Annexe 4 : description technique du jeu de test

Sommaire

Introduction et mises en garde

Le présent document est une synthèse technique présentée en annexe à l'étude du format d'archivage des bases de données relationnelles, le SIARD. Il énumère toutes les requêtes qui ont servi à l'élaboration de la base de test. Pour autant, il n'est qu'un complément à la description détaillée des différentes étapes de l'étude. Un certain nombre de tests effectués sur SIARD ne trouve pas ses fondements dans les requêtes SQL mais dans la manipulation rigoureuse des éléments de la base. De nombreux détails ont été expérimentés directement à partir de requêtes du type ALTER TABLE qui ne sont pas reprises ici. Le cheminement intellectuel de l'étude se trouve pleinement retranscrit dans ce qui suit.

Il paraissait essentiel de fournir la matière des résultats observés lors du maniement de *SiardSuite* et du format SIARD. Les *dumps* de Oracle et MS SqlServer sont fournis en annexes électroniques. Cependant ils ne fonctionnent qu'avec les versions des SGBD utilisés, en l'occurrence Oracle 10g XE et SqlServer 2008 express. Les fichiers mdb et accdb accompagnent également les annexes et correspondent aux deux versions d'Access testées.

Au-delà de cette approche, il est primordial de reporter les requêtes employées pour créer la base de données et en simuler la gestion. Très peu de contenus a été enregistré dans les tables car l'étude se focalise davantage sur le comportement du processus que sur l'archivage réel des données. De plus, le traitement est beaucoup plus lent lorsque toutes les données primaires sont insérées dans l'archive SIARD.

Un langage de requêtes a été défini pour manipuler des bases de données relationnelles, il s'agit du SQL qui a donné lieu à des normes. La dernière parue date de 2009 mais le format SIARD ne se réfère qu'à la version de 1999. Les trois produits (SGBDR) supportés par *SiardSuite* n'emploient pour autant pas directement ce langage. Un bref état de la question est reporté à la suite. Les langages de requêtes employées sont de différents types :

Certains comme le PL/SQL[1] ou le Transact SQL[2] sont des langages procéduraux propriétaires qui permettent de combiner des requêtes SQl et des instructions procédurales à l'intérieur même des SGBD. Il existe donc des syntaxes particulières et différentes selon les produits mais de manière générale, si l'on possède des notions en SQL, il est aisé de les appréhender.

D'autres langages propriétaires ou non peuvent être employés pour administrer des bases de données. Dans le cas de Access, le VBA[3] est très couramment indiqué dès lors que l'on veut programmer des événements de type macros ou modules. De même, des langages comme PHP ou Java peuvent être adoptés pour interroger par exemple les tables d'une base. Comme ils englobent des requêtes SQL, ils n'ont pas été reportés dans ce document car il n'est pas obligatoire de les employer, le SQL étant plus indiqué. Parfois cependant, l'emploi de Java via JDBC peut régler bien des problèmes lors de l'enregistrement de données pour certains types tels que le *CLOBS*. Dans le cas de Oracle, il est préférable d'utiliser une connexion JDBC pour insérer une valeur dans un champs *CLOB*. Dans ce cas, la syntaxe a été reportée.

Les SGBD supportés par *SiardSuite* proposent des outils pour interagir avec le moteur de base de données. Ces interfaces graphiques sont assez intuitives et permettent d'administrer facilement un projet. Il est à noter que dans Access 97, le moteur Jet n'étant pas nativement compatible avec SQL, il est impossible de paramétrer des requêtes dans ce langage. Dans la version 2007, ce détail technique a été résolu. Il est également possible de taper les requêtes SQL dans des lignes de commandes directement dans Oracle et SqlServer. Dans les deux cas, le résultat de l'opération en ligne de commande est exactement le même que celui issu des interfaces graphiques.

Les différents morceaux de codes dans les divers langages sont souvent issus d'exemples glanés dans la documentation des SGBD, dans des ouvrages spécialisés ou sur des forum informatiques en ligne. Les sources sont toujours mentionnées même si le résultat ne correspond jamais au code original et se retrouve toujours adapté. L'absence de mention de source signifie que la requête a été créée directement sans autre support.

Le schéma conceptuel de départ et la définition des contraintes courantes

[1] PL/SQL est le langage de requête employé pour l'administration des bases de données dans les SGBD Oracle.

[2] Transact SQL est employé pour l'administration des bases de données dans les SGBD SqlServer.

[3] VBA : Virtual Basic for Application. Il s'agit d'un langage propriétaire développé par Microsoft pour programmer directement dans les logiciels publiés par l'éditeur qui le supportent.

Le schéma conceptuel de la base de données de test est largement inspiré d'un exemple fourni dans l'ouvrage Bases de données[4] de Jean-Luc Hainaut. Il représente les besoins primaires rencontrés dans des opérations commerciales classiques et se compose de quatre tables. Le nom de ces tables et de leurs colonnes est assez compréhensible et ne sera pas plus expliqué. De toute évidence, il n'est pas primordiale de comprendre le contenu intellectuel de cette base de protocole puisqu'il sert juste de motif à nos tests.
Sous Access 97, il est impossible d'entrer ces requêtes directement et la création de la base ne peut s'effectuer qu'en utilisant l'interface graphique. Toutefois, on veillé à « transcrire » le résultat de cette syntaxe dans la base de données. Pour les autres SGBD, il est possible a priori d'entrer le code directement pour obtenir la base PROTOCOLE.

Requête SQL de création de la base de données :

```
create database PROTOCOLE ;

create table CLIENT (NCLI         char(10)       not null,
                     NOM          char(32)       not null,
                     ADRESSE      char(60)       not null,
                     LOCALITE     char(30) ,
                     CAT          char(2),
                     COMPTE       decimal(9,2)    unique    not
null,
         primary key (NCLI) ) ;

create table PRODUIT    (NPRO     char(15)       not null,
                         LIBELLE  char(60)       not null,
                         PRIX     decimal(6)     not null,
                         QSTOCK   decimal(8)     not null,
         primary key (NPRO) ) ;

create table COMMANDE   (NCOM     char(12)       not null,
                          NCLI    char(10)       not null,
                          DATECOM date           not null,
         primary key (NCOM),
         foreign key (NCLI) references CLIENT) ;

create table DETAIL     (NCOM     char(12)       not null,
                         NPRO     char(15)       not null,
                         QCOM     decimal(8)     not null,
         primary key (NCOM, NPRO),
         foreign key (NCOM) references COMMANDE,
         foreign key (NPRO) references PRODUIT) ;
```

Les éléments importants à retenir de ces premières lignes sont les suivants :

- Une base de données PROTOCOLE est préalablement créée.
- Les contraintes d'intégrité ont été indiquées pour toutes les tables (*primary key*).
- Des contraintes référentielles ont été définies entre certaines tables (*foreign key*).

[4] Jean-Luc Hainaut, *Bases de données, concepts, utilisation et développement,* collection Sicences Sup, Dunod, Paris, 2009. P.158

- Une contrainte d'unicité est présente pour une colonne de la table CLIENT (unique ou *candidate key*).
- La valeur *nullable* est différente selon les colonnes (not null ou non défini). LOCALITE et CAT peuvent ne pas être renseignés.
- Les types de données sont courants. Le test sur l'ensemble des types supportés par les SGBD sera donné ultérieurement.
- D'autre part, aucun schéma n'est défini ce qui signifie que l'on emploie celui par défaut (le dba[5] généralement désigné sous Oracle et la valeur « dbo »[6] sous SqlServer).

Autres contraintes définies au niveau de la table

Le schéma conceptuel de base définit des contraintes de gestion courantes normalement présentes dans toute base de données relationnelles. Pour répondre à des contraintes plus élaborées, il peut être nécessaire d'avoir recours à d'autres types de fonctionnalités normalisées dans SQL :1999 et implémentées dans les SGBD. Il s'agit de l'emploi des prédicats CHECK et des déclencheurs (*triggers* en anglais) :

a) Les contraintes de validation (CHECK) sont indiquées pour vérifier les données avant leurs enregistrements. Elles sont exprimées en SQL et sont supportées par tous les SGBD. Dans Access, la commande qui correspond à l'expression d'une contrainte CHECK est la clause « vérifier ».

b) Les déclencheurs sont utilisés pour lancer des actions avant ou après un enregistrement dans une colonne de la base de données. Ils peuvent s'exprimer en SQL depuis 1999 puisque c'est dans cette version de la norme qu'ils ont été formalisés (pour autant leur emploi date d'avant cette année). La syntaxe est différente entre Oracle et SqlServer. Access 97 et 2007 ne permettent pas de définir de *triggers*. Les macros pourraient éventuellement leur être assimilées , mais *SiardSuite* ne les archive pas dans la mesure où elles sont écrites en VBA. Dans les formulaires, il est possible de régler des évènements avant et après l'insertion. La définition de ce genre de contrainte peut s'assimiler à une sorte de déclencheur même si les formulaires ne sont pas archivés.

A) Définition d'une contrainte CHECK dans Oracle et SqlServer (inspirée de l'ouvrage de Jean-Luc Hainaut):

```
Alter table CLIENT add constraint CHECKCLIENT check  ( CAT is
null or CAT in ('B1', 'B2')) ;
```

Exemple de vérification :

```
insert        into        CLIENT        values('01','Octave',
'Retriver','Lugdunum','B2','5100');
```

- Il est impossible d'enregistrer une autre valeur que B1 ou B2 dans la colonne CAT. Il est en revanche permis de laisser le champs vide, telle que défini dans la valeur *nullable.*

B) 1- Exemple[7] d'une requête de création d'un déclencheur dans Oracle :

[5] Dba : DataBase administrator. Par défaut, il s'agit de SYS dans Oracle.

[6] Dbo : DataBase Object.

[7] Le présent code est inspiré de l'exemple de la page suivante : http://laltruiste.fr/

```
create table DUPLICATA  (COMPTE   decimal(9,2)    unique    not
null,
  NOM         char(32));
```

- Il est nécessaire de créer une nouvelle table dont les colonnes reprennent les spécifications de certaines colonnes de PRODUIT.

```
create trigger declencheur_insertion
    after insert on CLIENT
    for each row
when (new.COMPTE <= 100)
    begin
      insert into DUPLICATA values(:new.COMPTE, :new.NOM);
  end;
```

Exemple de déclenchement :

```
insert      into      CLIENT      values      ('02','Pompée',
'Leonberg','Condate','B2','100');
```

- Le *trigger* « declencheur_insertion » se déclenche après l'insertion d'une ligne dans CLIENT. Lorsque l'objet COMPTE est inférieur ou égal à 100, sa valeur et celle de NOM dans CLIENT sont insérées dans la table DUPLICATA.

2- Exemple d'une requête de création d'un déclencheur dans SqlServer :

```
create trigger avertissement
on CLIENT
for insert, update, delete
AS
  RAISERROR ('Attention une modification a été effectuée dans
la table CLIENT',16, 10)
GO
```

- Le *trigger* « avertissement » se déclenche lorsqu'une insertion, une mise à jour ou une suppression sont effectuées sur une ligne de la table CLIENT. Il affiche la phrase d'avertissement donnée entre cotes.

L'emploi des procédures stockées

Bien que les *triggers* permettent de déterminer un certain nombre de contraintes supplémentaires par rapport à celles du schéma de base, ils ne sont parfois pas suffisants. Les SGBD proposent alors de déclarer des procédures stockées (*routines* en anglais) qui peuvent appliquer de nouvelles actions ou de nouvelles contraintes. Leur syntaxe est différente selon qu'elles soient définies dans PL/SQL ou dans Transact SQL. Dans Access, il existe des modules qui peuvent s'assimiler à des procédures stockées mais ils sont codés en VBA. Même si le format SIARD ne prend pas en compte ce langage, à la différence d'instructions en PL/SQL ou en Transact SQL, un exemple a été reporté pour montrer que *SiardSuite* ne les archive pas.

1- Exemple d'une requête[8] de création de procédure stockée dans Oracle :

```
create table FOURNISSEUR  ( NFOUR      number,
                  NOMFOUR   varchar2(25)) ;
```

- Pour illustrer l'usage d'une procédure dans Oracle, il est nécessaire de créer une nouvelle table nommée FOURNISSEUR par commodité et pour rester en adéquation avec le sujet de la base.

```
CREATE OR REPLACE PROCEDURE ins_fournisseur(ID number, Nom in
varchar2) IS
    nbFour INTEGER;
    DejaPresent EXCEPTION;

BEGIN
       SELECT  COUNT(*)  INTO  nbFour  FROM  FOURNISSEUR  WHERE
NOMFOUR=Nom;
    IF nbFour > 0 THEN
        RAISE dejaPresent;
    ELSE
        INSERT INTO FOURNISSEUR VALUES (ID, Nom);
    END  IF;

EXCEPTION
    WHEN DejaPresent THEN Dbms_Output.put_line('Insertion non
réalisée : Ce fournisseur existe déjà.');
END;
```

Exemple de déclenchement :

```
begin
ins_fournisseur ('01','GordonFreeman');
end;
/
```

- La procédure « ins_fournisseur » vérifie avant l'enregistrement d'un nouveau nom de fournisseur si ce dernier n'existe pas déjà dans la colonne. Si c'est le cas, il renvoie un message d'erreur défini entre cotes. Il est à noter que SIARD considère ID et Nom comme des paramètres et qu'ils sont archivés comme tels.

2- Exemple de deux requêtes[9] de création de procédure stockée dans SqlServer :

a- Procédure Ajout_Enregistrement

[8] Exemple adapté de la page suivante : http://www.rcweb.fr/bd/plsql.html.

[9] Le premier exemple est inspiré de celui de la page suivante : http://laltruiste.fr/. Quant au second, il est adapté de celui fourni à la page suivante : http://sqlpro.developpez.com/cours/sqlserver/transactsql/#L4.1.

```
create procedure Ajout_Enregistrement
        @proc_NCOM CHAR(10),
        @proc_NCLI VARCHAR(20),
        @proc_DATECOM VARCHAR(20)
 AS
      insert into COMMANDE
                    (NCOM, NCLI, DATECOM)
       values (@proc_NCOM, @proc_NCLI, @proc_DATECOM)
GO
```

Exemple de déclenchement de la *routine :*

```
execute Ajout_Enregistrement '04', '01', '02/10/1986'
```

- La procédure stockée « Ajout_Enregistrement » définit trois paramètres (symbolisés par un @ en Transact Sql) qui correspondent aux colonnes de la table COMMANDE. Si l'administrateur de la base de données décide de lancer cette *routine,* il pourra insérer directement les valeurs qu'il souhaite enregistrer dans cette table. Il est évident que cette routine n'est pas forcément très intéressante dans une finalité de gestion mais telle n'est pas son but. Elle a été adjointe pour étudier comment le processus l'archive.

b- Procédure LISTE_CLENT

```
create procedure LISTE_CLIENT
@NCLI_DEBUT INTEGER, @NCLI_FIN INTEGER AS select
'-- début de liste client --'
UNION ALL
select NOM from CLIENT
where NCLI between @NCLI_DEBUT and @NCLI_FIN
UNION ALL select
'-- fin de liste client --'
```

Exemple de déclenchement de la procédure :

```
execute  LISTE_CLIENT @NCLI_DEBUT = 01, @NCLI_fin=02;
```

- La *routine* « LISTE_CLIENT » permet d'afficher les valeurs de NOM dans CLIENT lorsque l'identifiant de la table NCLI est compris entre le paramètre de début et celui de fin.

3- <u>Exemple d'une requête de création d'un module et d'une fonction dans Access (en VBA) :</u>

Module :

```
Sub TestProcedure()
    MsgBox "Les procédures sont mes amies"
End Sub
```

Module de classe :

```
Sub AppelProcedure()
    TestProcedure
End Sub
```

- La présente procédure affiche un message lorsque l'administrateur la déclenche. Cet exemple n'aurait aucun intérêt dans la réalité mais il n'a pas été créé dans ce but.

```
Function LongueurDuMot(test) As Integer
LongueurDuMot = Len("LongueurDuMot")
End Function
```

- La fonction « LongueurDuMot » calcule la longueur d'une chaîne de caractères lorsqu'elle est déclenchée. Elle n'aurait aucun intérêt dans la réalité mais fonctionne bien dans Access.

L'usage des tables virtuelles : les Vues

Certaines données contenues dans les bases de données peuvent être jugées sensibles, tant du point de vue de leur contenu que de celui du fonctionnement du système. Dans ce cas, l'administrateur d'une base peut décider de créer des tables virtuelles, couramment nommées vues. Les SGBD permettent tous de les créer et de les utiliser. Elles présentent également l'avantage de regrouper des informations au sein d'une entité définie par l'administrateur. Il faut considérer les vues comme des sortes d'images ou de copies des tables. Elles sont généralement définies pour des utilisateurs donnés qui auront des droits limités, le but étant que les tables physiques ne soient pas endommagées. Dans Access, les formulaires peuvent assez bien leur être assimilés et le format SIARD les considère de cette manière. Il est impossible de les définir en SQL dans Access. Il faut donc le faire dans l'interface graphique.

1- Définition d'une requête[10] de création de vue dans Oracle et SqlServer :

```
create view CLI(NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE) as
select * from CLIENT where CAT is null or CAT in ('B1','B2')
with check option;
```

- La vue « CLI » permet d'accéder au contenu des colonnes de la table CLIENT. La clause *select* * est employée pour interroger toutes les lignes de ces colonnes. D'autre part, un prédicat CHECK a été ajouté dans la syntaxe pour étudier comment il est archivé lorsqu'il est inséré dans une vue par rapport à son traitement lors de sa désignation directe dans le SGBD.

La gestion des droits d'utilisation : le contrôle d'accès

L'introduction des vues a ouvert une autre question importante de la gestion des bases de données, celle des droits d'utilisation. Il existe toujours par défaut un administrateur de base défini par le SGBD ('SYS' qui possède les privilèges 'dba' dans Oracle est presque l'équivalent de 'Administrateur' dans Access). Il incombe aux utilisateurs de ces produits de

[10] Jean-Luc Hainaut, *Bases de données, concepts, utilisation et développement,* collection Sicences Sup, Dunod, Paris, 2009.

définir des accès au serveur, différents selon les besoins de chacun. Généralement un nouveau compte de connexion est créé et possède des droits étendus d'administration. Pour autant, il serait inapproprié en terme de sécurité de confier à tous les utilisateurs, ce genre de droits. C'est pourquoi les SGBD proposent de définir plusieurs types d'accès qui répondent en théorie à tous les cas que la pratique impose. Globalement, peu de différences peuvent être relevées en fonctionnement courant entre Oracle et SqlServer, même si techniquement les deux systèmes procèdent de manière propre. Il faut simplement noter que SqlServer différencie le *login* de connexion du *user* ce que Oracle ne semble pas faire. Hormis cette différence, les requêtes SQL sont assez semblables. En ce qui concerne Access, il est impossible de définir cette question autrement qu'à travers l'interface graphique. De toute évidence, ce genre de fonctionnalités est beaucoup moins développé que dans les deux autres produits.

Trois niveaux de droits peuvent être définis : les utilisateurs, les privilèges et les rôles

1- Les utilisateurs

Le contrôle d'accès primaire se matérialise par la définition d'utilisateurs possédant plus ou moins de droits. L'utilisateur peut accéder aux objets de la base, les modifier, insérer de nouvelles données ou autre type d'actions selon les droits qui lui sont propres.

Dans SqlServer le *login* de connexion permet à un individu de se connecter au serveur de base de données.

```
CREATE LOGIN Gman WITH PASSWORD = 'BlackMesa'
```

Dans un second temps, il lui faut cependant disposer d'un objet utilisateur pour accéder réellement à la base :

```
use PROTOCOLE
GO
create user Xaero for login Gman
```

Dans Oracle, la requête de création d'un utilisateur est suffisante pour permettre de se connecter au SGBD :

```
create user Xaero identified by MotDePass;
```

- Pour autant la création du *user* Xaero ne suffit-elle encore pas à manipuler la base de données PROTOCOLE. Les droits des utilisateurs définis par défaut ne permettent aucune action d'aucune sorte sur la base de données.

2- Les privilèges

Il s'agit d'une autorisation donnée à un utilisateur d'effectuer une action sur un objet de la base de données. Il convient donc de spécifier une opération à effectuer sur des objets à déterminer. Dans les requêtes SQL sont contenus à la fois l'utilisateur qui dispense les droits, ainsi que le ou les utilisateurs à qui ils sont donnés.

```
grant select, update (NCLI, LOCALITE)
    on CLIENT
    to Xaero;
```

- Le présent utilisateur pourrait par exemple sélectionner et mettre à jour les colonnes NCLI et LOCALITE de la table CLIENT.

3- Les rôles

Les bases de données les plus complexes peuvent avoir à gérer un très grand nombre d'utilisateurs différents. Dans certains cas, la notion de privilège n'est plus appropriée et on lui préfère celle de rôle. Ce concept représente une classe intermédiaire d'utilisateurs, c'est-à-dire que l'administrateur peut attribuer un privilège à un utilisateur comme à un rôle. Il est ensuite possible d'accorder plusieurs types de rôles à un seul utilisateur.

```
create role CONSULTANT ;
grant update (ADRESSE) on CLIENT to CONSULTANT ;
grant CONSULTANT to XAERO ;
```

- Le rôle « CONSULTANT » est autorisé à mettre à jour les lignes de la colonne ADRESSE. Par extension, l'utilisateur « Xaero » possède le même droit. En utilisant les différents procédés de contrôle, il devient possible de créer toute sorte de combinaisons de droits d'accès.

Etude systématique des types de données supportés par les SGBD

Le schéma conceptuel de base définit des types de données pour chaque colonne des tables de la base de données. Il est entendu qu'il n'est pas adapté à l'étude complète de tous les types de données supportés par les SGBD. C'est pourquoi une table TYPE dédiée à cet usage, a été adjointe au schéma. Elle contient autant de champs que de types proposées et utilise une politique de nommage simpliste des colonnes. Il existe en effet de grands ensembles de types décrits dans la norme SQL. Ils ont donc été repris et l'on a ajouté des chiffres à leur suite. La table TYPE n'aurait aucune raison d'être dans une base de données en production car aucun besoin ne nécessite d'employer 25 ou 30 types de données différents. Elle est cependant légitime dans le cadre d'une synthèse systématique de cette fonctionnalité. Le décalage observé à maintes reprises entre les recommandations de la norme et leurs adaptions aux SGBD est encore une fois bien visible. Un certain nombre de types ne sont pas compatibles avec SQL et posent donc problème lors de l'archivage.

1- Les types proposés par Oracle

La présente requête n'introduit aucune originalité si ce n'est qu'elle est extrêmement longue. Elle se présente de cette manière :

```
create table TYPES (TYPE_CHARACTER1 char(20), TYPE_CHARACTER2
nchar(20), […] ) ;
```

Voici la liste entière des types de données classés thématiquement :

CHARACTER	NUMERIC	DATETIME	LOB	ROW	OTHER

char(size)	number(p,s)	date	bfile	rowid	xmltype
nchar(size)	numeric(p,s)	timestamp	blob	urowid(size)	uritype
nvarchar2(size)	float	timestamp with time zone	clob		dburitype
varchar2(size)	dec(p,s)	Timestamp with local time	nclob		xsburitype
long	decimal(p,s)	zone			httpuritype
raw	integer	interval year to month			sdo_geometry
long raw	int	interval day to second			sdo_topo_geometry
	smallint				
	real				
	double precision				
	binary_float				
	binary_double				

Il se peut que Oracle interdise le traitement d'une requête aussi longue. Dans ce cas, il peut être pertinent de procéder autrement en créant au préalable la table TYPE et en y insérant les types CHARACTER seulement. Ensuite une syntaxe de ce type permet de contourner la difficulté :

```
Alter table TYPE add (TYPE_NUMERIC1 (number), […] );
```

2- Les types proposées par SqlServer

De la même manière, la requête suivante est sans surprise mais se distingue par sa longueur relativement importante :

```
create table TYPES (type_nbre_ent1 bit, type_nbre_ent2
smallint, […] ) ;
```

Ci-dessous, la liste entière des types de données supportées pas SqlServer :

numeric	datetime	character	lob	other
bit	smalldatetime	char	binary	hierarchyid
tinyint	datetime	varchar	varbinary	xml
smallint	time	nchar	image	geography
int	date	nvarchar	timestamp	geometry
bigint	datetime2	ntext	uniqueidentifier	
decimal	datetimeoffset			
numeric				
smallmoney				
money				
real				

3- Les types de données proposées par Access

Dans la version 2007, il est permis de créer des tables en employant des requêtes SQL, le programme faisant la traduction de ce langage au moteur Jet. En revanche, dans sa version 97, Access ne détermine le type des colonnes qu'à travers l'interface graphique. C'est cette méthode qui a été finalement privilégié dans les deux versions du logiciel.

Le tableau suivant donne tous les types disponibles dans Access :

character	numeric	datetime	lob	other
texte	numérique monétaire numéro auto	date/heure	memo objet OLE liens hypertextes	oui/non

4- L'implantation des types définis par l'utilisateur ou les domaines

Oracle et SqlServer vont plus loin dans leur gestion des types de données puisqu'ils incluent une fonctionnalité de définition propre par l'utilisateur. Ces réalités techniques correspondent à deux noms selon les SGBD mais sont assez semblables. La syntaxe SQL diffère également :

Pour Oracle :

```
create type EtatCivil AS OBJECT
( NOM VARCHAR2(32), PRENOM VARCHAR2(20) ) ;
```

Pour SqlServer :

```
create type dbo.EtatCivil
from char(10) not null
go
```

- Pour que le « type défini par l'utilisateur » soit archivé, il est nécessaire de l'appliquer à une table de la base. Cette démarche a été réalisée lors de la création ultérieure de la table $CLIENT dans le cadre d'utilisation de l'identifiant dans les SGBD.

L'enregistrement dans des colonnes de type *lob*

Bien que la présente étude n'ait pas focalisé son attention sur l'archivage de données primaires en grand nombre, elle a cherché à appréhender le traitement des *lob*. La documentation de SIARD mentionne justement un fonctionnement particulier avec les objets binaires volumineux difficilement insérables dans les fichiers XML qui constituent une archive. Pour insérer ce type de contenu dans les colonnes, les SGBD emploient des syntaxes particulières qui vont être décrites.

1- Insertion d'un Blob dans Oracle[11] :

```
create or replace directory MY_FILES as 'c:\images';
```

- Il est recommandé de définir un dossier sur le poste informatique client qui recueille les fichiers binaires à insérer dans la base

[11] Exemple adapté de la page suivante : http://www.developpez.net/forums/d3265/bases-donnees/oracle/insertion-jpg-blob-oracle-8i/

```
create  or replace procedure insert_img AS
f_lob bfile;
b_lob blob;
begin
insert into TYPES(type_lob2) values (empty_blob() )
return type_lob2 into b_lob;
f_lob := bfilename( 'MY_FILES', 'image.jpg' );
dbms_lob.fileopen(f_lob, dbms_lob.file_readonly);
dbms_lob.loadfromfile(              b_lob,              f_lob,
dbms_lob.getlength(f_lob) );
dbms_lob.fileclose(f_lob);
commit ;
end ;
/
```

- La méthode d'enregistrement fait appel à une procédure stockée classique et s'applique à la table TYPES qui contient la colonne BLOB (type_lob2 en l'occurrence). Le fichier BLOB se nomme 'images.jpg' et se trouve dans le dossier défini préalablement.

```
Exemple de déclenchement de la procédure :
BEGIN
insert_img;
END;
/
```

2- Insertion d'un Blob ou d'un Clob dans SqlServer[12] :

Depuis la version 2005 du SGBD, il est possible d'utiliser une simple requête SQL pour insérer des *lob* dans la base de données en employant la fonction OPENROWSET BULK. La colonne de réception doit être typée *binary*, dans son sens large puisque tous les types de cet ensemble fonctionnent avec cette méthode. La table TYPES convient donc parfaitement puisque le champs « type_bin3 » est par exemple défini en image.

```
insert into TYPES(type_bin3)
select * from
openrowset   (bulk   'C:\images\image.jpg',   single_blob)   as
IMAGE ;
```

- Le même dossier « images » a été utilisé dans cet exemple. Il est important de ne pas oublier le nom de corrélation (as IMAGE) sinon le processus ne fonctionne pas. Cette méthode fonctionne très bien pour tous les types binaires. Un enregistrement dans une colonne typée XML, geometry ou autre peut s'effectuer également ainsi.

Caractères spéciaux et jeux de caractères

Les différents SGBD supportés par *SiardSuite* permettent d'insérer des caractères typographiques spéciaux et de modifier les jeux de caractères. Le comportement du processus

[12] Exemple tiré de la page suivante : http://msdn.microsoft.com/en-us/library/ms175915.aspx

d'archivage à ce sujet est bien décrit dans la documentation. Aussi la présente étude s'est-elle attachée à le vérifier. Les administrateurs de base de données peuvent selon la région du monde où ils se trouvent, utiliser divers jeux de caractères. Il est de même possible que soient insérés des caractères typographiques étrangers à ceux employés habituellement dans la base. En comptant l'insertion des caractères spéciaux et des identifiants dans les bases, on se retrouve facilement avec une grande quantité de cas différents à traiter. L'archive SIARD utilise le code caractère Unicode et l'encodage Utf-8 ce qui oblige le processus à transformer le reste.

1- Caractères 0 à 31 et 128 à 159

Par défaut, les clients des serveurs de base de données des SGBD utilisent le jeu de caractère employé sur le système d'exploitation où ils ont été installés. Les tests n'ayant été effectués que sur le système Microsoft Windows, c'est le code page 1252 de ce dernier qui est donc installé. Il présente un intérêt de taille dans la mesure où les caractères de 128 à 159 dit de contrôle, sont utilisés comme caractères typographiques. *SiardSuite* est pourtant censé les archiver. La syntaxe SQL pour Oracle et SqlServer est identique. Pour Access, on a entré la même chaîne de caractères que dans les deux autres SGBD via l'interface graphique.

Dans la table TYPES, la chaîne de caractère suivante a été insérée dans la colonne TYPE_CHARACTER3 (Oracle) ou type_caract3 (Sql server) : ¦Çüéâäàåçêëèïîì ÄÅÉæÆôöòûùÿÖÜø£Ø×ƒ . Elle correspond à ces fameux caractères 128 à 159.

2- L'usage des caractères spéciaux

Il s'agit de caractères qui posent problème dans le XML et que *SiardSuite* s'attache à transformer pour permettre la validation des fichiers de sortie. La syntaxe SQL employée est quasiment identique dans les SGBD Oracle et SqlServer[13]. Dans Access, les mêmes lignes de contenus ont été insérées dans la nouvelle table créée via l'interface graphique. Elle ne se compose en revanche que de deux colonnes et donc de types de données, « texte » et « memo ».

Oracle :

```
create table CARACTERES (caract1 char(25), caract2 nchar(25),
caract3 varchar2(128), caract4 nvarchar2(128), caract5 clob,
caract6 nclob);
```

SqlServer :

```
use PROTOCOLE create table CARACTERES (caract1 char(25),
caract2 nchar(25), caract3 varchar(128), caract4
nvarchar(128), caract5 text, caract6 ntext);
```

[13] Le type de données varchar et Clob qui correspondent à la même définition technique, sont cependant exprimés différemment.

- Une nouvelle table est créée pour tester spécifiquement le comportement du processus. Les types de colonnes définis contiennent tous des caractères et les deux derniers sont indiqués pour les longues chaînes de caractères.

```
a- insert          into          caracteres          values
   ('Essai','Essai','Essai','Essai','Essai','Essai');
b- insert into caracteres values ('Essai2&<\', 'Essai2&<\',
   'Essai2&<\', 'Essai2&<\', 'Essai2&<\', 'Essai2&<\');
c- insert   into   caracteres   values   ('Essai   3','Essai
   3','Essai 3','Essai 3','Essai 3','Essai 3');
d- insert    into    caracteres    values    ('Essai4&<\>',
   'Essai4&<\>', 'Essai4&<\>', 'Essai4&<\>', 'Essai4&<\>',
   'Essai4&<\>');
```

- Cette succession de requêtes permet de contrôler les caractères spéciaux traités à part par *SiardSuite*. Le a) est un simple essai qui ne pose aucun problème. Le b) insère les caractères spéciaux &<\ . Le c) expérimente le caractère d'espacement. Le d) introduit le > qui pourrait gêner la validation XML.

3- Le support de l'Identifiant

Oracle et SqlServer à la différence de Access permettent de manipuler des caractères réservés du langage SQL dans les enregistrements des données primaires. L'administrateur d'une base de données peut vouloir pour une raison qui lui est propre, placer le caractère $ dans le nom d'une table ou la nommer « type », ce qui est normalement rigoureusement interdit. La syntaxe employée est cependant différente d'un SGBD à l'autre.

Exemple dans Oracle[14] :

```
create table "$CLIENT"
("^First Name"     varchar2(12)        not null,
EtatCivil          EtatCivil ) ;
```

Exemple dans SqlServer[15] :

```
SET QUOTED_IDENTIFIER on
GO
create table "$CLIENT"
("^First Name"     varchar(12)         not null,
EtatCivil          EtatCivil) ;
```

- Une autre table est créée dans une volonté didactique. Il est clair qu'on pourrait modifier le nom d'une table déjà existante. Cependant il parait plus aisé de comprendre directement sur un exemple imaginé pour l'occasion, le fonctionnement du SGBD. De plus les requêtes SQL fournies dans cette synthèse sont le reflet des *dumps* composant les annexes électroniques.
- La colonne EtatCivil insère le TDU défini plus haut pour montrer comment il est archivé par *SiardSuite*.

[14] Exemple adapté de la page suivante : http://www.ispirer.com/doc/sqlways39/Output/SQLWays-1-038.html

[15] Exemple adapté de la page suivante : http://msdn.microsoft.com/en-us/library/aa224033(SQL.80).aspx

Conclusion

La démarche qui a prévalu au cours de l'étude a mélangé les notions en utilisant l'influence des contraintes inhérentes aux SGBD ainsi qu'au modèle relationnel et en éprouvant les dires de la documentation ou les fonctionnalités propres du format SIARD. En définitive, une base de données nommée PROTOCOLE a été créée pour tester techniquement et le plus finement possible le comportement du processus d'archivage des bases de données. Les *dumps* par ailleurs livrés en annexe électronique, auraient pu suffire à vérifier les résultats de cette étude. Pour autant, les problèmes de compatibilités entre les versions des SGBD et la volonté de d'expliciter la démarche intellectuelle ont amené à la rédaction de cette annexe. Elle doit être considérée comme un complément à la description détaillée de chaque étape.