Base de Données : Compte-rendu du TP2
Application "Zoo"

Line POUVARET, Mickaël TURNEL

2015-2016

5 - Transactions & JDBC

Etablissement de la connexion

```
1  /* Enregistrement du driver Oracle */
2  System.out.print("Loading␣Oracle␣driver...␣");
3  DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
4  try {
5          Class.forName("oracle.jdbc.OracleDriver").newInstance();
6  } catch (InstantiationException ex) {
7          Logger.getLogger(Banque.class.getName()).log(Level.SEVERE, null, ex);
8  } catch (IllegalAccessException ex) {
9          Logger.getLogger(Banque.class.getName()).log(Level.SEVERE, null, ex);
10 } catch (ClassNotFoundException ex) {
11         Logger.getLogger(Banque.class.getName()).log(Level.SEVERE, null, ex);
12 }
13 System.out.println("loaded");
14
15 /* Etablissement de la connexion */
16 System.out.print("Connecting␣to␣the␣database...␣");
17 Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
18
19 System.out.println("connected");
20
21 // Desactivation de l'autocommit
22 System.out.println("Autocommit␣disabled");
23 conn.setAutoCommit(false);
24
```

Question 1

1) Afficher la liste des animaux

```
1  /* Fonction d'affichage des animaux */
2  private static void listeAnimaux() throws SQLException {
3              Statement st = conn.createStatement();
4
5              ResultSet rs = st.executeQuery("SELECT␣*␣FROM␣LesAnimaux");
6              System.out.println("Liste␣de␣tous␣les␣animaux");
7
8              afficherRes(rs);
9                   System.out.println();
10         st.close();
11 }
```

```
12
13 /* Fonction d'affichage generale */
14 public static void afficherRes(ResultSet rs){
15         try {
16                 ResultSetMetaData rms = rs.getMetaData();
17                 int col = rms.getColumnCount();
18
19                 for(int i=1; i <= col; i++) {
20                         String nomCol = rms.getColumnName(i);
21                              System.out.print(String.format("%-15s", nomCol));
22                    }
23                 System.out.println();
24
25                 for(int i=1; i <= col; i++) {
26                         System.out.print("---------------");
27                 }
28                 System.out.println();
29
30                 while(rs.next()) {
31                         for(int i=1; i <= col; i++) {
32                                 String type = rms.getColumnClassName(i);
33
34                                 if (type.equals("java.lang.String")) {
35                                         System.out.print(String.format("%-15s"
    , rs.getString(i)));
36                                 } else if (type.equals("java.math.BigDecimal")
    ) {
37                                         System.out.print(String.format("%-15s"
    , String.valueOf(rs.getInt(i))));
38                                 } else if (type.equals("java.sql.Date")) {
39                                         System.out.print(String.format("%-15s"
    , rs.getDate(i)));
40                                 }
41                         }
42                         System.out.println();
43                 }
44         } catch (SQLException e) {
45                 System.out.println("ERREUR : " + e.getMessage());
46         }
47 }
48
```

2) Déplacer un animal de cage

```
1 /* Fonction qui deplace un animal vers une autre cage */
2 private static void deplacerAnimal() throws SQLException {
3          String animal;
4             int cage;
5             System.out.println("Quel animal voulez-vous deplacer ?");
6             animal = LectureClavier.lireChaine();
7             cage = LectureClavier.lireEntier("Vers quel numero de cage voulez-
    vous deplacer "+animal+" ?");
8
9             Statement st = conn.createStatement();
10
11            int nb = st.executeUpdate("UPDATE LesAnimaux SET noCage="+cage+"
    WHERE noma='"+animal+"'");
12            System.out.println(nb + " ligne(s) modifiee(s).");
13
```

```
            st.close();
}
```

3) Ajouter une maladie a un animal

```
/* Fonction qui ajoute une maladie a un animal */
private static void ajouterMaladie() throws SQLException {
            String animal, maladie;
            System.out.println("A quel animal souhaitez-vous ajouter une maladie ?");
            animal = LectureClavier.lireChaine();
            System.out.println("Quelle maladie souhaitez-vous ajouter ?");
            maladie = LectureClavier.lireChaine();

            Statement st = conn.createStatement();

            int nb = st.executeUpdate("INSERT INTO LesMaladies VALUES('"+animal+"', '"+maladie+"')");
            System.out.println(nb + " ligne (s) ajoutee (s) dans LesMaladies.");
            System.out.println();
            st.close();
}

```

4) Valider/Annuler une transaction

```
/* Fonction de validation d'une transaction */
private static void commit() throws SQLException {
            conn.commit();
            System.out.println("Transaction validee");
}

/* Fonction d'annulation d'une transaction */
private static void rollback() throws SQLException {
            conn.rollback();
            System.out.println("Transaction annulee");
}

```

5) Obtenir/Modifier le niveau d'isolation

```
/* Fonction permettant d'obtenir le niveau d'isolation */
private static void getIsolation() throws SQLException {
            int level = conn.getTransactionIsolation();

            System.out.print("Niveau d'isolation : ");
            switch(level){
                case Connection.TRANSACTION_READ_COMMITTED:
                    System.out.println("READ COMMITTED");
                    break;
                case Connection.TRANSACTION_READ_UNCOMMITTED:
                    System.out.println("READ UNCOMMITTED");
                    break;
                case Connection.TRANSACTION_REPEATABLE_READ:
                    System.out.println("REPEATABLE READ");
                    break;
```

```
                case Connection.TRANSACTION_SERIALIZABLE:
                    System.out.println("SERIALIZABLE");
                    break;
                default:
                    break;
            }
}

/* Fonction permettant de modifier le niveau d'isolation */
private static void setIsolation() throws SQLException {
            System.out.println("*** Choisir un niveau d'isolation : ***");
            System.out.println("0 : READ_UNCOMMITTED");
            System.out.println("1 : READ_COMMITTED");
            System.out.println("2 : REPEATABLE_READ");
            System.out.println("3 : SERIALIZABLE");

            int level = LectureClavier.lireEntier("Entrez un entier");
            boolean ok = true;

            switch(level){
                case 0: level=Connection.TRANSACTION_READ_UNCOMMITTED; break;
                case 1: level=Connection.TRANSACTION_READ_COMMITTED; break;
                case 2: level=Connection.TRANSACTION_REPEATABLE_READ; break;
                case 3: level=Connection.TRANSACTION_SERIALIZABLE; break;
                default: System.out.println("Pas le bon numero");ok=false;
    break;

            }
            if(ok)
                conn.setTransactionIsolation(level);
}

```

6 - Gestion des contraintes

Question 2

1) Le calcul du nombre de maladies pour chaque animal doit être automatisé en fonction de l'ajout ou de la suppression des maladies.

```
CREATE or REPLACE TRIGGER MaladiesTrig
AFTER INSERT or DELETE on LesMaladies
FOR EACH ROW
BEGIN
        IF INSERTING THEN UPDATE LesAnimaux SET nb_maladies = nb_maladies+1
    WHERE nomA=:new.nomA;
        ELSIF DELETING THEN UPDATE LesAnimaux SET nb_maladies = nb_maladies-1
    WHERE nomA=:old.nomA;
        END IF;
END;
/
```

2) Des animaux ne peuvent pas être placés dans une cage dont la fonction est incompatible avec ces animaux. On prendra en compte le fait que des animaux peuvent être ajoutés, mais aussi déplacés d'une cage.

```
CREATE OR REPLACE TRIGGER DeplacerCageTrig
BEFORE INSERT OR UPDATE OF noCage on LesAnimaux
FOR EACH ROW
DECLARE
        fct VARCHAR2(20);
BEGIN
        SELECT fonction into fct FROM LesCages WHERE noCage = :new.noCage;
        IF (fct != :new.fonction_cage) THEN
                raise_application_error(-20001, 'cage incompatible');
        END IF;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                raise_application_error(-20002, 'cage inexistante');
END;
/
```

3) Des animaux ne peuvent pas être placés dans une cage non gardée. On prendra en compte le fait que des animaux peuvent être ajoutés, mais aussi déplacés d'une cage.

```
CREATE OR REPLACE TRIGGER CageGardeeTrig
BEFORE INSERT OR UPDATE OF noCage on LesAnimaux
FOR EACH ROW
DECLARE
        nb INTEGER;
BEGIN
        SELECT count(*) into nb FROM LesGardiens WHERE noCage = :new.noCage;
        IF (nb = 0) THEN
                raise_application_error(-20003, 'Cage non gardee');
        END IF;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                raise_application_error(-20002, 'cage inexistante');
END;
/
```

4) Des animaux de type différent ne peuvent pas cohabiter dans une même cage. On prendra en compte le fait que des animaux peuvent être ajoutés, mais aussi déplacés d'une cage.

```
CREATE OR REPLACE TRIGGER CohabiterCageTrig
AFTER INSERT OR UPDATE OF noCage on LesAnimaux
DECLARE
        nb INTEGER;
BEGIN
        SELECT count(*) into nb FROM(
                SELECT noCage, count(DISTINCT type_an)
                FROM LesAnimaux
                GROUP BY noCage
                HAVING COUNT (distinct type_an) > 1
        );
        IF (nb > 0) THEN
                raise_application_error(-20003, 'Des animaux de type
   differents ne peuvent cohabiter');
        END IF;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                raise_application_error(-20002, 'Cage inexistante');
END;
```

```
19 /
```

Jeux de tests pertinents

Tests du premier trigger

```
1 SQL> SELECT * FROM LesAnimaux WHERE nomA='Charly';
```

NOMA	SEXE	TYPE_AN	FONCTION_CAGE	PAYS	ANNAIS	NOCAGE	NB_MALADIES
Charly	male	lion	fauve	Kenya	2010	11	3

```
1 SQL> INSERT INTO LesMaladies VALUES('Charly', 'conjonctivite');
2 1 row created.
3
4 SQL> SELECT * FROM LesAnimaux WHERE nomA='Charly';
```

NOMA	SEXE	TYPE_AN	FONCTION_CAGE	PAYS	ANNAIS	NOCAGE	NB_MALADIES
Charly	male	lion	fauve	Kenya	2010	11	4

On constate qu'à la suite de la création du déclencheur, lorsqu'on veut ajouter une maladie à un animal dans la table LesMaladies, la valeur nb_maladies associée à cet animal dans la table LesAnimaux est bien incrémentée (et décrémentée lors de la suppression d'une maladie).

Tests du deuxième trigger

```
1 SQL> UPDATE LesAnimaux SET noCage=1 WHERE nomA='Charly';
2 UPDATE LesAnimaux SET noCage=1 WHERE nomA='Charly'
3        *
4 ERROR at line 1:
5 ORA-20001: cage incompatible
6 ORA-06512: at "POUVAREL.DEPLACERCAGETRIG", line 6
7 ORA-04088: error during execution of trigger 'POUVAREL.DEPLACERCAGETRIG'
```

Lorsqu'on souhaite déplacer Charly (qui est dans une cage pour fauves) dans la fosse, on constate bien le déclenchement du trigger DeplacerCageTrig.

On ne peut déplacer Charly que vers une cage 'fauve'.

Tests du troisième trigger

```
1 SQL> SELECT * FROM LesGardiens;
```

noCage	nomE
11	Lachaize
12	Spinnard
12	Labbe
11	Spinnard
11	Labbe
1	Lachaize
3	Lachaize
12	Lachaize

```
1 SQL> INSERT INTO LesAnimaux VALUES('Polochon', 'male', 'poisson', 'aquarium', '
     France', 2014, 2, 0);
2
3 INSERT INTO LesAnimaux VALUES('Polochon', 'male', 'poisson', 'aquarium', 'France',
     2014, 2, 0)
4             *
5 ERROR at line 1:
```

```
ORA-20003: Cage non gardee
ORA-06512: at "POUVAREL.CAGEGARDEETRIG", line 6
ORA-04088: error during execution of trigger 'POUVAREL.CAGEGARDEETRIG'
```

Ici, on a vérifié si la cage 2 (aquarium) était gardée ou non. On trouve bien que le nombre de gardiens qui la gardent est égale à 0.

Donc, quand on veut insérer un nouvel animal dans la table LesAnimaux qui a ce numéro de cage, on déclenche le trigger CageGardeeTrig qui annonce une erreur.

Tests du quatrième trigger

```
SQL> SELECT * FROM LesAnimaux;
```

NOMA	SEXE	TYPE_AN	FONCTION_CAGE	PAYS	ANNAIS	NOCAGE	NB_MALADIES
Charly	male	lion	fauve	Kenya	2010	11	3
Arthur	male	ours	fosse	France	2000	1	0
Chloe	femelle	pie	petits oiseaux	France	2011	3	1
Milou	male	leopard	fauve	France	2013	11	1
Tintin	male	leopard	fauve	France	2013	11	0
Charlotte	femelle	lion	fauve	Kenya	2012	12	0

```
SQL> UPDATE LesAnimaux SET noCage=12 WHERE nomA='Milou';
UPDATE LesAnimaux SET noCage=12 WHERE nomA='Milou'
       *
ERROR at line 1:
ORA-20003: Des animaux de type differents ne peuvent cohabiter
ORA-06512: at "POUVAREL.COHABITERCAGETRIG", line 11
ORA-04088: error during execution of trigger 'POUVAREL.COHABITERCAGETRIG'
```

On a voulu déplacer Milou de cage (qui est un fauve MAIS un leopard) vers la cage de Charly (qui est un fauve MAIS un lion).

Notre requête déclenche bien le trigger CohabiterCageTrig annonçant une erreur avec comme message "Des animaux de type différents ne peuvent cohabiter".

Pour chaque les trois derniers triggers, si la modification ou l'insertion d'un animal avec un noCage qui n'existe pas dans la table LesCages, alors les triggers déclencheront une erreur avec comme message "Cage inexistante".

7 - Contraintes & JDBC

Question 3

Pour permettre à notre programme Java de détecter et gérer les erreurs d'intégrités renvoyés par le SGBD via nos triggers, il suffit juste d'entourer le corps de la boucle while(!exit), là où on va exécuter la fonction demandée par l'utilisateur, par des try et catch.

Nous n'annulons pas d'office les transactions qui violent les contraintes mais nous proposons à l'utilisateur d'annuler sa transaction via le menu.

```
try{
        menu();
        action = LectureClavier.lireEntier("votre choix ?");
        switch(action) {
        case 0 : exit = true; break;
        case 1 : listeAnimaux(); break;
        case 2 : deplacerAnimal(); break;
        case 3 : ajouterMaladie(); break;
        case 4 : commit(); break;
        case 5 : rollback(); break;
```

```
11          case 6 : setIsolation(); break;
12          case 7 : getIsolation(); break;
13          default : System.out.println("=> choix incorrect"); menu();
14          }
15  }
16  catch(SQLException e){
17          System.out.println(e.getMessage());
18          System.out.println("Vous pouvez annuler votre transaction si vous le
        desirer");
19  }
```