ELEMENTS DE CORRECTION POUR LA FEUILLE DE TD 2 de BDA (séances 2 et 3)

L'objectif est de *simuler* sur Oracle (qui reste, quand même, un SGBD relationnel) la définition de schéma ODL que l'on a produit dans le TD 1. Hypothèses : un client peut avoir plusieurs comptes, mais un compte a un unique possesseur.

```
Class Compte {
attribute integer              leNum ;
attribute enum TypeCompte      {CHEQUE, LIVRET, CODEVI}
                               leType ;
attribute float                leSolde ;
relationship Client            lePossesseur
       inverse                 Client::lesComptes
}

Class Client {
attribute string               leNom ;
attribute string               lAdresse
attrinute string               leTéléphone
attribute integer              leNumSS ;
relationship Set<Compte>       lesComptes
       inverse                 Compte::lePossesseur
}
```

1 Première version simpliste

Ici, on ne s'occupe pas encore de déclarer les clés primaires de tables que l'on va créer sous Oracle, ni les clés étrangères.

```
REM ***** CREATION DE TYPES ****

CREATE TYPE Compte AS OBJECT (
leNum NUMBER(6),
leType VARCHAR(10),
leSolde NUMBER(10,2) ,
lePossesseur NUMBER(6)
) ;


CREATE TYPE SetComptes AS TABLE OF Compte ;

REM *** NB : implantation du type set de ODL par une table, comme à p. 80
du cours ***


CREATE TYPE CLIENT AS OBJECT (
leNumSS NUMBER(6),
leNom   VARCHAR(25),
lAdresse VARCHAR(50),
leTelephone VARCHAR(20),
lesComptes SetComptes
);
```

NB : ici, Compte en un type, SetComptes est le type d'un ensemble d'objets qui sont de type comptes et lesComptes est un ATTRIBUT (pour le type CLIENT), et le type de cet attribut est SetComptes.

```
REM *** CREATION DE TABLES  ***


CREATE TABLE Comptes OF Compte ;
REM Attention au mot cle « OF » (voir cours) : on  crée ici une table
REM dont chaque ligne est un objet de type Compte, qui a donc les
REM attributs  leNum, leType etc.

CREATE TABLE Clients OF Client
NESTED TABLE lesComptes STORE AS LesComptesTables;
```

NB : Attention, ici on a crée une table, Clients, où chaque ligne est de type Client, et l'attribut lesComptes de client a le type SetComptes, qui est une table (représentant un ensemble). Donc, *de facto*, on a créé une table qui contient une table imbriquée (NESTED). Mais, comme expliqué en cours, cela est juste une façon de voir de l'utilisateur. En réalité, où l'utilisateur voit, par exemple :

R :

A	B	C
a1	b1	c1
		c2
a2	b1	c3
		c4

Oracle stocke ainsi :
Une table R' avec les attributs A et B (A est toujours la clé) et la table « plate » :

#A	C
a1	c1
a1	c2
a2	c3
a3	c4

Et c'est à cette dernière table que l'on donne un nom avec « STORE AS Mon-Nom-Préféré » (voir la dernière requête que l'on vient d'écrire). Cette dernière table pourra donc être consultée.

Commentaires sur certaines des requêtes de l'énoncé.

```
SELECT * FROM Clients;
```

NB : Ne fonctionne pas, et c'est normal, car en réalité la relation Clients est sauvée sur 2 tables, par Oracle. Tandis que :

```
SELECT * FROM LESCLIENTS;
```

fonctionnera, *après* avoir crée la vue LESCLIENTS comme c'est fait dans l'énoncé.

L'autre vue de l'énoncé permet d'observer la table imbriquée pour les clients :

```
CREATE VIEW LESCOMPTESCLIENTS
SELECT lc.*, c.leNom
FROM Clients c, table(c.lesComptes) lc;
```

```
REM **** INSERTIONS ***

REM D'abord dans Clients, mais avec un ensemble de comptes vide ou nul;
REM NB: le constructeur des objets de type Setcomptes s'appelle Setcomptes,
comme (pour un cas analogue) dans le cours à p.67

INSERT INTO Clients
VALUES(100,'Pierre','10 rue de la Gare','01...',SetComptes()) ;

INSERT INTO Clients
VALUES(200,'Paul','10 rue du Parc','06...',SetComptes()) ;


INSERT INTO Clients
VALUES(300,'Alain','10 rue du Lac','09...',Null);

INSERT INTO Clients
VALUES(400,'Alain OK','10 rue du Lac','09...',SetComptes());

REM In n'a pas utilisé le constructeur SetComptes pour le Alain qui
REM n'est pas OK !!


REM NB: insertion d'une ligne à la fois !

INSERT INTO Comptes
VALUES ( 1, 'CHEQUE', 1000, 100) ;

INSERT INTO Comptes
VALUES ( 2,'CODEVI', 2000, 100 );
```

```
INSERT INTO Comptes
VALUES ( 3,'CHEQUE', 10000, 200 );

INSERT INTO Comptes
VALUES ( 4,'EPARGNE', 9000, 300 );

INSERT INTO Comptes
VALUES ( 5,'LIVRET', 4000, 400 );
```

Attention, même si on a dit, ci-dessous, que le client 100 a les comptes 1 et 2, on l'a dit dans la table Comptes, mais on n'a pas renseigné la table Clients, qui contient une table imbriquée ! On le fait la :

```
REM REMPLISSAGE DES COMPTES DANS LA TABLE DES COMPTES DES CLIENTS

REM Il existe deux syntaxes pour l'insertion dans une table REM REM
imbriquée :
REM INSERT INTO THE (...) (...) ;
REM INSERT INTO TABLE (...) VALUES (...) ;


INSERT INTO
THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 100)
      (SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 100)
```

Ce qu'il faut comprendre :

a) ci-dessus, on insère où ?

On insère dans LE (seul) résultat (THE) qui est le résultat de la sous-requête

```
SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 100
```

c'est-à-dire dans la valeur de lesComptes du client 100 (qui actuellement est un ensemble vide).

b) On y insère quoi ? Les résultats de la sous-requête

```
(SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 100)
```

c'est à dire le plusieurs comptes que nous avons inséré dans la table Comptes pour le client 100.

```
INSERT INTO
      THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 200)
      (SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 200) ;

INSERT INTO
      THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 300)
      (SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 300) ;


INSERT INTO
      THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 400)
      (SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 400) ;
```

```
REM Essai de ré-insertion des mêmes comptes pour le client 400 (Alain OK)
INSERT INTO
      THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 400)
      (SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 400) ;

REM Les comptes ont été réinsérés ! Création de doublons, comme l'on
Voit en exécutant :

SELECT * FROM LESCOMPTESCLIENTS ;

REM Destruction des comptes dans la table imbriquée pour le client 400
(Alain OK)

DELETE FROM THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 400)
lc WHERE lc.lePossesseur = 400 ;
```

On vient d'effacer toute information sur Alain OK, alors on recommence l'insertion pour le client 400 (Alain OK) :

```
INSERT INTO
      THE (SELECT c.lesComptes FROM Clients c WHERE c.leNumSS = 400)
      (SELECT cp.* FROM Comptes cp WHERE cp.lePossesseur = 400) ;
SELECT * FROM LESCOMPTESCLIENTS ;

REM Requete pour voir les comptes de Pierre seulement
SELECT * FROM LESCOMPTESCLIENTS lc WHERE lc.leNom = 'Pierre' ;
```

2. Contraintes d'intégrités simples.

On n'a pas encore déclaré ni de clé primaires ni de clé étrangères, pour les tables ! Et on n'a pas non plus déclaré quels attributs de nos tables doivent être forcement non-nuls.
On ajoute les contraintes de clé primaires et valeurs non nulles ci-dessous, et on ajoutera les clés étrangères après.
Mais, d'abord, une explication préliminaire de la terminologie utilisée ci-dessous pour exprimer des propriétés que ce corrigé a choisies pour les contraintes.

Terminologie

NOT DEFERRABLE : La contrainte opère de que l'ordre SQL qui la met en cause est exécutée. Elle ne pourra jamais être déferrée (retardée, voir ci-dessous). C'est la propriété par défaut, dès que l'on déclare une contrainte.

DEFERRABLE : La vérification de la contrainte est retardée : elle sera effectuée au moment du COMMIT, qui valide l'ordre SQL la mettant en cause, plutôt que dès que cet ordre est formulé.

INITIALLY IMMEDIATE : La contrainte sera appliquée immédiatement, à l'application de la transaction. Elle est vérifiée au cours de la transaction pour chaque ordre SQL qui l'appelle.

ENABLE : Assure que tous les données qui seront entrées (dans le future) seront conformes à la contrainte (sinon, ils seront rejetés).

NOVALIDATE : des données déjà existantes peuvent ne pas être conformes à la contrainte.

```
ALTER TABLE Comptes
      ADD CONSTRAINT comptes_pk                        PRIMARY KEY ( leNum )
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;

REM On déclare un clé primaire et on donne notre nom à cette contrainte.
REM  On aurait pu aussi bien l'appeler «ma_cle_prim_pour_comptes », à la
REM place de « comptes-pk » !
```

Encore de la *terminologie* :

CHECK : Les contraintes CHECK assurent l'intégrité de domaine en limitant les valeurs acceptées par une colonne. Dans la déclaration immédiatement ci-dessous, on impose de contrôler que la valeur entrée pour l'attribut leType ne soit pas nul (s'il l'est, il faut rejeter l'insertion comme illégale).

Toutes les déclarations suivantes imposent que certains attributs ne puissent pas avoir des valeurs nulles. A nouveau, « compte_type_nn » etc, sont nos noms des contraintes, choisis par nous.

```
ALTER TABLE Comptes
      ADD CONSTRAINT comptes_type_nn     CHECK       (leType IS NOT NULL)
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;

ALTER TABLE Comptes
      ADD CONSTRAINT comptes_solde_nn    CHECK       (leSolde IS NOT NULL)
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;
```

```
ALTER TABLE Comptes
   ADD CONSTRAINT comptes_possesseur_nn  CHECK (lePossesseur IS NOT NULL)
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;


ALTER TABLE Clients
      ADD CONSTRAINT clients_pk                       PRIMARY KEY
      ( leNumSS )
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;

ALTER TABLE Clients
  ADD CONSTRAINT clients_lenom_nn   CHECK       (leNom IS NOT NULL)
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;
ALTER TABLE Clients
  ADD CONSTRAINT clients_lescomptes_nn CHECK   (lesComptes IS NOT NULL)
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;
```

2. Contraintes d'intégrités avancées (les clés étrangères).

1) Déclaration de clé étrangère

On n'a pas encore déclaré de clé étrangères, pour les tables ! On ajoute ces contraintes ci-dessous.
Mais, d'abord, à nouveau, une explication préliminaire de la terminologie utilisée ci-dessous, pour faire référence à des propriétés que ce corrigé a choisies pour les contraintes.

Encore de la *terminologie* :

DELETE CASCADE : Si A est la clé primaire d'une table R, et la clé étrangère d'une table S, dès qu'on efface (DELETE) une ligne de R ayant une certaine valeur *v* pour A, toutes les lignes de S qui contenaient *v* pour A seront effacées aussi (effet « boule de neige »).
Ci-dessous, REFERENCES Clients(leNumSS) dit que la clé étrangère de Comptes, qui est lePossesseur, n'est rien d'autre que la clé primaire de Clients, c'est-à-dire leNumSS.

```
ALTER TABLE Comptes
 ADD CONSTRAINT comptes_possesseur_fk          FOREIGN  KEY (lePossesseur)
      REFERENCES  Clients(leNumSS)             ON DELETE CASCADE
      NOT DEFERRABLE
      INITIALLY IMMEDIATE
      ENABLE NOVALIDATE ;
```

2) Discutez de la contrainte d'intégrité référentielle apportée par la présence de la clé étrangère…

Contrainte d'intégrité apportée par la clé étrangère :

- toute insertion de tuples dans la table *Comptes* ne pourra être effective que si la clé étrangère réfère à un tuple de clé primaire identique dans la table *Clients*.

Si la table Client ne possède pas de tuple avec la même clé, le SGBD annulera l'exécution de l'instruction SQL (*rollback*).

Contrainte d'intégrité apportée par ON DELETE CASCADE :

- toute destruction de tuples dans la table *Clients* entrainera automatiquement la destruction en cascade des tuples correspondants dans la table *Comptes*.

Ignorer la question 3 qui suit dans l'énoncé et que nous n'avons pas traité en TD.