblanc, *Les systèmes de gestion de bases de données,* http://www.irit.fr/~Herve.Leblanc/coursBDD.pdf, 16 janvier 2003.

[Mattou 2003] Mohammed-Lofti Mattou, *Aperçu des SGBD Open Source,* http://www.digitalisconsulting.com/doc/tech/OpenSource-SGBDR-6-2003.pdf, Juin 2003.

[MySQLCC 2003] MySQL, *Site de référence concernant le logiciel « MySQL Control Center ».* http://www.mysql.com/products/mysqlcc/.

[Pillou 2003] Jean-François Pillou, *Introduction à la technologie ODBC,* http://www.commentcamarche.net/odbc/odbcintro.php3, 2003.

[Sing 2002] Sing Joel, *Open Source Databases System – An introduction,* http://www.ionix.com.au/articles/osdb-intro.pdf, 2002.

[Webyog 2003] Webyog, *Site de référence concernant le logiciel « SQLyog » (logiciel graphique de gestion de bases de données avec la fonctionnalité de gestion des clés étrangères),* http://www.webyog.com.

ANNEXES

Annexe 1 : Le script Voyager relatif au générateur de code DDL de MySQL

Cette section reprend le script relatif au générateur de code DDL de MySQL. Comme expliqué dans le chapitre 5, l'appel de ce générateur se fait au départ du schéma physique mis en place dans l'atelier logiciel DB-MAIN. Le résultat produit par ce générateur est un fichier directement exploitable dans l'environnement de MySQL pour le déploiement de la base de données *(tables, colonnes, index, ...)*. Ce générateur est une extension de l'atelier logiciel DB-MAIN pour le SGBD MySQL. Le détail de l'architecture et du fonctionnement de ce programme figure dans la section 7.1.

```
/*****************************************
 ** Programme de génération de code DDL de MySQL
 ** Ecrit en Voyager 2
 *****************************************/

string: REMARK="--";
file: out;
schema: sch;

/*-------------------------------------------------------------------------
      Fonction de création d'un fichier ou va être écrit le script MySQL
      Retourne un entier (0 si la création s'est bien passée, 1 sinon)
      -------------------------------------------------------------------------*/
function integer OuvreFichier()
 string: namefile;
{ namefile:=BrowsePrint("Create a MySQL DDL file",
                "Files MySQL (*.mysql)|*.MYSQL|All (*.*)|*.*","MySQL");
 out:=OpenFile(namefile,_W);
 if IsVoid(out) then {
  print("INCORRECT FILE !\n");
  return 1;
 }
 else { return 0; }
}

/*-----------------------------------------
      Fonction donnant le type d'une colonne
      Retourne un string contenant le type de "si"
      -----------------------------------------*/
function string FindType(si_attribute: si)
        integer: l,d;
        data_object: dom;
        string : temp;
        string : temp2;
        list : liste_valeur;
        string : enum_set;
        string : type_enum_set;
```

```
        string : valeurs_enum_set;
        cursor : c;

{       switch (si.type) {
          case DATE_ATT:
               return "date";
          case CHAR_ATT:
              return StrConcat("char(",StrConcat(StrItos(si.length),")"));
          case VARCHAR_ATT:
               l:=si.length;
               if l=N_CARD then {
                    return "long varchar";
               } else {
                    return StrConcat("varchar(",StrConcat(StrItos(l),")"));
               };
          case NUM_ATT:
               l:=si.length;
               d:=si.decim;
               if d=0 then {
                              if l <= 10 then {
                                   return "int";}
                              else {
                                   return "bigint";};
               }
               else {
                    return StrConcat("decimal(",StrConcat(StrItos(l),StrConcat(",",
                      StrConcat(StrItos(d),")"))));
               };
          case FLOAT_ATT:
               return StrConcat("float(",StrConcat(StrItos(si.length),")"));
          case BOOL_ATT:
               return "char";
          case USER_ATT:
              dom := GetFirst(DATA_OBJECT[dom]{DOMAIN:[si]});
              temp := dom.name;

/* ----------------------------------------------------------------*/
/* GESTION des valeurs de ENUM ou SET */
               liste_valeur := si."Value constraint";
               attach c to liste_valeur;
               while IsNoVoid(c) do {

                         temp2 := StrConcat(temp2,StrConcat("'",StrConcat(get(c),"'")));
                         c>>;
                         if IsNoVoid(c) then {
                                   temp2 := StrConcat (temp2,", ");
                         };
               };
/* ---------------- */
```

```
            switch (temp){
                case "enum" :
                    return StrConcat("enum(",StrConcat(temp2,")"));
                case "set" :
                    return StrConcat("set(",StrConcat(temp2,")"));
                case "ENUM" :
                    return StrConcat("enum(",StrConcat(temp2,")"));
                case "SET" :
                    return StrConcat("set(",StrConcat(temp2,")"));
                case "Enum" :
                    return StrConcat("enum(",StrConcat(temp2,")"));
                case "Set" :
                    return StrConcat("set(",StrConcat(temp2,")"));};
        };
}


/*-----------------------------------------------
    Fonction donnant le nom d'un composant d'un groupe
    Retourne un string contenant le nom de "real"
    -----------------------------------------------*/
function string NameOf(real_component: real)
    attribute: att;
    role: ro;
    group: gr;
    integer: ttype;
{       ttype:=GetType(real);
        if ttype=ROLE then {
            ro:=real;
            return ro.name;
        };
        if (ttype=SI_ATTRIBUTE) or (ttype=CO_ATTRIBUTE) or
        (ttype=DO_ATTRIBUTE) then {
            att:=real;
            return att.name;
        };
        if ttype=GROUP then {
            gr:=real;
            return gr.name;
        };
        return "-- error --";
}


/*-----------------------------------------------
    Fonction donnant la liste des composants d'un groupe
    Retourne une liste contenant les composants de "gr"
    -----------------------------------------------*/
function list GetListOfComponents(group: gr)
    real_component: r;
```

```
    component: c;
{ return REAL_COMPONENT[r]{REAL_COMP:COMPONENT[c]{@GR_COMP:[gr]}};
}


/*-----------------------------------------------------
    Procédure générant la création d'une table
    Génère dans le fichier "out" le CREATE TABLE de "ent"
    -----------------------------------------------------*/
procedure GenererTable(entity_type: ent)
    owner_of_att: own;
    data_object: dto;
    attribute: att;
    si_attribute: si;
    group: gr;
    collection: col;
    coll_et: cet;
{
      printf(out,["create table ",ent.name," (\n"]);
      own:=ent;
      for att in ATTRIBUTE[att]{@OWNER_ATT:[own] with
        GetType(att)=SI_ATTRIBUTE} do {
        if GetType(att)=SI_ATTRIBUTE then {
            si:=att;
            printf(out,['\t',si.name," ",FindType(si)]);
            if si.min_rep then {
                printf(out," not null");
            };
            if IsNoVoid(_GetNext(OWNER_ATT,own,att)) then {
                printf(out,",\n");
            };
        } else {
            printf(out,['\t',att.name," -- ERROR\n"]);
        };
      };
      for gr in GROUP[gr]{@DATA_GR:[ent]} do {
        if gr.primary or gr.secondary then {
            if gr.primary then {
                printf(out,",\n\tprimary key (");
            } else {printf(out,",\n\tunique (");
            };
            PrintComponents(gr);
            printf(out,')');
        };
      };
      printf(out,')');
```

```
/* ajout du type de table si cela est spécifié pour l'entité
avec la valeur de type-table */

        if ent."type-table" <> "" then {
        printf(out,["\n\ttype=",ent."type-table"]);};

/* ---------- */

    printf(out,";\n\n");
}


/*-------------------------------------------------------
    Procédure générant la liste des composants d'un groupe
    Génère dans le fichier "out" la liste des composants de "gr"
    -------------------------------------------------------*/
procedure PrintComponents(group: gr)
    cursor: c;
    list: l;
    integer: ttype;
    attribute: at;
{
        l:=GetListOfComponents(gr);
        attach c to l;
        while IsNoVoid(c) do {
           ttype:=GetType(get(c));
           if (ttype=SI_ATTRIBUTE) or (ttype=DO_ATTRIBUTE) then {
                at:=get(c);
                printf(out,at.name);
           };
           if ttype=CO_ATTRIBUTE then {
                at:=get(c);
                printf(out,[at.name," -- atr decomp --"]);
           };
           c>>;
           if IsNoVoid(c) then {
                printf(out,", ");
           };
        };
}


/*-------------------------------------------------------
    Procédure générant une contrainte référentielle d'une table
    Génère dans le fichier "out" la FOREIGN KEY "gr" de "ent"
    -------------------------------------------------------*/
procedure GenererForeignKey(entity_type: ent,group: gr)
    member_cst: mcst,mcst2;
    entity_type: ent2,target_ent;
    real_component: real1, real2;
```

```
    constraint: cst;
    group: target;
    data_object: data;
    list: comp1,comp2;
    cursor: c,d;
{       for mcst in MEMBER_CST[mcst]{@GR_MEM:[gr]} do {
          if mcst.mem_role=OR_MEM_CST then {
                printf(out,["alter table ",ent.name," add constraint ",gr.name,
                                                "\n\tforeign key ("]);
                PrintComponents(gr);
                for target in
                  GROUP[target]{GR_MEM:MEMBER_CST[mcst2]{@CONST_MEM:
                  CONSTRAINT[cst]{CONST_MEM:[mcst]}} with target<>gr} do {
                     target_ent:=GetFirst(DATA_OBJECT[data]{DATA_GR:[target]});
                     if target.primary then {
                          printf(out,[")\n\treferences ",target_ent.name,";\n\n"]);
                     }
                     else {
                          printf(out,[")\n\treferences ",target_ent.name,'(']);
                          PrintComponents(target);
                          printf(out,");\n\n");
                     };
                     cst:=GetFirst(CONSTRAINT[cst]{CONST_MEM:[mcst]});
                     if cst.type=EQ_CONSTRAINT then {
                          printf(out,[REMARK,"alter table ",target_ent.name,
                               " add constraint ",
                               "\n",REMARK,"\t check(exist(select * from ",ent.name,
                               "\n",REMARK,"\t          where "]);
                               comp1:=GetListOfComponents(gr);
                               comp2:=GetListOfComponents(target);
                               attach c to comp1;
                               attach d to comp2;
                               while IsNoVoid(c) do {
                                        real1:=get(c);
                                        real2:=get(d);
                                        printf(out,[ent.name,".",NameOf(real1)," =",
                                        NameOf(real2)]);
                                        c>>;
                                        d>>;
                                        if IsNoVoid(c) then {
                                                printf(out," and ");
                                        } else {
                                                printf(out,"));\n\n");
                                        };
                               };
                     };
                };
          };
    };
}
```

```
/*-----------------------------------------------------------------
      Procédure générant un check pour un groupe coexistence, exclusive
      ou at-least-one
      Génère dans le fichier "out" le CHECK de "gr"
      -----------------------------------------------------------------*/
procedure GenererCheck(group: gr)
      entity_type: ent;
      real_component: real1, real2;
      list: comp;
      data_object: data;
      cursor: c, d;
{       ent:=GetFirst(DATA_OBJECT[data]{DATA_GR:[gr]});
        comp:=GetListOfComponents(gr);
        if gr.coexistence then {
           attach c to comp;
           if IsNoVoid(c) then {
                printf(out,["alter table ",ent.name," add constraint ",gr.name,
                                                    '\n',"\tcheck(("]);
           };
           while IsNoVoid(c) do {
                real1:=get(c);
                printf(out,[NameOf(real1)," is not null"]);
                c>>;
                if IsNoVoid(c) then {
                     printf(out," and ");
                } else {
                     printf(out,")\n");
                };
           };
           attach c to comp;
           if IsNoVoid(c) then {
                printf(out,["\t     or ("]);
           };
           while IsNoVoid(c) do {
                real1:=get(c);
                printf(out,[NameOf(real1)," is null"]);
                c>>;
                if IsNoVoid(c) then {
                     printf(out," and ");
                }
                else {
                     printf(out,"));\n\n");
                };
           };
      }
      else {
           if gr.exclusive and gr.atleastone then {
                attach c to comp;
                if IsNoVoid(c) then {
                     printf(out,["alter table ",ent.name," add constraint ",gr.name,
```

```
                                        '\n',"\tcheck(("]);
    };
    while IsNoVoid(c) do {
        real1:=get(c);
        attach d to comp;
        while IsNoVoid(d) do {
            real2:=get(d);
            if real1=real2 then {
                printf(out,[NameOf(real2)," is not null"]);
            }
            else {
                printf(out,[NameOf(real2)," is null"]);
            };
            d>>;
            if IsNoVoid(d) then {
                printf(out," and ");
            }
            else {
                printf(out,')');
            };
        };
        c>>;
        if IsNoVoid(c) then {
            printf(out,"\n\t     or (");
        }
        else {
            printf(out,");\n\n");
        };
    };
}
else {
    if gr.atleastone then {
        attach c to comp;
        if IsNoVoid(c) then {
            printf(out,["alter table ",ent.name," add constraint ",gr.name,
                                                '\n',"\tcheck("]);
        };
        while IsNoVoid(c) do {
            real1:=get(c);
            printf(out,[NameOf(real1)," is not null"]);
            c>>;
            if IsNoVoid(c) then {
                printf(out," or ");
            }
            else {
                printf(out,");\n\n");
            };
        };
    }
    else {
```

```
                    if gr.exclusive then {
                        attach c to comp;
                        if IsNoVoid(c) then {
                            printf(out,["alter table ",ent.name," add constraint ",gr.name,
                                                                '\n',"\tcheck(("]);
                        };
                        while IsNoVoid(c) do {
                            real1:=get(c);
                            attach d to comp;
                            while IsNoVoid(d) do {
                                real2:=get(d);
                                if real1=real2 then {
                                    printf(out,[NameOf(real2)," is not null"]);
                                }
                                else {
                                    printf(out,[NameOf(real2)," is null"]);
                                };
                                d>>;
                                if IsNoVoid(d) then {
                                    printf(out," and ");
                                }
                                else {
                                    printf(out,')');
                                };
                            };
                            c>>;
                            if IsNoVoid(c) then {
                                printf(out,"\n\t     or (");
                            }
                            else {
                                printf(out,"\n\t     or (");
                                attach d to comp;
                                while IsNoVoid(d) do {
                                    real2:=get(d);
                                    printf(out,[NameOf(real2)," is null"]);
                                    d>>;
                                    if IsNoVoid(d) then {
                                        printf(out," and ");
                                    }
                                    else {
                                        printf(out,"));\n\n");
                                    };
                                };
                            };
                        };
                    };
                };
            };
        };
    };
}
```

```
/*-------------------------------------------------
    Procédure générant les index d'une table
    Génère dans le fichier "out" les CREATE INDEX de "ent"
    -------------------------------------------------*/
procedure GenererIndex(entity_type: ent)
    group: gr;
    data_object: data;
{       data:=ent;
        for gr in GROUP[gr]{@DATA_GR:[data]} do {
            if not(gr.primary or gr.secondary) and gr.key then {
                printf(out,["create index ",gr.name,"\n\t on ",ent.name," ("]);
                PrintComponents(gr);
                printf(out,");\n\n");
            };
        };
}


/*-------------------------------------------------------------
    Procédure générant les contraintes d'une table
    Génère dans le fichier "out" les FOREIGN KEY et CHECK de "ent"
    -------------------------------------------------------------*/
procedure GenererContrainte(entity_type: ent)
    group: gr;
    data_object: data;
{
        data:=ent;
        for gr in GROUP[gr]{@DATA_GR:[data]} do {
            GenererForeignKey(ent,gr);
            GenererCheck(gr);
        };
}


/*-------------------------------------------------------------
    Procédure générant le script SQL STANDARD d'un schéma
    Génère dans le fichier "out" le script SQL STANDARD de "sch"
    -------------------------------------------------------------*/
procedure GenererMySql()
    collection: col;
    entity_type: ent;
    data_object: data;
{
        printf(out,[REMARK," ****************************\n",
                                REMARK," * génération de code MySQL *\n",
                                REMARK," ****************************\n\n\n",
                                REMARK," Database Section\n",
                                REMARK," _________________\n\n"
                            ]);
```

```
printf(out,["create database ",sch.name,";\n\n\n"]);
printf(out,["use ",sch.name,";\n\n\n"]);

printf(out,[REMARK," Pas de DBSpace Section\n",
                    REMARK," _____\n\n"
                ]);

printf(out,[REMARK," Table Section\n",
                    REMARK," _____\n\n"
                ]);

print(["Write CREATE TABLE\n"]);
for data in DATA_OBJECT[data]{@SCH_DATA:[sch] with
GetType(data)=ENTITY_TYPE } do
{
   ent:=data;
   GenererTable(ent);
};

printf(out,['\n',REMARK," Constraints Section\n" ,
REMARK," _____\n",
REMARK," !!Génération des contraintes,\n",REMARK," Dans MySQL, les foreign
keys et les checks\n",
REMARK," seront DECLARES mais PAS GERES !!",
            '\n',REMARK," _____\n\n"]);
print(["\nWrite ALTER TABLE\n"]);
print("\n!!Génération des contraintes,\nDans MySQL, les foreign keys et les
checks\nseront DECLARES mais PAS GERES !!\n\n");
for data in DATA_OBJECT[data]{@SCH_DATA:[sch] with
GetType(data)=ENTITY_TYPE } do
{
   ent:=data;
   GenererContrainte(ent);
};
printf(out,['\n',REMARK," Index Section",
                    '\n',REMARK," _____\n\n"]);
print(["Write CREATE INDEX\n"]);
for data in DATA_OBJECT[data]{@SCH_DATA:[sch] with
GetType(data)=ENTITY_TYPE } do
{
   ent:=data;
   GenererIndex(ent);
};
}
```

```
/*------------------
    Programme principal
    ------------------*/
begin
    if not(OuvreFichier())  then{
        sch:=GetCurrentSchema();
        if not(IsVoid(sch)) then {
            print("Génération de MySQL\n\n");
            SetPrintList("","","");
            GenererMySql();
            CloseFile(out);
            print("\nOK!\n");
        };
    };

end
```

Annexe 2 : Le script Voyager relatif au générateur du fichier contenant la représentation des clés étrangères

Les lignes de code reprises dans cette section sont celles du générateur du fichier représentatif des clés étrangères d'un schéma physique d'une base de données mise en place dans l'atelier logiciel DB-MAIN. Le fichier produit en résultat est exploité par la librairie de fonctions relatives à la gestion des clés étrangères. Dans le fichier produit par ce générateur, les informations sont représentées de manière structurée. Le format des clés étrangères, la description et l'architecture de ce module sont repris dans la section 7.2 de ce mémoire.

```
/*****************************************
 ** Programme de génération du fichier H
 ** Ecrit en Voyager 2
 *****************************************/


string: REMARK="--";

file: out;
schema: sch;
integer : n;

/*-------------------------------------------------------------------------
      Fonction de création d'un fichier dans lequel va être écrit le script décrivant
        la structure de la base de données (fichier header file (.h))
      Retourne un entier (0 si la création s'est bien passée, 1 sinon)
      -------------------------------------------------------------------------*/
function integer OuvreFichier()
 string: namefile;
{ namefile:="structbd.h";
 out:=OpenFile(namefile,_W);
 if IsVoid(out) then {
   print("INCORRECT FILE !\n");
   return 1;
 }
 else { return 0; }
}

/*-------------------------------------------------
      Fonction donnant la liste des composants d'un groupe
      Retourne une liste contenant les composants de "gr"
      -------------------------------------------------*/
function list GetListOfComponents(group: gr)
      real_component: r;
      component: c;
{ return REAL_COMPONENT[r]{REAL_COMP:COMPONENT[c]{@GR_COMP:[gr]}};
}
```

```
/* ------------------------------ */
procedure PrintComponents(group: gr)
      cursor: c;
      list: l;
      integer: ttype;
      attribute: at;
{       l:=GetListOfComponents(gr);
        attach c to l;
        while IsNoVoid(c) do {
           ttype:=GetType(get(c));
           if (ttype=SI_ATTRIBUTE) or (ttype=DO_ATTRIBUTE) then {
                at:=get(c);
                printf(out,at.name);
           };
           if ttype=CO_ATTRIBUTE then {
                at:=get(c);
                printf(out,[at.name," -- atr decomp --"]);
           };
           c>>;
           if IsNoVoid(c) then {
                printf(out,",");
           };
        };
}


/*-------------------------------------------------
      Procédures écrivant dans le fichier structbd.h la liste des clés de type TO et FROM
      -------------------------------------------------*/

procedure PrintComponentsofkeyTO(group: gr, entity_type: ent, group: target, entity_type:
target_ent)
      integer: ttype;
      attribute: at,targetat;
{
        printf(out,["strcat (foreignkeyTO.liste_foreignkeyTO", ',',"",  ent.name,"@"]);
        PrintComponents(gr);
        printf(out,["#", target_ent.name, "@"]);
        PrintComponents(target);
        printf(out,["#;",'\',"0","","");\n"]);

}

procedure PrintComponentsofkeyFROM(group: gr, entity_type: ent, group: target,
entity_type: target_ent)
      integer: ttype;
      attribute: at,targetat;
{
        printf(out,["strcat (foreignkeyFROM.liste_foreignkeyFROM", ',',"",
                    target_ent.name,"@"]);
```

```
        PrintComponents(target);
        printf(out,["#", ent.name, "@"]);
        PrintComponents(gr);
        printf(out,["#;",'\',"0","","");\n"]);
}


/*-----------------------------------------------------------
      Procédures générant une contrainte référentielle d'une table
      -----------------------------------------------------------*/

procedure GenererForeignKeyTO_Groupe_Table(entity_type: ent,group: gr)
      member_cst: mcst,mcst2;
      entity_type: ent2,target_ent;
      real_component: real1, real2;
      constraint: cst;
      group: target;
      data_object: data;
      list: comp1,comp2;
      cursor: c,d;
{ for mcst in MEMBER_CST[mcst]{@GR_MEM:[gr]} do {
            if mcst.mem_role=OR_MEM_CST then {

               for target in
GROUP[target]{GR_MEM:MEMBER_CST[mcst2]{@CONST_MEM:CONSTRAINT[cst]{
CONST_MEM:[mcst]}} with target<>gr} do {
                        target_ent:=GetFirst(DATA_OBJECT[data]{DATA_GR:[target]});
                        PrintComponentsofkeyTO(gr, ent, target, target_ent);
                        };

            };
      };
}


procedure GenererForeignKeyFROM_Groupe_Table(entity_type: ent,group: gr)
      member_cst: mcst,mcst2;
      entity_type: ent2,target_ent;
      real_component: real1, real2;
      constraint: cst;
      group: target;
      data_object: data;
      list: comp1,comp2;
      cursor: c,d;
{ for mcst in MEMBER_CST[mcst]{@GR_MEM:[gr]} do {
            if mcst.mem_role=OR_MEM_CST then {

               for target in
GROUP[target]{GR_MEM:MEMBER_CST[mcst2]{@CONST_MEM:CONSTRAINT[cst]{
CONST_MEM:[mcst]}} with target<>gr} do {
```

```
                    target_ent:=GetFirst(DATA_OBJECT[data]{DATA_GR:[target]});
                    PrintComponentsofkeyFROM(gr, ent, target, target_ent);
                    };

        };
    };
}


/*-------------------------------------------------------
    -------------------------------------------------------*/

procedure GenererForeignKeyTO_Table(entity_type: ent)
    group: gr;
    data_object: data;
    integer : n;

{       data:=ent;
        n:=0;
        for gr in GROUP[gr]{@DATA_GR:[data]} do {
          n:=n+1;
          GenererForeignKeyTO_Groupe_Table(ent,gr);
        };
}


procedure GenererForeignKeyFROM_Table(entity_type: ent)
    group: gr;
    data_object: data;
    integer : n;

{       data:=ent;
        n:=0;
        for gr in GROUP[gr]{@DATA_GR:[data]} do {
          n:=n+1;
          GenererForeignKeyFROM_Groupe_Table(ent,gr);
        };
}


/*-------------------------------------------------------
    Procédure générant l'entête du fichier H
    définition des structures foreignkeyTO et foreignkeyFROM
    -------------------------------------------------------*/
procedure GenererENTETE()
{
        printf(out,"struct \n");
        printf(out,"{char liste_foreignkeyTO[20000] ; \n");
        printf(out,"} foreignkeyTO ; \n");
        printf(out,"struct \n");
```

```
        printf(out,"{char liste_foreignkeyFROM[20000] ; \n");
        printf(out,"} foreignkeyFROM ; \n\n");
}


/*-------------------------------------------------------
      Procédures générant la structure des clés étrangères de type TO
        et de type FROM
      -------------------------------------------------------*/
procedure GenererForeignKeyTO()
      collection: col;
      entity_type: ent;
      data_object: data;
{
        print(["Write key.liste_key \n"]);

        for data in DATA_OBJECT[data]{@SCH_DATA:[sch] with
        GetType(data)=ENTITY_TYPE } do {
                ent:=data;
                GenererForeignKeyTO_Table(ent);
        };
}

procedure GenererForeignKeyFROM()
      collection: col;
      entity_type: ent;
      data_object: data;
{
        print(["Write key.liste_key \n"]);

        for data in DATA_OBJECT[data]{@SCH_DATA:[sch] with
        GetType(data)=ENTITY_TYPE } do {
                ent:=data;
                GenererForeignKeyFROM_Table(ent);
        };

        printf(out,"} \n");
}
```

```
/*------------------
    Programme principal
    ------------------*/
begin
    if not(OuvreFichier())  then{
        sch:=GetCurrentSchema();
        if not(IsVoid(sch)) then {
            print("Génération de MySQL\n\n");
            SetPrintList("","","");
            GenererENTETE();

            printf(out,"\n");
            printf(out,"void generer_key() \n");
            printf(out,"{ \n");

            printf(out,["strcpy (foreignkeyTO.liste_foreignkeyTO,","",'\',"0","","");\n"]);

            GenererForeignKeyTO();

            printf(out,["strcpy
                (foreignkeyFROM.liste_foreignkeyFROM,","",'\',"0","","");\n"]);

            GenererForeignKeyFROM();

            CloseFile(out);

            print("\nOK!\n");

        };
    };

end
```

Annexe 3 : Le script de la librairie de fonctions de gestion des clés étrangères

Cette partie contient les lignes de code de la librairie de fonctions permettant la gestion des clés étrangères. Comme expliqué dans le chapitre 6, ce module consulte un fichier dans lequel se trouvent les informations concernant les clés étrangères d'une base de données et utilise la technologie ODBC pour interagir avec une base de données MySQL. L'objectif de cette librairie de fonctions est de valider une instruction d'interaction en utilisant la stratégie « No Action ». Les trois types d'interactions possibles sont l'insertion, la suppression et la modification de données. Les détails, les diagrammes d'actions et l'architecture de cette librairie de fonctions figurent dans la section 7.3 de ce mémoire.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <windows.h>
#include <sql.h>
#include <structbd.h>  /* structbd.h : fichier de définition des clés étrangères */

/* ligne de commande */
/* champ de gestion de la ligne de commande */
char* ligne_commande ="";
char ligne_commande_copy[500];
char table_commande[64];
char* table_param = "";

/* condition_param = conditions exprimées pour le where */
char* condition_param ="";

/* colonnes_insert = (col1, col2, ..)  values ( valeur1, valeur2, ..) */
char* colonnes_insert ="";

/* colonnes_update = col1=valeur1, colo2=valeur2 */
char* colonnes_update ="";

/* adresses pour découpage de la ligne de commande */
char* adr1 = "" ;
char* adr2 = "";
char* adr3 ="";

/* champ de 6 positions pour déterminer le type d'instruction
INSERT OU DELETE OU UPDATE */
char type_trt[6];

/* type de traitement :
1 = DELETE
2 = INSERT
3 = UPDATE */
```

```
int type_traitement;


/* colonnes de travail */
char* colonnes_insert_wo ="";
char valeur_colonnes_insert_wo[200];
char* colonnes_update_wo ="";
char valeur_colonnes_update_wo[200];

int taille_param;


char *token ="";
char *tokenwo ="";
char *token2 ="";
char *table ="";
char *colonnes ="";
char *colonnes_wo ="";
char *adrwo ="";
char *tablecible ="";
char *colonnestable ="";
char *colonnescible ="";

char *adr_and ="";
char *adr_and_between ="";
char *adr_or ="";


char condition_param_tmp[900];
char condition_param_tmp2[900];
char condition_param_edt[900];

char ligneselect[10000];

char retourparam[10000];

char ligneselectjointure[1000];

/* */
char table_origine[64];
char colonnes_origine[10000];
char table_cible[64];
char colonnes_cible[10000];

struct {
  char colonne_origine[64];
  char colonne_cible[64];
 } tab_colonnes[15];
```

```
struct {
  char colonne_insert[64];
  char valeur_insert[256];
 } tab_colonnes_insert[15];

struct {
  char colonne_update[64];
  char valeur_update[256];
 } tab_colonnes_update[15];

char *ligne ="";

static char seps[] = ";";
static char seps3[] = ",";
static char seps2[] = ",()";

int i,j,k,l,m,n,o,p,q, nb_col, nb_col_insert, nb_col_insert_select, nb_col_update ;

/* nb_col_insert_select sert à compter le nbr de colonnes réellement insérées dans la ligne
de commande si nb_col_insert_select est < nb_col, on n'insère pas ! */

/* utile pour choix de type de clé dans le trt de préparation de ligne de update */
int update_to_from;

int number;
int total;
int position[50];
int indice;
int iteration;
int length;

int ctrl_retour;

// Variable permettant de contrôler si l'instruction
// de la ligne de commande est acceptée ou non ..
// !!! valeur 0 : ACCEPTE !!!! valeur 1 : REFUSE !!!!

bool ligne_commande_ok_ko;

// ***********************************************
// Variables connexion ODBC
// ***********************************************
HENV hEnv = NULL;
HDBC hDBC = NULL;
HSTMT hStmt = NULL;
UCHAR szDSN[SQL_MAX_DSN_LENGTH];
UCHAR* szUID = NULL;
UCHAR* szPasswd = NULL;
UCHAR szModel[128];
SDWORD cbModel;
```

```
  RETCODE retcode;
  char requete[500];
  SQLCHAR res_count[80]  ;
  SQLINTEGER cbres_count;

// ***************************************************

// **********************
// Module de connexion
// **********************

extern "C" __declspec(dllexport) void definir_nom_lien_odbc (char* nom_lien){
  // nom du Lien ODBC
        strcpy(szDSN,nom_lien);
}

void connexion(){
        SQLAllocEnv(&hEnv);
        SQLAllocConnect(hEnv,&hDBC);
        retcode =
            SQLConnect(hDBC,szDSN,SQL_NTS,szUID,SQL_NTS,szPasswd,SQL_NTS);
}

void requete_select(char* requete){

 // préparation et select + recherche count
    retcode = SQLAllocStmt(hDBC,&hStmt);

    retcode = SQLExecDirect(hStmt,(SQLCHAR*)requete, SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    {
        SQLBindCol (hStmt, 1, SQL_C_CHAR, res_count, 80, &cbres_count);

        retcode = SQLFetch (hStmt);

    }
    else
    {
        printf("\n erreur dans l'instruction encodee");
    }
}

void requete_BD (char* requete){
  if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
        {

        // préparation et select + recherche count
                retcode = SQLAllocStmt(hDBC,&hStmt);

                retcode = SQLExecDirect(hStmt,(SQLCHAR*)requete, SQL_NTS);
```

```
                if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
                {
                        printf("\n OK");
                        ctrl_retour=1;
                }
                else
                {
                        printf("\n ERREUR " );
                        ctrl_retour=2;
                }
        }
}

void deconnexion(){
        SQLFreeStmt (hStmt,SQL_DROP);
        SQLDisconnect (hDBC);
        SQLFreeConnect (hDBC);
        SQLFreeEnv (hEnv);
}


// procédure qui envoie la requête vers la base de données
// lorsque c'est accepté.
void traitement_BD(){
  if (ligne_commande_ok_ko)
  {
        printf("\n\n ==> instruction acceptee\n");
        connexion();
        requete_BD(ligne_commande_copy);
  }
  else
  {
        printf("\n\n ==> instruction refusee\n");
        ctrl_retour=3;
  }
}


/* ----------------FORMATAGE COLONNES : CAS INSERT------------------- */

void formatage_colonnes_insert ()
{
        k=0;
        while (k<15)
         {
                strcpy(tab_colonnes_insert[k].colonne_insert," ");
                l=0;
               while (l<64)
                {
                        strcat(tab_colonnes_insert[k].colonne_insert," ");
```

```
                l++;
        }
        strcpy(tab_colonnes_insert[k].colonne_insert,"\0");
        k++;
}

k=0;
while (k<15)
{
        strcpy(tab_colonnes_insert[k].valeur_insert," ");
        l=0;
        while (l<256)
        {
                strcat(tab_colonnes_insert[k].valeur_insert," ");
                l++;
        }
        strcpy(tab_colonnes_insert[k].valeur_insert,"\0");
        k++;
}

adrwo = strstr(colonnes_insert," VALUES ");
colonnes_insert_wo= colonnes_insert;
strcpy(valeur_colonnes_insert_wo,adrwo + 8);
token = strtok( colonnes_insert_wo, seps2 );
nb_col_insert=0;
n=0;
while( token < adrwo - 2 )
{
        tokenwo=token;
        strcpy(tab_colonnes_insert[n].colonne_insert, tokenwo);
        token = strtok( NULL, seps2 );
        n++;
        if (token < adrwo - 2)
                nb_col_insert++;
}

token = strtok( valeur_colonnes_insert_wo, seps2 );
n=0;
while( token !=NULL )
{
        tokenwo=token;
        strcpy(tab_colonnes_insert[n].valeur_insert, tokenwo);
        token = strtok( NULL, seps2 );
        n++;
}
}
```

```
/* ----------FORMATAGE CONDITION : CAS DELETE ET UPDATE ---------- */

void formatage_condition_delete_update ()
{  /* d'abord trt sur le AND */

        while (adr_and = strstr(condition_param," AND "))
        {       length = adr_and - condition_param ;
                strcat(condition_param_tmp,table_param);
                strcat(condition_param_tmp,".");
                strncat(condition_param_tmp,condition_param, length);
                strcat(condition_param_tmp," AND ");
                condition_param=adr_and + 5;
                adr_and = strstr(condition_param," AND ");
                m++;
        }
        length = adr_and - condition_param ;
        strcat(condition_param_tmp,table_param);
        strcat(condition_param_tmp,".");
        strncat(condition_param_tmp,condition_param, length);
        condition_param=condition_param_tmp;

 // Traitement sur le BETWEEN => consiste à enlever le string "nomtable."
 // placé devant la deuxième valeur (après le AND, à cause du premier trt (AND))

        while (adr_and_between = strstr(condition_param," BETWEEN "))
        {
                length = adr_and_between - condition_param + 9 ;
                strncat(condition_param_tmp2,condition_param, length);
                condition_param=adr_and_between + 9;
                adr_and = strstr(condition_param," AND ");
                length = adr_and - condition_param + 5 ;
                strncat(condition_param_tmp2,condition_param, length);
                condition_param=adr_and + 5;
                adr_and = strstr(condition_param,".");
                condition_param=adr_and + 1;
                adr_and_between = strstr(condition_param," BETWEEN ");
        }
        length = adr_and_between - condition_param ;
        strncat(condition_param_tmp2,condition_param, length);
        condition_param=condition_param_tmp2;

  /* ensuite trt sur le OR */
        n=0;
        while (adr_or = strstr(condition_param," OR "))
        {
                length = adr_or - condition_param ;
                if (n>0)
                {
                        strcat(condition_param_edt,table_param);
                        strcat(condition_param_edt,".");
```

```
                }
                strncat(condition_param_edt,condition_param, length);
                strcat(condition_param_edt," OR ");
                condition_param=adr_or + 4;
                adr_or = strstr(condition_param," OR ");
                n++;
        }

        length = adr_or - condition_param ;
        if (n>0)
        {
                strcat(condition_param_edt,table_param);
                strcat(condition_param_edt,".");
        }
        strncat(condition_param_edt,condition_param, length);
  }

/* ----------------FORMATAGE COLONNES : CAS UPDATE---------------------- */

void formatage_colonnes_update ()
{
        k=0;
        while (k<15)
        {
                strcpy(tab_colonnes_update[k].colonne_update," ");
                l=0;
                while (l<64)
                {
                        strcat(tab_colonnes_update[k].colonne_update," ");
                        l++;
                }
                strcpy(tab_colonnes_update[k].colonne_update,"\0");
                k++;
        }

        k=0;
        while (k<15)
        {
                strcpy(tab_colonnes_update[k].valeur_update," ");
                l=0;
                while (l<256)
                {
                        strcat(tab_colonnes_update[k].valeur_update," ");
                        l++;
                }
                strcpy(tab_colonnes_update[k].valeur_update,"\0");
                k++;
        }

        adrwo = strstr(colonnes_update," WHERE ");
```

```
        if (adrwo != NULL)
        {
                length = adrwo - colonnes_update;
                strncpy(colonnes_update_wo,colonnes_update,length);
                colonnes_update_wo[length]='\0';
                strcat(colonnes_update_wo, "\0");
        }
        else
        {
                strcpy(colonnes_update_wo,colonnes_update);
                strcat(colonnes_update_wo, "\0");
        }

        token = strtok( colonnes_update_wo, seps3 );
        nb_col_update=0;
        while(token != NULL)
        {
                tokenwo=token;
                adrwo = strchr(tokenwo, '=');
                if (adrwo != NULL)
                {
                        length = adrwo - tokenwo;
                        strncpy(tab_colonnes_update[nb_col_update].colonne_update,
                        tokenwo,length);
                        tab_colonnes_update[nb_col_update].colonne_update[length]='\0';
                        strcpy(tab_colonnes_update[nb_col_update].valeur_update, adrwo + 1);
                        strcat(tab_colonnes_update[nb_col_update].valeur_update,"\0");
                        token = strtok( NULL, seps3 );
                }
                else
                {
                        strcat(tab_colonnes_update[nb_col_update].valeur_update, tokenwo);
                        strcat(tab_colonnes_update[nb_col_update].valeur_update,"\0");
                        token = strtok( NULL, seps3 );
                }
                if (token !=NULL)
                        nb_col_update ++;
        }
}


/* ------------------------ RECHERCHE DES CLES de type FROM ----------*/

char* rechercher_cle_FROM ()
/* recherche des clés appartenant à cette table dans la liste complète des clés*/
{
        strcpy(retourparam,"");
        generer_key();
```

```
taille_param = strlen(table_param);

tokenwo = "";
tablecible = "";
colonnescible = "";
colonnestable = "";
number = 0;
total = 0;
indice = 0;

token = strtok( foreignkeyFROM.liste_foreignkeyFROM, seps );

while( token != NULL )
{
        token2= token;
        if (strncmp(strupr(token2),table_param,taille_param)==0)
        {
                position[number]=indice;
                number++;
        }
        total++;
        indice++;
        token = strtok( NULL, seps );
}

/* si 0 occurences .. ok */

/* si une ou plusieurs occurrences, parsing de l'occurrence en fct de la position et du nombre*/

        iteration= 0;
        i = 0;
        generer_key();
        token = strtok( foreignkeyFROM.liste_foreignkeyFROM, seps );
        while (i < number)
        {       while (iteration < position[i])
                {/*printf("\n\t%d, %d, %d", iteration, position[i], i);*/
                        token = strtok( NULL, seps );
                        iteration++;
                }

                /*printf("\n\t%d, %d, %d", iteration, position[i], i);*/

                /*      printf ("\n%s ", token);*/
                token2 = token;

                if (i  == 0)
                        {strcpy(retourparam,token2);}
                else
                        {strcat(retourparam,token2);}
```

```
                strcat(retourparam,";");

                token = strtok( NULL, seps );
                iteration++;
                i++;
        }

        strcat(retourparam,"\0");
        return retourparam;
}


/* ---------------------- RECHERCHE DES CLES de type TO ----------*/

char* rechercher_cle_TO ()
/* recherche des clés appartenant à cette table dans la liste complète des clés*/
{
        strcpy(retourparam,"");

        generer_key();

        taille_param = strlen(table_param);

        tokenwo = "";
        tablecible = "";
        colonnescible = "";
        colonnestable = "";
        number = 0;
        total = 0;
        indice = 0;

        token = strtok( foreignkeyTO.liste_foreignkeyTO, seps );
        while( token != NULL )
        {
                token2= token;
                if (strncmp(strupr(token2),table_param,taille_param)==0)
                {
                        position[number]=indice;
                        number++;
                }
                total++;
                indice++;
                token = strtok( NULL, seps );
        }


/* si 0 occurences .. ok */

/* si une ou plusieurs occurrences, parsing de l'occurrence en fct de la position et du nombre*/
```

```
	iteration= 0;
	i = 0;

	generer_key();

	token = strtok( foreignkeyTO.liste_foreignkeyTO, seps );

	while (i < number)
	{	while (iteration < position[i])
		{/*printf("\n\t%d, %d, %d", iteration, position[i], i);*/
			token = strtok( NULL, seps );
			iteration++;
		}

		/*printf("\n\t%d, %d, %d", iteration, position[i], i);*/
		 /*	printf ("\n%s ", token);*/
		token2 = token;

		if (i == 0)
		{
			strcpy(retourparam,token2);
		}
		else
		{
			strcat(retourparam,token2);
		}

		strcat(retourparam,";");

		token = strtok( NULL, seps );
		iteration++;
		i++;
	}

	strcat(retourparam,"\0");
	return retourparam;
}


/* réinitialisation de table origine, table cible et leurs colonnes */

void reinit(){
	j=0;
	strcpy(table_origine," ");
	while (j<63)
	{
		strcat(table_origine," ");
		j++;
	}
	strcpy(table_origine,"\0");
```

```
        j=0;
        strcpy(colonnes_origine," ");
        while (j<9999)
        {
                strcat(colonnes_origine," ");
                j++;
        }
        strcpy(colonnes_origine,"\0");
        j=0;
        strcpy(table_cible," ");
        while (j<63)
        {
                strcat(table_cible," ");
                j++;
        }
        strcpy(table_cible,"\0");
        j=0;
        strcpy(colonnes_cible," ");
        while (j<9999)
        {
                strcat(colonnes_cible," ");
                j++;
        }
        strcpy(colonnes_cible,"\0");
}

void reinit_colonnes ()
{
        k=0;
        while (k<15)
        {
                strcpy(tab_colonnes[k].colonne_origine," ");
                strcpy(tab_colonnes[k].colonne_cible," ");
                l=0;
                while (l<63)
                        {
                                strcat(tab_colonnes[k].colonne_origine," ");
                                strcat(tab_colonnes[k].colonne_cible," ");
                                l++;
                        }
                strcpy(tab_colonnes[k].colonne_origine,"\0");
                strcpy(tab_colonnes[k].colonne_cible,"\0");
                k++;
        }
}
```

```
void remplir_struct_colonne ()
{
 /*colonnes origine */
        nb_col = 0;
        colonnes=colonnes_origine;
        colonnes_wo = strchr(colonnes,',');
        while (colonnes_wo != NULL)
        {
                length = colonnes_wo - colonnes;
                colonnes_wo = colonnes_wo - length;
                strncpy(tab_colonnes[nb_col].colonne_origine,colonnes_wo,length);
                colonnes_wo = colonnes_wo + length + 1;
                colonnes = colonnes + length + 1;
                colonnes_wo = strchr(colonnes,',');
                tab_colonnes[nb_col].colonne_origine[length]='\0';
                nb_col ++;
        }
        colonnes_wo=colonnes;
        colonnes_wo = strchr(colonnes,'\0');
        length = colonnes_wo - colonnes;
        colonnes_wo = colonnes_wo - length;
        strncpy(tab_colonnes[nb_col].colonne_origine,colonnes_wo,length);
        tab_colonnes[nb_col].colonne_origine[length]='\0';

  /* colonnes cible*/
        nb_col = 0;
        colonnes=colonnes_cible;
        colonnes_wo = strchr(colonnes,',');
        while (colonnes_wo != NULL)
        {
                length = colonnes_wo - colonnes;
                colonnes_wo = colonnes_wo - length;
                strncpy(tab_colonnes[nb_col].colonne_cible,colonnes_wo,length);
                colonnes_wo = colonnes_wo + length + 1;
                colonnes = colonnes + length + 1;
                colonnes_wo = strchr(colonnes,',');
                tab_colonnes[nb_col].colonne_cible[length]='\0';
                nb_col ++;
        }
        colonnes_wo=colonnes;
        colonnes_wo = strchr(colonnes,'\0');
        length = colonnes_wo - colonnes;
        colonnes_wo = colonnes_wo - length;
        strncpy(tab_colonnes[nb_col].colonne_cible,colonnes_wo,length);
        tab_colonnes[nb_col].colonne_cible[length]='\0';
}
```

```
/* préparation des selects qui permettront de vérifier la validité de la ligne de
commande insérée */
void verifier_integrite ()
{

        token = strtok( ligne, seps );
        i=0;
        while (i < number)
        {
                token = strupr(token);
                printf("\n%s ", token);
                adrwo=strchr (token,'@');
                length = adrwo - token;
                strncpy(table_origine,token,length);
                table_origine[length]='\0';

                token=token + length + 1;
                adrwo=strchr (token,'#');
                length = adrwo - token;
                strncpy(colonnes_origine, token,length);
                colonnes_origine[length]='\0';

                token=token + length + 1;
                adrwo=strchr (token,'@');
                length = adrwo - token;
                strncpy(table_cible,token,length);
                table_cible[length]='\0';

                token=token + length + 1;
                adrwo=strchr (token,'#');
                length = adrwo - token;
                strncpy(colonnes_cible, token,length);
                colonnes_cible[length]='\0';

                reinit_colonnes();
                remplir_struct_colonne();

    /*-----------------------------------------------------------------*/
    /*
    Création du select type delete
    */
    if (type_traitement == 1)
    {
        if (nb_col == 0)
        {
                strcpy(ligneselectjointure, "SELECT COUNT(DISTINCT ");
                strcat(ligneselectjointure, table_origine);
                strcat(ligneselectjointure, ".");
                strcat(ligneselectjointure, tab_colonnes[nb_col].colonne_origine);
                strcat(ligneselectjointure, ") FROM ");
```

```
        strcat(ligneselectjointure, table_origine);
        strcat(ligneselectjointure, ", ");
        strcat(ligneselectjointure, table_cible);
        strcat(ligneselectjointure, " WHERE ");
        strcat(ligneselectjointure, table_origine);
        strcat(ligneselectjointure, ".");
        strcat(ligneselectjointure, tab_colonnes[nb_col].colonne_origine);
        strcat(ligneselectjointure, "=");
        strcat(ligneselectjointure, table_cible);
        strcat(ligneselectjointure, ".");
        strcat(ligneselectjointure, tab_colonnes[nb_col].colonne_cible);
}
else
{
        strcpy(ligneselectjointure, "SELECT COUNT(DISTINCT ");
        m=0;
        while (m<=nb_col)
        {
                strcat(ligneselectjointure, table_origine);
                strcat(ligneselectjointure, ".");
                strcat(ligneselectjointure, tab_colonnes[m].colonne_origine);
                if (m<nb_col) strcat(ligneselectjointure, ", ");
                else  strcat(ligneselectjointure, ") FROM ");
                m++;
        }
        strcat(ligneselectjointure, table_origine);
        strcat(ligneselectjointure, ", ");
        strcat(ligneselectjointure, table_cible);
        strcat(ligneselectjointure, " WHERE ");
        m=0;
        while (m<=nb_col)
        {
                strcat(ligneselectjointure, table_origine);
                strcat(ligneselectjointure, ".");
                strcat(ligneselectjointure, tab_colonnes[m].colonne_origine);
                strcat(ligneselectjointure, "=");
                strcat(ligneselectjointure, table_cible);
                strcat(ligneselectjointure, ".");
                strcat(ligneselectjointure, tab_colonnes[m].colonne_cible);
                if (m<nb_col) strcat(ligneselectjointure, " AND ");
                m++;
        }

}
if (adr2 != NULL)
{
        strcat(ligneselectjointure, " AND (");
        strcat(ligneselectjointure, condition_param_edt);
        strcat(ligneselectjointure, ")");
}
```

```
        strcat(ligneselectjointure, ";");
        printf("\n%s\n",ligneselectjointure);

// lancement de la requête pour obtenir le COUNT
        strcpy(requete,ligneselectjointure);
        connexion();
        requete_select(requete);

// gestion du booléen ligne_command_ok_ko
// si résultat positif, on ne peut pas lancer le delete
        if (atoi(res_count) > 0)
        {
                ligne_commande_ok_ko = FALSE;
                printf("\n %s : instruction refusee",res_count);
        }
        else
                printf("\n %s : instruction acceptee",res_count);
        }



 /* ----------------------------------------------------------------*/

    if (type_traitement == 2)
    {
     /*     création du select type insert       */
        m=0;
        nb_col_insert_select=-1;
        while (m<=nb_col)
        {
                n=0;
                while
                (strcmp(strupr(tab_colonnes[m].colonne_origine),tab_colonnes_insert[n].colon
                ne_insert) !=0 && n<15)
                {
                        n++;
                }

                if
                (strcmp(strupr(tab_colonnes[m].colonne_origine),tab_colonnes_insert[n].colon
                ne_insert) ==0)
                {
                        nb_col_insert_select++;
                        if (m<1)
                        {
                                if (strcmp(tab_colonnes_insert[n].valeur_insert,"NULL") != 0)
                                {
                                  strcpy(ligneselectjointure, "SELECT COUNT(*) FROM ");
                                        strcat(ligneselectjointure, table_cible);
```

```
                        strcat(ligneselectjointure, " WHERE ");
                    }
                }
                    /* conversion vers champ colonne cible */
                if (m>0)
                {
                    strcat(ligneselectjointure, " AND ");
                }
                o=0;
                while (o<=nb_col)
                {
                    if(strcmp(strupr(tab_colonnes[m].colonne_origine),
                      tab_colonnes[o].colonne_origine) ==0)
                    {
                        strcat(ligneselectjointure,
                        tab_colonnes[o].colonne_cible);
                    }
                    o++;
                }
                strcat(ligneselectjointure, " = ");
                strcat(ligneselectjointure, tab_colonnes_insert[n].valeur_insert);
            }
            m++;
        }

        strcat(ligneselectjointure, ";");

        if (nb_col_insert_select==nb_col)
        {
            printf("\n%s\n",ligneselectjointure);

// lancement de la requête pour obtenir le COUNT

            strcpy(requete,ligneselectjointure);
            strcat(requete,"\0");
            connexion();
            requete_select(requete);

// gestion du booléen ligne_command_ok_ko
// si résultat == 0, on ne peut pas lancer l'insert car pas de valeur référencée
            if (atoi(res_count) == 0)
            {
                ligne_commande_ok_ko = FALSE;
                printf("\n %s : instruction refusee",res_count);
            }
            else
                printf("\n %s : instruction acceptee",res_count);
        }
```

```
        else
        {
                printf("\n erreur sur les colonnes de la cle");
                ligne_commande_ok_ko = FALSE;
                printf("\n instruction refusee");
        }
    }

/* ----------------------------------------------------------------*/

    if (type_traitement == 3)
    {
     /*      création du select type update       */
     if (update_to_from == 2)  /* clé de type TO */
     {
       m=0;
     /* p, compteur utile pour savoir si nombre de colonnes update est bien égal au nombre
     de colonnes dans la clé */
       p=-1;
       while (m<=nb_col_update)
       {
           if (strcmp(tab_colonnes_update[m].valeur_update,"NULL") != 0)
           {
               n=0;
               while (n<=nb_col)
               {
               if
               (strcmp(strupr(tab_colonnes[n].colonne_origine),tab_colonnes_update[m].colo
               nne_update) ==0)
               {
                       p++;
                       if (p==0)
                       {
                               strcpy(ligneselectjointure,"SELECT COUNT(*) FROM ");
                               strcat(ligneselectjointure,table_cible);
                               strcat(ligneselectjointure," WHERE ");
                       }
                       if (p>0)
                       {
                               strcat(ligneselectjointure, " AND ");
                       }
                       strcat(ligneselectjointure, tab_colonnes[n].colonne_cible );
                       strcat(ligneselectjointure, "=" );
                       strcat(ligneselectjointure, tab_colonnes_update[m].valeur_update );
                       }
                       n++;
               }
       }
       m++;
       }
```

```
        //printf("\n compteur %d ,%d",nb_col,p);
        /* requête lancée si bonne concordances entre colonnes update et colonnes de la clé */
        if (p == nb_col)
        {
          strcat(ligneselectjointure, ";");
          strcat(ligneselectjointure, "\0");
          printf("\n%s\n",ligneselectjointure);

// lancement de la requête pour obtenir le COUNT
          strcpy(requete,ligneselectjointure);
          connexion();
          requete_select(requete);

// gestion du booléen ligne_command_ok_ko
// si résultat == 0, on ne peut pas lancer l'update car pas de valeur référencée
          if (atoi(res_count) == 0)
          {
                ligne_commande_ok_ko = FALSE;
                printf("\n %s : instruction refusee",res_count);
          }
          else
          {
                printf("\n %s : instruction acceptee",res_count);
          }
        }

        else
        { /* instruction refusée si un partie de la clé est updatée*/
          if (p >=0)
          { printf("\n instruction refusee \n nombre de colonnes a modifier incorrect\n");
                ligne_commande_ok_ko = FALSE;
          }
            /* else on modifie une zone externe a la clé ==> ok */
        }
      }


        if (update_to_from == 1)      /* cle de type FROM */
        {
          strcpy(ligneselectjointure, "\0");
          m=0;
          while (m<=nb_col_update)
          {
                if (strcmp(tab_colonnes_update[m].valeur_update,"NULL") != 0)
                {
                      n=0;
                      while (n<=nb_col)
                      {
```

```
                    if
                    (strcmp(strupr(tab_colonnes[n].colonne_origine),tab_colonnes_upd
                    ate[m].colonne_update) ==0)
                    {
                       strcpy(ligneselectjointure, "SELECT COUNT(DISTINCT ");
                       strcat(ligneselectjointure, table_origine);
                       strcat(ligneselectjointure, ".");
                       strcat(ligneselectjointure, tab_colonnes[n].colonne_origine);
                       strcat(ligneselectjointure, ") FROM ");
                       strcat(ligneselectjointure, table_origine);
                       strcat(ligneselectjointure, ", ");
                       strcat(ligneselectjointure, table_cible);
                       strcat(ligneselectjointure, " WHERE ");
                       q=0;
                       while (q<=nb_col)
                       { if (q>0)
                         {
                              strcat(ligneselectjointure, " AND ");
                         }
                         strcat(ligneselectjointure, table_origine);
                         strcat(ligneselectjointure, ".");
                         strcat(ligneselectjointure, tab_colonnes[q].colonne_origine);
                         strcat(ligneselectjointure, "=");
                         strcat(ligneselectjointure, table_cible);
                         strcat(ligneselectjointure, ".");
                         strcat(ligneselectjointure, tab_colonnes[q].colonne_cible);
                         q++ ;
                       }

                       if (adr3 != NULL)
                       {
                              strcat(ligneselectjointure, " AND (");
                              strcat(ligneselectjointure, condition_param_edt);
                              strcat(ligneselectjointure, ")");
                       }

                       strcat(ligneselectjointure, ";");
                       printf("\n%s\n",ligneselectjointure);

// lancement de la requête pour obtenir le COUNT
                       strcpy(requete,ligneselectjointure);
                       connexion();
                       requete_select(requete);

// gestion du booléen ligne_command_ok_ko
// si résultat positif, on ne peut pas lancer le update (FROM)
                       if (atoi(res_count) > 0)
                       {
                              ligne_commande_ok_ko = FALSE;
                              printf("\n %s : instruction refusee",res_count);
```

```
                                }
                                else
                                {
                                        printf("\n %s : instruction acceptee",res_count);
                                }
                        }
                        n++;
                }
                }
                m++;
                }
                }
        }
        reinit();
        token = strtok( NULL, seps );
        i++;
  }
}


/* ---------- MAIN ----------*/

extern "C" __declspec(dllexport) int validation (char* ligne_from_ext){

ctrl_retour = 0;
ligne_commande= ligne_from_ext;

/* conversion ligne de commande en lettres majuscules, en MySQL, pas de problème de
Casse ! */

strcpy(ligne_commande_copy,ligne_commande);
ligne_commande = strupr(ligne_commande);
ligne_commande = strtok(ligne_commande, seps );

// par défaut Valeur True pour le passage de la ligne de commande vers la BD

ligne_commande_ok_ko = TRUE;


/* récupération des 6 premiers caractères pour la suite */
strncpy (type_trt, ligne_commande,6);

/* traitement delete */
        if (strcmp(type_trt,"DELETE") == 0)
        {
                type_traitement=1;
                adr1 = strstr(ligne_commande,"FROM ") + 5;
                adr2 = strstr (ligne_commande,"WHERE ");
                if (adr2 != NULL)
                {
```

```
                length = adr2 - adr1 - 1;
                strncpy (table_commande,adr1,length);
                table_commande[length]='\0';
                condition_param = adr2 + 6;
                strcat(condition_param, "\0");
        }
        else
        {
                strcpy (table_commande,adr1);
        }

        table_param = table_commande;

        formatage_condition_delete_update();

        strcat(table_param, "@");
        strcat(table_param, "\0");

        ligne = rechercher_cle_FROM();

        printf("\ncle FROM %d", number);

        verifier_integrite();

        traitement_BD();

}

if (strcmp(type_trt,"INSERT") == 0)
{
        type_traitement=2;
        adr1 = strstr(ligne_commande,"INTO ") + 5;
        adr2 = strstr (ligne_commande,"(");
        if (adr2 != NULL)
        {
                length = adr2 - adr1 - 1;
                strncpy (table_commande,adr1,length);
                table_commande[length]='\0';
                colonnes_insert = adr2;
                strcat(colonnes_insert, "\0");
        }
        else
        {
                strcpy (table_commande,adr1);
        }

        table_param = table_commande;

        formatage_colonnes_insert();
```

```
            strcat(table_param, "@");
            strcat(table_param, "\0");

            ligne = rechercher_cle_TO();

            printf("\ncle TO");

            verifier_integrite();

            traitement_BD();
    }


    if (strcmp(type_trt,"UPDATE") == 0)
    {
            type_traitement=3;
            adr1 = strstr(ligne_commande,"UPDATE ") + 7;
            adr2 = strstr (ligne_commande,"SET ") ;
            length = adr2 - adr1 - 1;
            strncpy (table_commande,adr1,length);
            table_commande[length]='\0';
            colonnes_update = adr2 + 4;
            strcat(colonnes_update, "\0");
            table_param=table_commande;

            formatage_colonnes_update ();

            adr3 = strstr (ligne_commande,"WHERE ");

            if (adr3 != NULL)
            {
                    condition_param = adr3 + 6;
                    strcat(condition_param, "\0");
                    formatage_condition_delete_update();
            }

            strcat(table_param, "@");
            strcat(table_param, "\0");



/* ------- partie FROM ------- */
            ligne = rechercher_cle_FROM();
             printf("\ncle FROM %d ", number);
            update_to_from = 1;
            verifier_integrite();


/* ------partie TO ------ */
            ligne = rechercher_cle_TO();
```

```
                printf("\ncle TO %d", number);
                update_to_from = 2;
                verifier_integrite();

                traitement_BD();

        }

        deconnexion();

  // getchar ();

/*réinitialisation pour la prochaine itération */

        strcpy(condition_param,"\0");
        j=0;
        strcpy(condition_param_tmp," ");
        while (j<900)
        {
                strcat(condition_param_tmp," ");
                j++;
        }
        strcpy(condition_param_tmp,"\0");

        j=0;
        strcpy(condition_param_tmp2," ");
        while (j<900)
        {
                strcat(condition_param_tmp2," ");
                j++;
        }
        strcpy(condition_param_tmp2,"\0");

        j=0;
        strcpy(condition_param_edt," ");
        while (j<900)
        {
                strcat(condition_param_edt," ");
                j++;
        }
        strcpy(condition_param_edt,"\0");

        j=0;
        strcpy(ligneselectjointure," ");
        while (j<1000)
        {
                strcat(ligneselectjointure," ");
                j++;
        }
        strcpy(ligneselectjointure,"\0");
```

```
        j=0;
        strcpy(requete," ");
        while (j<500)
        {
                strcat(requete," ");
                j++;
        }
        strcpy(requete,"\0");

        return ctrl_retour;
}

/* code par défaut à insérer lors de la création d'une DLL */

BOOL APIENTRY DllMain( HANDLE hModule,
                DWORD ul_reason_for_call,
                LPVOID lpReserved )
{
  switch (ul_reason_for_call)
        {
                case DLL_PROCESS_ATTACH:
                case DLL_THREAD_ATTACH:
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        break;
  }
  return TRUE;
}
```

Annexe 4 : Le script d'un programme faisant appel à la librairie de fonctions de gestion des clés étrangères

Cette partie contient les lignes de code d'un programme client faisant appel à la librairie de fonctions relatives à la gestion des clés étrangères. Ce programme consiste en une console simple qui permet à un utilisateur d'entrer une commande d'interaction avec une base de données MySQL. Comme expliqué dans le chapitre 6, il faut avoir défini au préalable un lien ODBC vers la base de données avec laquelle le programme client va interagir. Par après, il faut importer la librairie de fonctions de gestion des clés étrangères dans l'environnement du programme client. Ensuite, à l'intérieur du programme, les fonctions de la librairie *(définition du lien ODBC et validation de l'instruction)* sont importées et appelées suivant la méthode décrite dans la section 6.4. Le résultat de la validation est retourné et la console affiche un message pour informer l'utilisateur de l'acceptation ou du refus de la commande entrée.

```
#include <stdio.h>
#include <string.h>

extern "C" __declspec(dllimport)  void definir_nom_lien_odbc (char* nom_lien);
extern "C" __declspec(dllimport)  int validation (char* ligne_from_ext);


void main()
{
        char commande[2000];
        int ctrl;

        /* le nom du lien ODBC est mis en argument de la procédure */
        definir_nom_lien_odbc("lien_odbc_baseMySQL");

        while ((strcmp(commande,"FIN") !=0) && (strcmp(commande,"fin") !=0))
                {
                printf("\n>Veuillez entrer ligne de commande (pour terminer
                                tapez 'FIN') \n>");

                gets(commande);

                if  ((strcmp(commande,"FIN") !=0) && (strcmp(commande,"fin") !=0) &&
                    (strcmp(commande,"")!=0))

                        ctrl=validation(commande);

                printf("\n %d",ctrl);
        }
}
```