TP JDBC

1° Ecrivez un programme Java qui se connecte à la base et affiche tous les numéros de vols (issus de la table `Vol`), en utilisant l'interface `PreparedStatement`.

2° En utilisant la table `Vol`, écrivez un programme Java qui propose les vols pour un tour du monde au départ de Paris avec des escales et des durées d'escale prédéfinies dans la table `Escales`.

Escales :

Numescale	Ville_escale	Duree_escale
1	Moscou	5
2	Singapour	5
3	Sydney	4
4	Tahiti	4
5	Honolulu	4
6	Los Angeles	5
7	New York	4
8	Londres	3

Rappel

La table `Vol` a été créée avec

```
CREATE TABLE Vol (Numvol VARCHAR2(8), Heure_depart DATE,
                  Heure_arrive DATE, Ville_depart VARCHAR2(20),
                  Ville_arrivee VARCHAR2(20))
```

et a le contenu suivant :

```
AF118 08:30 10:57       Paris       Athens
AF212 09:21 14:10       Paris       Moscow
AF178 12:56 14:15       Paris       London
TA215 08:00 10:10       Tunis       Paris
OA005 14:20 17:00       Athens      Paris
SA854 22:00 10:14       Singapore   Athens
AA111 15:45 21:10       Beijing     Singapore
AF218 21:12 09:16       Beijing     Paris
SA012 07:57 11:26       Sydney      Singapore
AF109 07:39 14:10       Tahiti      Sydney
AA517 23:57 07:12       Honolulu    Tokyo
JA014 15:35 19:00       Tokyo       Beijing
AF002 15:52 00:12       Tokyo       Paris
JA115 21:26 10:10       Los Angeles Tokyo
AA015 20:50 07:00       New York    Lima
AA515 07:20 12:38       New York    Los Angeles
AF010 07:53 14:19       Paris       New York
AF012 07:58 20:10       Paris       Los Angeles
AA118 07:15 13:10       New York    Paris
AF001 22:10 12:00       Paris       Tahiti
PA022 10:12 23:55       Lima        Paris
```

Instructions pratiques

Fixez la variable d'environnement `CLASSPATH`

```
monLogin@kirov:~> export CLASSPATH=.:/opt/oracle/product/10.1.0/db_1/jdbc/lib/ojdbc14.jar
```

Pour ne pas avoir à le faire à chaque session, vous pouvez entrer cette ligne dans votre fichier `.profile` (à la racine de votre compte).

Pour compiler le fichier `TpJdbc.java`

```
monLogin@kirov:~> javac TpJdbc.java
```

Pour exécuter le TP

```
monLogin@kirov:~> java TpJdbc monLogin monPassword AF212
```

Voici le fichier `TpJdbc.java`

```
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.Locale;

public class TpJdbc {
    private final static String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver";
    private final static String JDBC_URL = "jdbc:oracle:thin:@kirov:1521:NFA011";

    private Connection conn = null;
    private SimpleDateFormat dateFormat = null;

    public TpJdbc() {
        dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.FRANCE);
    }

    public void initConnection(String login, String password) throws SQLException {
        try {
            Class.forName(JDBC_DRIVER);
            conn = DriverManager.getConnection(JDBC_URL, login, password);
        } catch (ClassNotFoundException e) {
            throw new SQLException("Impossible de trouver le driver JDBC : " + e.getMessage());
        }
    }

    public void closeConnection() {
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            // Exception ignoree
        }
    }

    public void displayFlightInformation(String id) throws SQLException {
    // à remplir!
    }

    public static void main(String[] args) {
        if (args.length != 3) {
            System.err.println("Utilisation : java TpJdbc [login] [password] [numero de vol]");
            System.exit(1);
        }

        TpJdbc myTp = new TpJdbc();

        try {
            myTp.initConnection(args[0], args[1]);
            myTp.displayFlightInformation(args[2]);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            myTp.closeConnection();
        }
    }
}
```

3° Ecrivez un programme Java qui, pour chaque enregistrement de la table `Vol`, insère un nouvel enregistrement pour le vol retour associé : au `Numvol` on ajoute le caractère '`R`' à la fin, les horaires

sont recalculés en considérant un départ 2 heures après l'arrivée (ou sont laissés vides…), la ville de départ et la ville d'arrivée sont échangées.

Par exemple, à partir de l'enregistrement

```
AF118       08:30       10:57       Paris       Athens
```

on insérera l'enregistrement

```
AF118R      12:57       15:24       Athens      Paris
```

Utilisez pour cela un curseur modifiable.

4° Ecrivez un programme Java capable de faire des propositions de tours du monde, prenant en entrée la ville de départ (qui est aussi la destination finale) et deux bornes (supérieure et inférieure) pour le nombre d'escales. Exploitez la fonction récursive PL/SQL suivante (déclarée comme fonction stockée) grâce à l'interface `CallableStatement`.

```
CREATE OR REPLACE FUNCTION
      tourValide(destCourante IN Vol.Ville_depart%TYPE,
                           villeDepart IN Vol.Ville_depart%TYPE,
                           minEscales IN INTEGER, maxEscales IN INTEGER)
            RETURN INTEGER
      IS
            valeurRetour INTEGER := 0;
            quelVol Vol%ROWTYPE;
            CURSOR curseurVol IS SELECT * FROM Vol
                              WHERE (Ville_arrivee = destCourante);
BEGIN
      FOR quelVol IN curseurVol LOOP
            IF (((quelVol.Ville_depart = villeDepart) AND (minEscales>0))
               OR
               ((quelVol.Ville_depart != villeDepart) AND (maxEscales<1)))
            THEN
                  NULL;
            ELSIF (quelVol.Ville_depart = villeDepart) THEN
                  DBMS_OUTPUT.PUT_LINE('De ' || quelVol.Ville_depart ||
                                       ' à ' || destCourante || ' : ' ||
                                        quelVol.Numvol);
                  valeurRetour := 1;
            ELSIF tourValide(quelVol.Ville_depart, villeDepart,
                              minEscales-1, maxEscales-1) THEN
                  DBMS_OUTPUT.PUT_LINE('De ' || quelVol.Ville_depart ||
                                       ' à ' || destCourante || ' : ' ||
                                        quelVol.Numvol);
                  valeurRetour := 1;
            END IF;
      END LOOP;
      RETURN valeurRetour;
EXCEPTION
      WHEN OTHERS THEN RETURN 0;
END tourValide;
```

Afin de pouvoir visualiser les sorties de `DBMS_OUTPUT.PUT_LINE` il faudra introduire dans votre programme Java, avant l'appel de la procédure PL/SQL,

```
DbmsOutput dbmsOutput = new DbmsOutput(connexion);
dbmsOutput.enable(1000000);
```

et, après l'appel de la procédure PL/SQL,

```
dbmsOutput.show();
dbmsOutput.close();
```

après avoir introduit et compilé la classe `DbmsOutput` ci-dessous :

```
import java.sql.*;
class DbmsOutput
{
/*
 * our instance variables. It is always best to
 * use callable or prepared statements and prepare (parse)
 * them once per program execution, rather then one per
 * execution in the program.  The cost of reparsing is
 * very high.  Also -- make sure to use BIND VARIABLES!
 *
 * we use three statments in this class. One to enable
 * dbms_output - equivalent to SET SERVEROUTPUT on in SQL*PLUS.
 * another to disable it -- like SET SERVEROUTPUT OFF.
 * the last is to "dump" or display the results from dbms_output
 * using system.out
 *
 *
 *http://groups.google.com/group/comp.databases.oracle.server/browse_frm/thread/
 * c4b223fe088763c1/6fbc1caeb45133b0?hl=en&lr=&ie=UTF-
 * 8&rnum=14&pr%20ev=/groups%3Fq%3Dauthor:tkyte%2540oracle.com%26hl%3Den%20%26
 * lr%3D%26ie%3DUTF-8%26start%3D10%26sa%3DN#6fbc1caeb45133b0
 */
private CallableStatement enable_stmt;
private CallableStatement disable_stmt;
private CallableStatement show_stmt;
/*
 * our constructor simply prepares the three
 * statements we plan on executing.
 *
 * the statement we prepare for SHOW is a block of
 * code to return a String of dbms_output output.  Normally,
 * you might bind to a PLSQL table type but the jdbc drivers
 * don't support PLSQL table types -- hence we get the output
 * and concatenate it into a string.  We will retrieve at least
 * one line of output -- so we may exceed your MAXBYTES parameter
 * below. If you set MAXBYTES to 10 and the first line is 100
 * bytes long, you will get the 100 bytes.  MAXBYTES will stop us
 * from getting yet another line but it will not chunk up a line.
 *
 */
public DbmsOutput( Connection conn ) throws SQLException
{
    enable_stmt  = conn.prepareCall( "begin dbms_output.enable(:1); end;" );
    disable_stmt = conn.prepareCall( "begin dbms_output.disable; end;" );
    show_stmt = conn.prepareCall(
          "declare " +
          "    l_line varchar2(255); " +
          "    l_done number; " +
          "    l_buffer long; " +
          "begin " +
          "  loop " +
          "    exit when length(l_buffer)+255 > :maxbytes OR l_done = 1; " +
          "    dbms_output.get_line( l_line, l_done ); " +
          "    l_buffer := l_buffer || l_line || chr(10); " +
```

```
          "  end loop; " +
          " :done := l_done; " +
          " :buffer := l_buffer; " +
          "end;" );
}


/*
 * enable simply sets your size and executes
 * the dbms_output.enable call
 *
 */
public void enable( int size ) throws SQLException
{
    enable_stmt.setInt( 1, size );
    enable_stmt.executeUpdate();
}


/*
 * disable only has to execute the dbms_output.disable call
 */
public void disable() throws SQLException
{
    disable_stmt.executeUpdate();
}


/*
 * show does most of the work.  It loops over
 * all of the dbms_output data, fetching it in this
 * case 32,000 bytes at a time (give or take 255 bytes).
 * It will print this output on stdout by default (just
 * reset what System.out is to change or redirect this
 * output).
 */
public void show() throws SQLException
{
int               done = 0;
    show_stmt.registerOutParameter( 2, java.sql.Types.INTEGER );
    show_stmt.registerOutParameter( 3, java.sql.Types.VARCHAR );
    for(;;)
    {
        show_stmt.setInt( 1, 32000 );
        show_stmt.executeUpdate();
        System.out.print( show_stmt.getString(3) );
        if ( (done = show_stmt.getInt(2)) == 1 ) break;
    }
}


/*
 * close closes the callable statements associated with
 * the DbmsOutput class. Call this if you allocate a DbmsOutput
 * statement on the stack and it is going to go out of scope --
 * just as you would with any callable statement, result set
 * and so on.
 */
public void close() throws SQLException
{
    enable_stmt.close();
```

```
    disable_stmt.close();
    show_stmt.close();
}
}
```

5° (Déclencheur Java) Ecrivez et testez une procédure stockée Java permettant de vérifier, pour un numéro de vol (**`Numvol`**), une date de départ (**`Date_dep`**) et un numéro d'avion (**`Numav`**) transmis comme paramètres, si l'avion employé a une capacité au moins égale au nombre de réservations et a comme entrepôt la ville de départ. Si la capacité est insuffisante et/ou l'entrepôt n'est pas la ville de départ, des messages spécifiques sont affichés.

Ecrivez et testez un déclencheur (*trigger*) **`BEFORE INSERT OR UPDATE`** basé sur cette procédure : si les deux conditions sont satisfaites la modification est faite, sinon les messages sont affichés.