Esercitazione TIGA: JDBC – Soluzione proposta

JAVA/JDBC è un *framework* sviluppato da Sun© per facilitare la connessione di applicazioni JAVA con i più svariati database SQL. La connessione con una base di dati richiede, semplicemente, la conoscenza del driver di connessione, solitamente fornito dal produttore del DBMS, e la disponibilità di un account fornito di credenziali valide per l'accesso.

Mysql-server è un noto *DBMS* Open-Source scaricabile gratuitamente all'indirizzo *http://www.mysql.com*. Per utilizzarlo è necessario scaricare il pacchetto di installazione e, una volta installato sul sistema, avviare l'applicazione

```
mysqld-nt.exe
```

presente nella directory `\path\mysql-version\bin`.

Nella stessa directory si possono trovare numerosi file di utilità, incluse varie versioni del server ottimizzate per le prestazioni, o architetture differenti; molto interessate è il file `mysql`, mediante il quale si avvia la console di amministrazione del server, utile per controllare ed amministrare gli utenti del sistema, creare database e tabelle ed assegnare i diritti di accesso ai dati.

Per avviare la console eseguire il seguente comando

```
\path\mysql-version\bin\mysql -u root
```

Una volta in console, con il commando

```
> create database coffeebreack;
```

si crea il database necessario per il completamento dell'esercizio. Riguardo a quest'ultimo, la soluzione che vi accingete a consultare non comporta particolari difficoltà implementative, tuttavia bisogna fare alcune precisazioni.

TheCoffeeBreack richiede la presenza di due tabelle `COFFEES` e `SUPPLIERS` all'interno del database *CoffeeBreack*, è necessario, quindi, crearle ed inizializzarle. Questa operazione, come mostrato in seguito, può essere fatta a *runtime* mediante semplice codice Java, tuttavia, non è la soluzione migliore. Si tratta, infatti, di codice che, generalmente, deve essere eseguito una sola volta, all'inizializzazione del sistema. In genere, questa operazione viene fatta mediante script per la shell dei comandi, come quella fornita tramite il comando `mysql`.

La tabella `SUPPLIERS` richiede la presenza di una colonna `SUP_ID`, che ne costituisce la chiave primaria. Una soluzione più snella dell'esercizio prevede di specificare questa proprietà, direttamente in fase di inizializzazione, indicando anche la direttiva `Auto_Increment` che permette di incrementare automaticamente questo campo all'inserimento di una nuova riga.

Codifica in linguaggio Java:

```
package www.coffeebreack.com;

import java.sql.*;

public class TheCoffeeBreak {

      private static final String  DBMS_DRIVER = "org.gjt.mm.mysql.Driver";
      private static final String  DB_URL      =
                  "jdbc:mysql://localhost:3306/coffeebreak?user=root&password=";
      private static final String  SQL_CREATE_COFFEES=
                              "create table COFFEES " +
                              "(COF_NAME varchar(32), " +
                              "SUP_ID int, " +
                              "PRICE float, " +
                              "SALES int)";
      private static final String  SQL_CREATE_SUPPLIERS    =
                              "create table SUPPLIERS " +
                              "(SUP_ID int Auto_Increment, " +
                              "SUP_NAME varchar(40), " +
                              "STREET varchar(40), " +
                              "CITY varchar(20), " +
```

```
                        "STATE char(2), ZIP char(5), " +
                        "PRIMARY KEY(SUP_ID))";

public static String showCoffees(){

      final String  sql_sel = "select COF_NAME,PRICE,SALES from COFFEES";
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;
      String result = "Coffees:\n\t";

      try {
            Class.forName(DBMS_DRIVER);

      } catch(java.lang.ClassNotFoundException e) {
            e.printStackTrace();
      }
      try {
            con = DriverManager.getConnection(DB_URL);
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql_sel);
            while(rs.next()){
                  result += "Name= " + rs.getString("COF_NAME") +
                        "\n\t\t" +
                        "Price= " + rs.getFloat("PRICE") + "\n\t\t"
                        "Sales= " + + rs.getInt("SALES") + "\n\t";
            }
            stmt.close();
            con.close();

      } catch(SQLException e) {
            e.printStackTrace();
            return null;
      }
      return result;
}

public static String showCoffeesBySupplier(){

      final String  sql_join =
            "select  SUP_NAME,COF_NAME from COFFEES, SUPPLIERS " +
            "where SUPPLIERS.SUP_ID = COFFEES.SUP_ID ";
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;
      String result = "Coffees by Supplier: \n\t";

      try {
            Class.forName(DBMS_DRIVER);

      } catch(java.lang.ClassNotFoundException e) {
            e.printStackTrace();
      }
      try {
            con = DriverManager.getConnection(DB_URL);
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql_join);
            while(rs.next()){
                  result += "Supplier= " + rs.getString("SUP_NAME") +
                        "\n\t\t" +
                        "Coffee= " + rs.getString("COF_NAME") + "\n\t";
            }
            stmt.close();
```

```
            con.close();

      } catch(SQLException e) {
            e.printStackTrace();
            return null;
      }
      return result;
}

public static boolean savePurchaseOrder(int[] sales, String[] coffees){

      final String  sql_update = "update COFFEES set SALES = SALES + ? " +
                              "where COF_NAME like ?";
      Connection con = null;
      PreparedStatement updateSales = null;

      try {
            Class.forName(DBMS_DRIVER);

      } catch(java.lang.ClassNotFoundException e) {
            e.printStackTrace();
      }
      try {
            con = DriverManager.getConnection(DB_URL);

            updateSales = con.prepareStatement(sql_update);
            for(int i=0; i< coffees.length; i++){
                  updateSales.setInt( 1, sales[i]);
                  updateSales.setString( 2, coffees[i]);
                  updateSales.executeUpdate();
            }
            updateSales.close();

            con.close();

      } catch(SQLException e) {
            e.printStackTrace();
            return false;
      }
      return true;
}

public static boolean deleteSupplier(int supplierID){

      final String  sql_delete_1    =
            "delete from COFFEES where SUP_ID = ?";
      final String  sql_delete_2    =
            "delete from SUPPLIERS where SUP_ID = ?";
      Connection con = null;
      PreparedStatement deleteCoffees = null;
      PreparedStatement deleteSupplier = null;

      try {
            Class.forName(DBMS_DRIVER);

      } catch(java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
      }
      try {
            con = DriverManager.getConnection(DB_URL);
            con.setAutoCommit(false);
```

```
                con.setTransactionIsolation(
                      Connection.TRANSACTION_SERIALIZABLE);

                deleteCoffees = con.prepareStatement(sql_delete_1);
                deleteCoffees.setInt(1, supplierID);
                deleteCoffees.executeUpdate();
                deleteCoffees.close();

                deleteSupplier = con.prepareStatement(sql_delete_2);
                deleteSupplier.setInt(1, supplierID);
                deleteSupplier.executeUpdate();
                deleteSupplier.close();

                con.commit();
                con.setAutoCommit(true);
                con.close();

          } catch(SQLException e) {
                e.printStackTrace();
                if(con != null){
                      try{
                            System.out.println("Transaction is being rolled
                                  back");
                            con.rollback();
                      }
                      catch(SQLException ex){
                            ex.printStackTrace();
                      }
                }
                return false;
          }
          return true;
     }

     public static boolean addSupplier(String coffeeName, float coffeePrice,
String[] supplierData){

          final String  sql_insert_1    = "insert into SUPPLIERS " +
                              "values ('', ?, ?, ?, ?, ?)";
          final String  sql_insert_2    = "insert into COFFEES " +
                              "values (?, ?, ?, 0)";
          Connection con = null;
          PreparedStatement insertSupplier = null;
          PreparedStatement insertCoffee = null;
          int supplierID = 0;

          try {
                Class.forName(DBMS_DRIVER);

          } catch(java.lang.ClassNotFoundException e) {
                e.printStackTrace();
          }
          try {
                con = DriverManager.getConnection(DB_URL);
                con.setAutoCommit(false);
                con.setTransactionIsolation(
                      Connection.TRANSACTION_SERIALIZABLE);

                insertSupplier = con.prepareStatement(sql_insert_1);

                for(int i=0; i< supplierData.length; i++){
                      insertSupplier.setString( i+1, supplierData[i]);
                }
```

```
            insertSupplier.executeUpdate();
            ResultSet generatedKey = insertSupplier.getGeneratedKeys()[1];
            generatedKey.next();
            supplierID = generatedKey.getInt(1);
            insertCoffee = con.prepareStatement(sql_insert_2);

            insertCoffee.setString( 1, coffeeName);
            insertCoffee.setInt( 2, supplierID);
            insertCoffee.setFloat( 3, coffeePrice);

            insertCoffee.executeUpdate();

            insertSupplier.close();
            insertCoffee.close();

            con.commit();
            con.setAutoCommit(true);
            con.close();

      } catch(SQLException e) {
            e.printStackTrace();
            if(con != null){
                  try{
                        System.out.println("Transaction is being rolled
                                    back");
                        con.rollback();
                  }
                  catch(SQLException ex){
                        ex.printStackTrace();
                  }
            }
            return false;
      }
      return true;
}

public static void init(){

      boolean initCoffees = true;
      boolean initSuppliers = true;
      Connection con = null;
      Statement stmt = null;

      try {
            Class.forName(DBMS_DRIVER);

      } catch(java.lang.ClassNotFoundException e) {
            e.printStackTrace();
      }
      try {
            con = DriverManager.getConnection(DB_URL);

            DatabaseMetaData dbmeta = con.getMetaData();
            ResultSet rsdb = dbmeta.getTables("","","",null);
            while(rsdb.next()){
                  String tableName = rsdb.getString("TABLE_NAME");
                  if(tableName.compareToIgnoreCase("coffees")==
                              0)initCoffees = false;
                  if(tableName.compareToIgnoreCase("suppliers")==
                              0)initSuppliers = false;
```

[1] Recupera le chiavi autogenerate dallo statement e ritorna un ResultSet, contente le chiavi.

```
            }
            if( !(initCoffees&&initSuppliers) ){
                  System.out.println("COFFEESS and SUPPLIES tables already
                              present");
                  System.out.println("Dropping COFFEES and SUPPLIES
                              tables");
                  stmt = con.createStatement();

                  stmt.executeUpdate("drop table COFFEES");
                  stmt.executeUpdate("drop table SUPPLIERS");
            }
            System.out.println("Inizializing COFFEESS and SUPPLIES
                              tables");
            stmt = con.createStatement();
            stmt.executeUpdate(SQL_CREATE_COFFEES);
            stmt.executeUpdate(SQL_CREATE_SUPPLIERS);

            //Initialising SUPPLIERS table
            stmt.executeUpdate("insert into SUPPLIERS " +
                        "values(1, 'Superior Coffee', '1 Party Place', " +
                        "'Mendocino', 'CA', '95460')");

            stmt.executeUpdate("insert into SUPPLIERS " +
                        "values(2, 'Acme, Inc.', '99 Market Street', " +
                        "'Groundsville', 'CA', '95199')");

            stmt.executeUpdate("insert into SUPPLIERS " +
                  "values(3, 'The High Ground', '100 Coffee Lane', " +
                  "'Meadows', 'CA', '93966')");

            // Initialising COFFEES table
            stmt.executeUpdate("insert into COFFEES " +
                  "values('Colombian', 2, 7.99, 0)");

            stmt.executeUpdate("insert into COFFEES " +
                  "values('French_Roast', 1, 8.99, 0)");

            stmt.executeUpdate("insert into COFFEES " +
                  "values('Espresso', 3, 9.99, 0)");

            stmt.executeUpdate("insert into COFFEES " +
                  "values('Colombian_Decaf', 2, 8.99, 0)");

            stmt.executeUpdate("insert into COFFEES " +
                  "values('French_Roast_Decaf', 1, 9.99, 0)");

            stmt.close();
            con.close();
      }
      catch(SQLException e) {
            e.printStackTrace();
      }
}

public static void main(String[] args) {

            TheCoffeeBreak.init();

            System.out.print("\n\n");
            System.out.println("Now I'm showing COFFEES table data:\n");
            System.out.println(TheCoffeeBreak.showCoffees());
            System.out.print("\n\n");
```

```
                System.out.println("This is COFFEES table data ordered by
                        SUPPLIER\n");
                System.out.println(TheCoffeeBreak.showCoffeesBySupplier());
                System.out.print("\n\n");

                System.out.println("Now I'm doing a purchase order:");
                System.out.println("\t Colombian= 2; Espresso=3");
                if(!TheCoffeeBreak.savePurchaseOrder(new int[]{2,3}, new
                        String[]{"Colombian","Espresso"})){
                      System.out.println("ERROR: I can't save purchase
                            order\n");
                }
                System.out.print("\n");

                System.out.println("COFFEES table data, after the order:\n");
                System.out.println(TheCoffeeBreak.showCoffees());
                System.out.print("\n\n");

                System.out.println("Finally I delete Acme, Inc supplier");
                if(!TheCoffeeBreak.deleteSupplier(2)){
                      System.out.println("ERROR: I can't delete the
                            supplier\n");
                }
                System.out.print("\n");

                System.out.println(".. and I insert NewCoffee supplier:");
                System.out.println("\t Address: NewCoffee, via street,Pisa,IT,
                        1234");
                System.out.println("\t Produced coffee: Espresso");
                if( !TheCoffeeBreak.addSupplier("Espresso", 1, new
                      String[]{"NewCoffee","street","Pisa","IT","1234"}) ){
                      System.out.println("ERROR: I can't add the supplier\n");
                }
                else{
                      System.out.print("\n");
                      System.out.println("This is new COFFEES table data
                            ordered by SUPPLIER\n");

      System.out.println(TheCoffeeBreak.showCoffeesBySupplier());
                }
      }
}
```

Esercitazione TIGA: JDBC

Una piccola caffetteria, chiamata The Coffee Break, utilizza un Database per tenere traccia della quantità di caffé venduto e dei relativi fornitori. L'obiettivo del proprietario della caffetteria è quello di poter memorizzare le informazioni relative alle qualità di caffé offerte ai sui clienti e i riferimenti dei rispettivi produttori.

Il Database è costituito da due tabelle: `COFFEES` e `SUPPLIERS`.
Le colonne della tabella `COFFEES` sono

- `COF_NAME`, utilizzata per memorizzare il nome dei caffé venduti, contiene valori di tipo `VARCHAR` e lunghezza massima di 32 caratteri,
- `SUP_ID`, contiene un identificatore del produttore, di tipo `INTEGER`
- `PRICE`, di tipo `FLOAT`,
- `SALES` per memorizzare la quantità di caffé venduto nella settimana, di tipo `INTEGER`,

Le colonne della tabella `SUPPLIERS` contengono i dati dei fornitori e sono

- `SUP_ID`, di tipo `INTEGER`, è la chiave primaria della tabella, identifica univocamente un fornitore,
- `SUP_NAME`, nome del fornitore, di tipo `VARCHAR` e lunghezza massima di 40 caratteri
- `STREET`, indirizzo, di tipo `VARCHAR` e lunghezza massima di 40 caratteri
- `CITY`, di tipo `VARCHAR` e lunghezza massima di 20 caratteri
- `STATE`, di tipo `VARCHAR` e lunghezza massima di 2 caratteri
- `ZIP`, di tipo `VARCHAR` e lunghezza massima di 5 caratteri

Le operazioni che posso essere fatte sulla caffetteria `TheCoffeeBreak` sono (almeno) le seguenti:

- `public String showCoffees()` che ritorna l'elenco delle qualità di caffé disponibili, il relativo prezzo e la quantità di caffé venduto con il seguente formato:

```
Coffees:
   Name=coffeeName_1
         Price=price_1
         Sales=sales_1
   ….
   Name=coffeeName_n
         Price=price_1
         Sales=sales_1
```

- `public String showCoffeesBySupplier()` che visualizza, per ogni produttore, le qualità di caffé prodotte, con il seguente formato:

```
Coffees by Supplir:
   Supplier= supplierName_1
         Coffee= coffeeName
   Supplier= supplierName_n
         Coffee = coffeeName
```

- `public boolean savePurchaseOrder(int[] sales, string[] coffees)`[1] che aggiorna il database in base all'ordinazione fatta dal tavolo e ritorna `true` se l'operazione è andata a buon fine
- `public boolean deleteSupplier(int supplierID)`[2][3] che elimina un fornitore in base all'`ID` fornito, ritorna `true` se l'operazione è andata a buon fine
- `public boolean addSupplier(String coffeeName, Float coffeePrice, String[] supplierData)`[4] aggiunge un fornitore e la qualità di caffé offerta, ritorna `true` se l'operazione è andata a buon fine

[1] La dimensione dei due array è identica, `sales[i]` contiene il numero di ordinazioni di `coffees[i]`
[2] L'operazione di cancellazione di un Fornitore richiede la cancellazione delle relative qualità di caffé offerte
[3] Le operazioni `deleteSupplier` che `addSupplier` agiscono sulle due tabelle, l'aggiornamento del database deve essere fatto come operazione atomica
[4] Non si dà indicazione del supplierID, se trovate difficoltà aggiungetelo alla firma del metodo, in questo modo: `public boolean addSupplier(String coffeeName, Float coffeePrice, Int supplierID, String[] supplierData)`