

**UNIVERSITÉ DE MONS-HAINAUT
FACULTÉ DES SCIENCES**

Interpréteur pour l'Algèbre Relationnelle

Vince Stéphane

Service de science des systèmes d'information

Directeur : Jef Wijzen

Année académique 2003-2004

Table des matière

I.	INTRODUCTION.....	4
II.	DESCRIPTION DU PROJET.....	5
1.	L'ALGEBRE RELATIONNEL SPJRUD	5
1.	<i>Les opération de base :</i>	5
2.	<i>Définition BNF</i>	5
2.	LE SIMPLE QUERY FORMAT	5
3.	LE PROGRAMME	7
III.	IMPLEMENTATION.....	8
1.	CHOIX TECHNIQUES.....	8
3.	<i>Généralités</i>	8
4.	<i>Choix du SGBD</i>	8
5.	<i>Choix du langage de programmation</i>	10
2.	STRUCTURE DE L'APPLICATION	10
1.	<i>Structure globale (flux d'information)</i>	10
2.	<i>Structure du traitement</i>	11
3.	<i>Schéma des modules et des classes</i>	12
4.	<i>La gestion des erreurs</i>	14
5.	<i>Exemple d'affichage et compatibilité</i>	14
3.	PROBLEMES RENCONTRES	15
1.	<i>Dans Visual Basic</i>	15
2.	<i>Propre au SGBD</i>	16
IV.	CONCLUSION.....	18
V.	RÉFÉRENCES	20
1.	SITES WEB	20
2.	LIVRES	20

I. Introduction

Dans le cadre de son cours de base de données, Monsieur Wijzen souhaite fournir à ses étudiants un outil interprétant des requêtes formatées en algèbre relationnelle.

Le but de ce projet est de créer un programme didactique permettant aux étudiants en bases de données de saisir des requêtes en algèbre relationnelle et d'en observer le résultat.

Afin de répondre au mieux à ces objectifs, il semble évident que l'application se doit d'être assez facile d'installation et aisément distribuable et ce afin que les étudiants puissent non seulement l'exécuter dans les laboratoires, mais également à domicile.

Enfin, comme il s'agit d'étudiants (et non d'experts) en base de données, une bonne gestion des erreurs indiquant le plus clairement possible les fautes de syntaxes ainsi que certaines fautes de logique semble être incontournable.

II. Description du projet

1. L'algèbre relationnel SPJRUD

1. Les opérations de base :

- **S - SELECT** : Permet de ne retenir que les tuples qui satisfont une condition donnée.
- **P - PROJECT** : Permet de ne retenir que certains attributs d'une relation.
- **J - JOIN** : Pour joindre deux relations qui peuvent avoir des attributs communs.
- **R - RENAME** : Pour renommer un attribut d'une relation.
- **U - UNION** : Equivalant de l'union de la théorie ensembliste.
- **D - DIFFERENCE** : Equivalant de la différence ensembliste.

2. Définition BNF

Les notations mathématiques de l'algèbre relationnelle ($\sigma, \pi, \bowtie, \rightarrow, \cup, -$), étant difficilement encodables à l'aide d'un clavier, une syntaxe spécifique et intuitive a été définie :

```
<relational expression> ::= <relvar name> | <relational operation> |  
                           (<relational expression>)  
<relational operation> ::= <select> | <project> | <join> | <rename> |  
                           <union> | <difference>  
<select> ::= <relational expression> WHERE <boolean expression>  
<project> ::= <relational expression> PROJECT <attribute name commalist>  
<join> ::= <relational expression> JOIN <relational expression>  
<rename> ::= <relational expression> RENAME <renaming commalist>  
<union> ::= <relational expression> UNION <relational expression>  
<difference> ::= <relational expression> MINUS <relational expression>
```

2. Le Simple Query Format

Monsieur Wijzen a également défini une syntaxe de travail baptisée Simple Query Format (SQF), dont les mots clés se situent obligatoirement en début de ligne, et son précédé du caractère '@'. Les instructions SQF sont au nombre de cinq :

- Définition des données :
 - @RELATION, suivi du nom de la relation à créer.
 - @ATTRIBUTE, suivi du nom d'un attribut de la relation.
 - @DATA, suivi d'un retour à la ligne et d'une liste de données au format CSV¹.

¹ CSV - Comma Separated Value : chaque donnée est séparée par une virgule et chaque tuple par un retour à la ligne.

- Définition des requêtes :
 - @LET, suivi d'une instruction d'algèbre relationnelle.
 - @PRINT, suivi du nom d'une relation (provoque l'impression de cette relation en suivant la syntaxe de définition de données ci-dessus).

La possibilité d'ajout de commentaire est également possible ; les lignes précédées du caractère '%' sont ignorées.

Plus concrètement, si on prends l'exemple de deux relations simple à comprendre :

COURS	Nom	Prof	Ects
	Gestion de projets	Mens	4
	Base de données	Wijsen	10
	Analyse	Troestler	10

NOTES	Nom	C	Note
	Ed	Gestion de projets	13
	Ed	Base de données	15
	Tim	Base de données	13
	Eric	Gestion de projets	19

- Exemple de fichier SQF reprenant ces deux tables ainsi qu'une série de requêtes avec l'impression de deux d'entre elles :

```
% Ceci est le contenu du fichier input.sqf.
% On décrit d'abord les tables.
@relation COURS
@attribute Nom
@attribute Prof
@attribute Ects
@data
Gestion de projets, Mens, 4
Bases de données, Wijsen, 10
Analyse, Troestler, 10
@relation NOTES
@attribute Nom
@attribute C
@attribute Note
@data
Ed, Gestion de projets, 13
Ed, Bases de données, 15
Tim, Bases de données, 13
Eric, Gestion de projets, 19
% Puis on spécifie les requêtes.
@let CCOURS = COURS RENAME Nom AS C
@let M = (CCours WHERE Prof="Mens") PROJECT C
@let EM = (M JOIN NOTES) PROJECT Nom
@print EM
@let A_EU = (CCOURS JOIN NOTES) PROJECT Nom, Prof
@let TOUT = (COURS PROJECT Prof) JOIN (NOTES PROJECT Nom)
@let N_A_PAS_EU = (TOUT MINUS A_EU) RENAME Nom AS Etudiant
@print N_A_PAS_EU
```

- Les relations renvoyées par le programme seront donc les suivantes :

EM	Nom
	Ed
	Eric

N_A_PAS_EU	Etudiant	Prof
	Ed	Troestler
	Eric	Troestler
	Eric	Wijsen
	Tim	Mens
	Tim	Troestler

- Les mêmes relations au format SQF :

```
@relation EM
@attribute Nom
@data
Ed
Eric
@relation N_A_PAS_EU
@attribute Etudiant
@attribute Prof
@data
Ed, Troestler
Eric, Troestler
Eric, Wijsen
Tim, Mens
Tim, Troestler
```

3. Le programme

Le programme sera une application de type console prenant comme argument le nom du fichier (ou des fichiers) qu'il devra traiter. La commande proposée est :

```
spjrud [datas.sqf] queries.sqf
```

L'idée est de laisser le choix à l'utilisateur :

- Soit de travailler dans un seul fichier contenant les données et les requêtes.
- Soit d'utiliser un fichier pour les données et un autre pour les requêtes.

Le résultat est renvoyé sur la sortie standard (qui peut être redirigée vers un fichier à l'aide de l'opérateur '>').

III. Implémentation

1. Choix techniques

3. Généralités

Au vu de l'énoncé du problème, on se pose d'abord la question de savoir par quoi les données vont être traitées. Et là, deux choix s'offrent à vous :

- Soit, se lancer dans l'écriture de son propre moteur de traitement des données (avec le *petit soulagement* de savoir qu'il ne doit pas être spécialement performant puisqu'il traitera un nombre très restreint de tuples)
- Soit, de s'appuyer sur un système de gestion de base de données existant, le cœur de l'interpréteur « se contentant » de transformer l'algèbre relationnel en SQL.

Dans mon cas j'ai préféré retenir la deuxième solution pour diverses raisons. Tout d'abord le premier choix me paraissait stratégiquement plus risqué, l'entreprise d'écrire un moteur de base de données aussi léger soit-il m'a clairement refroidi. Et puis, est-il toujours judicieux de réinventer la roue ? Ensuite, étant assez familier des SGBD, la syntaxe SQF me paraissant, à première vue, fort proche de celle du SQL, le challenge de l'écriture d'un convertisseur SQF → SQL me paraissait plus abordable.

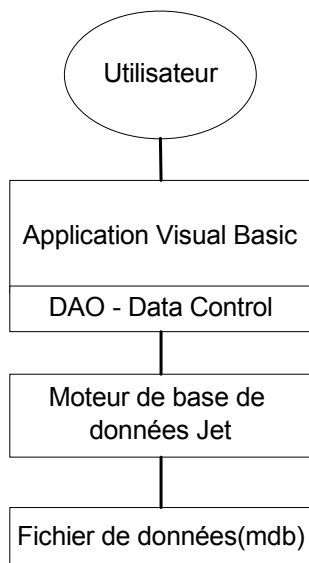
4. Choix du SGBD

La recherche du système de gestion de base de données fût un travail bien plus long que celui que je m'imaginai au départ... Ma première idée était d'essayer de travailler avec un SGBD issu du monde libre afin de m'affranchir des problèmes de licences et autres contraintes rédhibitoires. Il fallait cependant garder à l'esprit que l'application que j'allais développer devait être facilement installable et malgré que l'utilisateur cible est étudiant en informatique, il n'est probablement pas expert en base de données. J'étais donc à la recherche d'un petit moteur de base de données que je pouvais interroger en SQL. Autant dire que je cherchais une voiture avec des roues et un volant...

Mes premiers essais furent donc réalisés avec le très célèbre MySQL et je comptais programmer l'interpréteur en C++. En effet, ce SGBD est très répandu et facile d'installation. Quelle déception... Tout d'abord, la taille de la distribution standard est tout de même, à l'heure actuelle, d'une vingtaine de mégas (ce qui est assez conséquent par rapport aux quelques kilooctets de données que nous allons traiter). Ensuite, bien que le moteur MySQL existe pour la plupart des plates-formes, les bibliothèques clientes déjà compilées, elles, n'existent que pour linux. Mais, il m'en faut plus pour m'arrêter, je décide donc de me lancer dans la compilation des bibliothèques clientes... Tout d'abord,

je tente de prendre les sources linux et de les compiler avec le portage de GCC sous Windows, sans succès, on pouvait s'y attendre... Plus tard, je trouve une librairie écrite en C++ pour le compilateur Windows de Borland (qui est disponible gratuitement au téléchargement) mais une fois de plus, je dus déchanter car, une fois la librairie compilée (avec quelques warnings), cette dernière s'avéra fonctionnelle mais très instable (certaines requêtes faisaient « planter » mon PC). De plus, ma réflexion mûrissante quant à la façon de convertir efficacement le Simple Query Format, m'amena à penser que le SGBD devrait supporter la définition de vues, ce qui n'est pas le cas de MySQL². C'en était trop, je devais me résoudre à laisser tomber cette piste...

La deuxième idée fut tuée dans l'œuf. Un très bon SGBD du monde libre et curieusement plus méconnu du tout un chacun est sans aucun doute PostgreSQL. L'étendue de ses capacités est réellement impressionnante, mais dans cette situation, il fallait juste qu'il supporte la définition de vues, ce qui était le cas. Le problème c'est qu'on ne peut pas imposer aux futurs étudiants d'informatique d'être tous des linuxiens chevronnés et convaincus. J'entends par là qu'il n'existe pas à ce jour une version Windows exécutable³. Il me fallait trouver autre chose...



Comme souvent, la troisième fut la bonne. Après une brève introspection, je me suis souvenu d'avoir lu récemment qu'aujourd'hui encore, plus de 95 % des PC étaient équipés de systèmes d'exploitations Microsoft. Bien que le milieu des informaticiens soit plus propice à affaiblir ce pourcentage, il me semble raisonnable de penser qu'une très large majorité des ordinateurs que nous utilisons tourne sous Windows. J'en veux pour preuve que tous les labos d'informatique que j'ai eu l'occasion de fréquenter à l'université étaient tous, sans exception, équipés de cet OS. Je ne cherche pas à faire l'éloge de Windows, il s'agit juste d'un constat. Mais qu'a donc cet OS de si intéressant ? Et bien, c'est qu'il comporte nativement un moteur de base de données répondant à tous les critères susmentionnés à savoir Microsoft Jet⁴, pour ne pas le nommer. De plus, la librairie d'accès à Jet (dao.dll) se trouve également en standard dans l'OS. Toutes les conditions semblent donc réunies pour que ce SGBD soit le support de ma future application, au seul sacrifice que cette dernière ne tournera que sous Windows... Ainsi soit-il !

² Bien que ce problème peut être résolu en créant une nouvelle table en lieu et place d'une vue, mais c'est un processus réellement très lourd et très *inélegant*.

³ Encore une fois, une solution alternative existe pour faire tourner n'importe quel programme linux sous Windows, par exemple par l'intermédiaire d'émulateur tel que cygwin. Mais si je suivais cette piste, je m'éloignais une fois de plus de la contrainte « simple d'installation ».

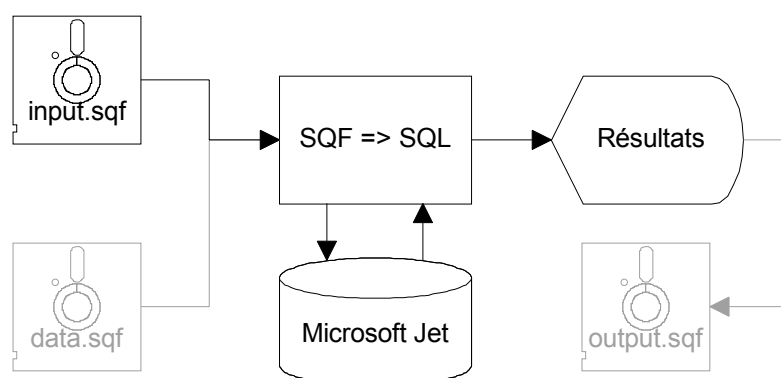
⁴ Une autre alternative gratuite également disponible chez Microsoft était MSDE, mais elle nécessitait une grosse installation (45 Mo) et est optimisée pour les grosses bases de données.

5. Choix du langage de programmation

Pour pouvoir utiliser la librairie d'accès DAO⁵, deux choix possibles : Visual Basic ou Visual C++. Après avoir comparé les deux produits, mon dévolu s'est porté sur Visual Basic pour diverses raisons. Premièrement, la communauté d'utilisateurs du couple Jet + VB est vraiment très étendue, ce qui est très intéressant, car j'étais peu habitué à ce langage (j'avais déjà eu l'occasion de pratiquer un peu de VBA - Visual Basic for Application aux sein de la suite office, *comme tout le monde*, pour réaliser des petits traitements infaisables à l'aide des macro). De plus, l'approche du C++ par Microsoft est légèrement différente aussi bien en terme d'environnement de développement que en terme de compilateur par rapport à celle proposée par Borland à laquelle je suis plus familier. Enfin, je dois avouer que bon nombre de mes collègues m'avait déjà vanté les mérites de Visual Basic pour l'interfaçage avec les bases de données Microsoft⁶. J'ai donc décidé de me forger mon propre avis en réalisant ce projet à l'aide de cette technologie !

2. Structure de l'application

1. Structure globale (flux d'information)



Description : Afin de répondre à l'énoncé, l'application prendra 1 ou 2 fichier(s) en entrée, effectuera un traitement en s'appuyant sur le Moteur Jet, puis finalement enverra ses résultats sur la sortie standard (éventuellement redirigée vers un fichier).

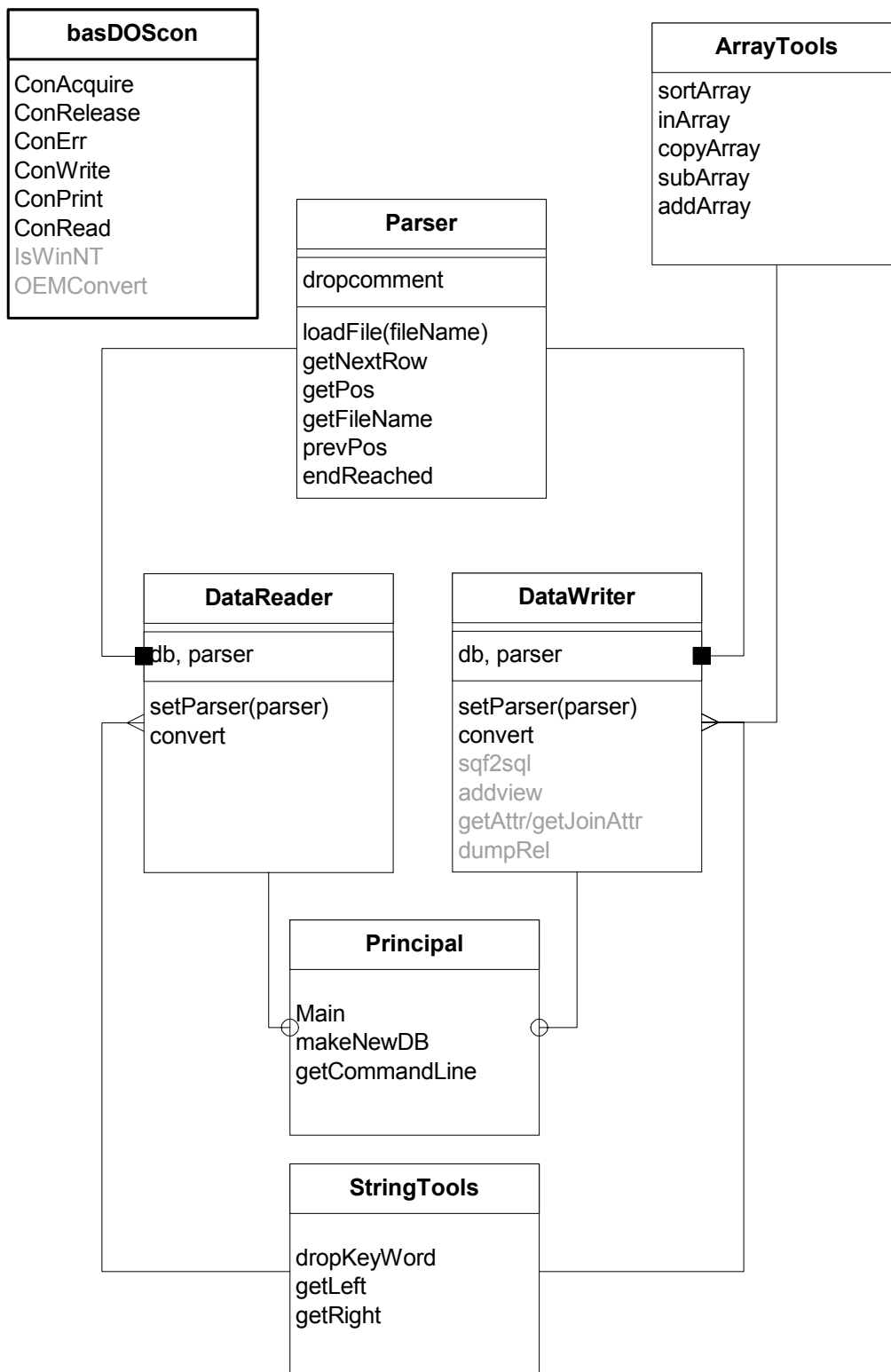
⁵ DAO : Data Access Objects.

⁶ Qu'elles reposent sur SQL Serveur, MSDE ou Jet.

2. Structure du traitement

- **Chargement du fichier en mémoire avec normalisation du texte (parsing)** : Chaque ligne se verra supprimer les éventuels espaces de début et de fin de ligne. Les tabulations seront converties en espaces. Les espaces entre les mots, s'ils sont multiples, seront ramenés à un seul.
- **Interprétation des instructions de définitions de données** : Les 3 premières instructions SQF (@relation, @attribute, @data) seront traduites dans leur expression équivalente SQL (CREATE TABLE, INSERT INTO). La détection d'erreur veillera à vérifier la cohérence des définitions.
- **Interprétation des instructions de définitions de requêtes et d'impression de résultats** : Les instructions d'algèbre relationnelle SPJRUD (directement précédée de l'instruction SQF @let) seront converties en une définition de vues SQL, et ce ligne par ligne (si il y a imbrication d'instructions à l'aide de parenthèses, une vue temporaire sera crée pour chaque niveau d'imbrication). Dès qu'une instruction @print sera rencontrée, elle enclenchera la procédure d'impression des données de la relation concernée sur la sortie standard. La détection d'erreur sera assurée en partie par l'application et en partie par le moteur de base de données.

3. Schéma des modules et des classes



- basDOScon** : Ce module contient toutes les procédures qui permettent à Visual Basic d'écrire, de lire et d'envoyer les erreurs sur les flux standard STDIN, STDOUT, STDERR (voir le point 3 pour plus de détails).

- **Parser** : cette classe assure le parsing des fichiers passés en arguments à la ligne de commande. Le fichier est intégralement chargé en mémoire et est retravaillé de façon à ne laisser qu'un seul espace entre chaque mot. Les lignes vides et les lignes de commentaires sont également supprimées (cette fonction est paramétrable à l'aide de l'attribut booléen *dropComment*). Les méthodes principales sont :
 - *loadFile* : permet de spécifier l'emplacement du fichier à charger.
 - *getNextRow* : renvoie le contenu de la ligne suivante.
 - *getPos* : renvoie le numéro de ligne qu'on est occupé à traiter (très utile dans la gestion d'erreur).
 - *prevPos* : Recule le pointeur d'instruction de une position.
 - *endReached* : Renvoie *True* si on a atteint la fin du fichier.

- **DataReader** : cette classe a pour rôle de convertir les instructions SQF de définitions de données. Elle reçoit comme principal paramètre une instance de Parser ainsi que le nom du fichier de base de données.

- **DataWriter** : cette classe assure la conversion des instructions SQF et SPJRUD de définition des requêtes vers le langage SQL. Elle reçoit une instance de Parser contenant les lignes de codes qui lui sont destinées. Ces méthodes privées sont :
 - *sqf2sql* : c'est la procédure clé de l'application, c'est elle qui converti une instruction SPJRUD en une requête SQL. Elle effectue également des tests afin de vérifier la validité de certaines instructions.
 - *addView* : ajoute une vue à la BD.
 - *getAttr* : renvoie les attributs d'une relation
 - *getJoinAttr* : renvoie les attributs communs à deux relations.
 - *dumpRel* : renvoie, au format SQF, la définition et les données d'une relation sur la sortie standard.

- **ArrayTools** : Ce module contient des procédures classiques d'opérations à effectuer sur les divers tableaux que contient l'application.

- **StringTools** : Ce module contient trois fonctions constamment employée par DataReader et DataWriter :
 - *dropKeyword* : Supprime un mot clé SQF et renvoie le reste de la ligne si le mot clé est trouvé, sinon renvoie une chaîne vide.
 - *getLeft* : renvoie ce qui est à gauche d'un caractère choisi.
 - *getRight* : renvoie ce qui est à droite d'un caractère choisi.

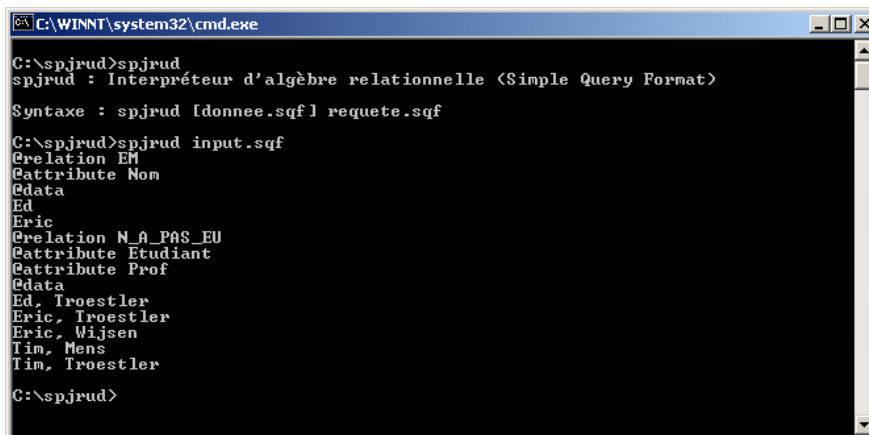
- **Principal** : Ce module contient la procédure main, la procédure de création du fichier MDB, ainsi qu'un mini-parser pour la ligne de commande.

4. La gestion des erreurs

Visual Basic propose une structure de gestion des erreurs assez basique, mais qui a le mérite d'exister. L'instruction 'On Error' permet notamment d'effectuer un branchement à une sous routine si une interruption du programme pour erreur est déclenchée dans le code qui lui succède. Le langage permet également au programmeur de provoquer des erreurs à l'aide de l'objet 'Err' et de sa méthode 'Raise'. Si l'erreur n'est pas interceptée dans la routine en cours, elle est propagée à la procédure ou à la fonction appelante et ainsi de suite jusqu'à l'éventuel arrêt du programme en lui même.

Pour cette application, j'ai instauré deux niveaux d'erreurs. Les erreurs fatales et les warnings. Les premières sont soit déclenchées volontairement (ex : détection d'une erreur dans les parenthèses), soit propagées par un composant de l'application (le plus souvent, il s'agit du moteur Jet) dans le cas d'une erreur de logique par exemple(c'est le cas si l'utilisateur tente de définir deux relations portant le même nom). Les erreurs fatales sont propagées jusqu'à la procédure *main* qui les intercepte⁷ et les rediriges vers STDERR puis met fin à l'application. Le warnings se décline également en deux catégories : soit ils sont envoyés suite à un test qui les détecte, soit ils sont interceptés par l'instruction 'On Error' et l'erreur n'est pas propagée. Dans les deux cas, l'erreur est imprimée directement sur STDERR.

5. Exemple d'affichage et compatibilité



```
C:\WINNT\system32\cmd.exe
C:\spjrud>spjrud
spjrud : Interpréteur d'algèbre relationnelle (Simple Query Format)
Syntaxe : spjrud [donnee.sqf] requete.sqf
C:\spjrud>spjrud input.sqf
Relation EM
Attribute Nom
data
Ed
Eric
Relation N_A_PAS_EU
Attribute Etudiant
Attribute Prof
data
Ed, Troestler
Eric, Troestler
Eric, Wijsen
Tim, Mens
Tim, Troestler
C:\spjrud>
```

Ci-dessus, un exemple de l'application exécutée sous Windows 2000.

Il est important à noter que le choix du langage assure que le programme fonctionnera sur toutes les versions de Windows 32 bits (95, 98, ME, NT 4, 2000 et XP).

⁷ Cette étape est nécessaire, car par défaut Visual Basic renvoie l'erreur dans une fenêtre de type *pop-up*, ce qui est incompatible avec une application purement console.

Les fichiers d'entrées doivent être au format ANSI – PC, la sortie est également formatée sous cette norme. Pour des raisons de convivialité, la sortie standard des erreurs (STDERR) renvoie ces valeurs au format de codage DOS.

La base de donnée est créée lors de l'appel du programme (elle est stockée dans le répertoire temporaire de l'utilisateur, ce qui évite les problèmes de droit en écriture) et est détruite à la fin de l'exécution de celui-ci. L'option '/d' (pour debug) a pour effet de conserver le fichier après le traitement. Elle a été prévue afin de pouvoir contrôler le contenu du fichier de la base données (spjud.mdb) à l'aide d'un programme compatible (ex : Microsoft Access).

3. Problèmes rencontrés

1. Dans Visual Basic

Si vous êtes familier de cet environnement, il vous semblera peut être curieux de réussir à développer une application console avec Visual Basic... En effet, il y a truc ! C'est grâce à une petite recherche sur un de mes sites de référence que j'ai trouvé le module *basDOScon*⁸ que j'ai légèrement modifié pour qu'il convertisse les erreurs formatée en ANSI, vers le format ASCII. Le module fournit, référence, directement la librairie d'accès au noyau du système '*kernel32.dll*' en se basant sur les spécifications fournies par Microsoft sur MSDN. L'auteur fourni également un linker modifié afin de palier au problème de la récupération directe des résultats sur la console courante plutôt que sur une console fille, comme c'est initialement le cas avec le linker de Microsoft.

La première grosse critique que je ferai à propos de ce langage est son comportement avec les tableaux. VB travaille toujours par référence avec les variables passées aux procédures et aux fonctions. Il n'est donc pas inconcevable que celui-ci autorise à une fonction de retourner un tableau (puisque que c'est l'adresse de ce tableau qui sera retourné). Toutefois, le compilateur se comporte étrangement quand les tableaux sont déclarés en série (ex : *Dim tab1(), tab2(), tab3() as String*), en effet ces tableaux ne sont pas directement affectables avec l'opérateur égal comme on est en droit de l'attendre. Pour parvenir à ses fins, il faut déclarer les variables une à une (ex : *Dim tab1() as String ... Dim tab2() as String*) ! Curieux, non ? De surcroît, il n'y a pas de fonction telle que *isEmpty()* pour permettre de déterminer « proprement » si un tableau est vide ou non. J'ai dû me contraindre à une interception d'erreur de type dépassement d'indice pour résoudre se problème, ce qui me paraît fort contraignant.

⁸ Page web de ce merveilleux et indispensable module :
<http://www.Planet-Source-Code.com/vb/scripts/ShowCode.asp?txtCodeId=38590&lngWId=1>

Ma deuxième critique concerne les possibilités objets que laissent présager la présentation du langage. On a l'impression que Microsoft s'est contenté de copier C/C++ en reproduisant les mêmes erreurs. La structure même du langage ne permet pas d'exploiter toutes les fonctionnalités de la programmation orientée objet tout simplement parce que l'héritage est impossible en VB. Pourtant, en parcourant la documentation, on y voit les mots polymorphisme, interface, etc. En fait, il est possible de décrire des interfaces en VB et de les implémenter différemment. C'est ce que Microsoft appelle du polymorphisme...

La troisième (et dernière) critique concerne la gestion d'événements. Tout simplement, parce que le déclenchement d'événements est toujours séquentiel. En fait VB ne propose pas de support pour le multi-threading au programmeur, ce qui fait que la gestion d'événements est essentiellement restreinte à leur capture au sein des interfaces graphiques développées. Pour ce qui est du déclenchement d'événement au sein du code, un appel à une procédure contenant le même code aurait exactement le même effet !

2. Propre au SGBD

Ils ont été divers et tiennent parfois de l'anecdote. Le premier a été la syntaxe de nomination des tables et des attributs propre SQL de Jet. Pour éviter les problèmes, chaque relation et chaque attribut est entouré de crochets (ex : [relation].[attribut]), ce qui permet de définir des noms contenant des espaces et des caractères spéciaux, mais (et c'est plus important) lève le problème des mots réservés.

Un autre problème a été rencontré pour la traduction du *JOIN*, bien que le SQL reprennent cette expression, il ne permet pas de faire un produit cartésien (dans le cas où il n'y a aucun attribut en commun). J'ai donc pris l'option de transformer les opérations *JOIN* en un produit cartésien restreint par une clause *WHERE* sur l'égalité des éventuels attributs communs.

Le troisième problème est que le moteur Jet autorise des attributs de même noms (pas dans les tables, bien sûr). Si on effectue une requête sur une relation constituée des attribut A et B, qui a pour but de renommer l'attribut B en A, le moteur l'accepte ! Il corrigera automatiquement et renverra les attributs A et Expr1. Cette erreur est donc détectée pendant la traduction du SQF vers le SQL et provoque une erreur fatale.

Les opérations ensemblistes (*UNION*, *MINUS*, *INTERSECT*) existent bien en SQL, mais Jet n'implémente que *UNION*. De plus, le SQL ne se soucie pas du nom des attributs pour effectuer l'union. En effet du moment que le nombre d'attribut est le même, il est satisfait. Cette erreur est donc détectée également pendant la traduction du SQF vers le SQL et provoque une erreur fatale. Comme Jet n'implémente pas le *MINUS*, il a été converti sous une autre forme. La commande SQL : (*SELECT nom FROM cours*) *MINUS* (*SELECT nom FROM notes*) devient donc *SELECT cours.nom FROM*

cours LEFT JOIN notes ON cours.nom = notes.nom WHERE notes.nom IS NULL, ce qui retourne le même résultat.

Le dernier problème est à ranger au rang des anecdotes. L'algèbre relationnelle ne permet d'avoir que deux tuples contenant les mêmes données. C'est pourquoi toutes les opérations *SELECT* sont suivies du mot clé *DISTINCT* afin de chasser les doublons.

IV. Conclusion

Ce projet m'a permis de découvrir l'étendue des possibilités ainsi que ce que je me permets de qualifier de faiblesses de Visual Basic. Je ne reviendrai pas sur les quelques défauts dont j'ai déjà fait état dans ce rapport, mes je m'attarderai sur les qualités qu'il faut lui reconnaître (et elles sont nombreuses).

Tout d'abord la syntaxe : les branchements et les boucles sont très explicites grâce à l'utilisation de mots clés non-ambigus tels que *if... Then ... Else ... End If* ou bien encore *Do While ... Loop*. La quasi obligation⁹ de retour à la ligne comme délimiteur d'instructions. Ensuite, la rigidité de la mise en forme automatique qui si dans un premier temps s'avèrent très contraignante en obligeant le programmeur de respecter certaine règle de formatage, s'avère finalement très bonne pour la production d'un code très lisible. Pour finir, les nombreuses fonctions disponibles pour réaliser le déboguage ainsi que le *backtraking* des erreurs apportent un réel confort lors des tests.

Il semble évident que cet environnement de développement est réservé au programmeurs « juniors », et qu'il a sa place parmi les outils à recommander.

Ce projet a été fort agréable et très amusant à réaliser. De plus, savoir qu'il est susceptible d'être utilisé par des étudiants à des fins didactiques n'a pas manqué d'ajouter une motivation supplémentaire afin d'arriver à un produit fini et fonctionnel.

⁹ On peut la contourner avec l'opérateur ' : '

V. Références

1. Sites Web

- **The Microsoft Developer Network (MSDN) – Visual Basic 6 :**
<http://msdn.microsoft.com/library/default.asp?URL=/library/devprods/vs6/vbasic/vbcon98/vbstartpage.htm>
- **VB France :**
<http://www.vbfrance.com/>
- **Planet Source Code :**
<http://www.Planet-Source-Code.com/vb/>
- **Free VB Code :**
<http://www.freevbcode.com/>

2. Livres

- **Visual Basic 6 et les bases de données :**
Rémy Lentzner
Editions Eyrolles
ISBN : 2-7464-0378-1
- **Visual Basic 6.0 : Guide du programmeur :**
Dominique Maniez
Microsoft Press
ISBN : 2-1000-7233-1