```java
package Conexao;

/**
 *
 * @author Cynthia Lopes
 * @author Edwar Saliba Júnior
 */
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public final class Database {

    private Access informationDB;
    private Connection connectionDB;
    private Statement queryDB;
    private boolean enableMessages;
    private final String c_DATABASE_FILE = "JSP_Ex02.txt";
    private int dbms; // Sistema Gerenciador de Banco de Dados

    /**
     * Este método retorna um número inteiro que corresponde ao Sistema Gerenciador
     * de Banco de Dados que está sendo tratado pela classe Database. O valor a
     * ser retornado poderá ser:
     * 1 - quando o SGBD utilizado for o PostgreSQL e
     * 2 - quando o SGBD utilizado for o MySQL.
     */
    public int getDbms() {
        return dbms;
    }

    /**
     * Este método configura um parâmetro importante para se trabalhar com a
     * classe Database, nele deve-se escolher entre dois tipos de Sistemas
     * Gerenciadores de Banco de Dados (PostgreSQL e MySQL). O valor a ser
     * passado para o parâmetro "sgdb" deverá ser:
     * 1 - quando o SGBD utilizado for o PostgreSQL e
     * 2 - quando o SGBD utilizado for o MySQL.
     * @param dbms
     */
    public void setDbms(int dbms) {
        this.dbms = dbms;
    }

    public Database() throws FileNotFoundException, ClassNotFoundException {

        FileReader file = new FileReader(c_DATABASE_FILE);
        BufferedReader reader = new BufferedReader(file);

        String dataBaseName;
        try {
            dataBaseName = reader.readLine();
            String password = reader.readLine();
            String host = reader.readLine();

            reader.close();
            file.close();

            this.informationDB = new Access(dataBaseName, password, host);
            this.connectionToDB();
            this.enableMessages = true;

            // Valor padrão para o atributo é o SGBD PostgreSQL.
            this.dbms = 1;
        } catch (IOException ex) {
            Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public Database(String dataBaseName, String password, String host) {
        try {
            this.informationDB = new Access(dataBaseName, password, host);
            this.connectionToDB();
            this.enableMessages = false;
        } catch (Exception ex) {
            Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

```

```
89      public Database(Access info) {
90          this.informationDB = info;
91          this.enableMessages = false;
92      }
93
94      /**
95       * O parâmetro "tipoBanco" deve receber: - 1 para PostgreSQL - 2 para MySQL
96       *
97       * @param tipoBanco (int)
98       * @throws ClassNotFoundException
99       */
100     public void connectionToDB() throws ClassNotFoundException {
101         try {
102             this.startDriver();
103
104             connectionDB = DriverManager.getConnection(
105                     this.informationDB.getURL(),
106                     this.informationDB.getUsuario(),
107                     this.informationDB.getSenha());
108
109             if (this.enableMessages) {
110                 System.out.println("Connection with database '"
111                         + this.informationDB.getNomeBD() + "' sucess completed.");
112             }
113
114             this.queryDB = this.connectionDB.createStatement(
115                     ResultSet.TYPE_SCROLL_INSENSITIVE,
116                     ResultSet.CONCUR_READ_ONLY);
117         } catch (SQLException e) {
118             System.out.println(e.toString());
119         }
120     }
121
122     public void startDriver() throws ClassNotFoundException {
123         switch (dbms) {
124             case 1:
125                 Class.forName("org.postgresql.Driver");
126                 break;
127             case 2:
128                 Class.forName("com.mysql.jdbc.Driver");
129         }
130     }
131
132     public void insertValues(String tableName, String fieldsNames[],
133             String fieldsValues[]) throws SQLException {
134         String query;
135         boolean putQuotationMarks;
136         putQuotationMarks = dbms == 1;
137
138         if (putQuotationMarks) {
139             query = "INSERT INTO \"" + tableName + "\" (";
140             query += returnFieldsNames(fieldsNames) + ")";
141             query += " VALUES(";
142             query += returnValues(fieldsValues, true) + ")";
143         } else {
144             query = "INSERT INTO " + tableName + " (";
145             query += returnFieldsNames(fieldsNames) + ")";
146             query += " VALUES(";
147             query += returnValues(fieldsValues, true) + ")";
148         }
149
150         if (this.enableMessages) {
151             System.out.println(query);
152         }
153
154         this.queryDB.execute(query);
155         if (dbms == 1) {
156             this.connectionDB.commit();
157         }
158     }
159
160     public void deleteValues(String tableName, String condition) throws SQLException {
161         String query;
162         if (dbms == 1) {
163             query = "DELETE FROM \"" + tableName + "\"";
164         } else {
165             query = "DELETE FROM " + tableName + " ";
166         }
167
168         if (!condition.equals("")) {
169             query += " WHERE " + condition;
170         }
171
172         if (this.enableMessages) {
173             System.out.println(query);
174         }
175
176         this.queryDB.execute(query);
177         if (dbms == 1) {
```

```
177         if (dbms == 1) {
178             this.connectionDB.commit();
179         }
180     }
181 
182     public void updateValues(String tableName, String fields[], String values[],
183             String condition) throws SQLException {
184         String query;
185 
186         if (dbms == 1) {
187             query = "UPDATE \"" + tableName + "\" SET ";
188             query += this.returnSetValues(fields, values, true);
189         } else {
190             query = "UPDATE " + tableName + " SET ";
191             query += this.returnSetValues(fields, values, false);
192         }
193         if (!condition.equals("")) {
194             query += " WHERE " + condition;
195         }
196 
197         if (this.enableMessages) {
198             System.out.println(query);
199         }
200 
201         this.queryDB.execute(query);
202         if (dbms == 1) {
203             this.connectionDB.commit();
204         }
205     }
206 
207     public boolean existRow(String table, String condition) throws SQLException {
208         boolean foundRow;
209 
210         String query = "SELECT 1 ";
211         query += " FROM \"" + table + "\"";
212 
213         if (!condition.equals("")) {
214             query += " WHERE " + condition;
215         }
216 
217         ResultSet resultSet = this.queryDB.executeQuery(query);
218         foundRow = resultSet.first();
219 
220         if (this.enableMessages) {
221             System.out.println(query);
222         }
223 
224         return (foundRow);
225     }
226 
227     public void printSelection(String table, String fields[], String condition)
228             throws SQLException {
229         String query = "SELECT ";
230         query += returnFieldsNames(fields);
231         query += " FROM \"" + table + "\"";
232 
233         if (!condition.equals("")) {
234             query += " WHERE " + condition;
235         }
236 
237         ResultSet resultSet = this.queryDB.executeQuery(query);
238 
239         if (this.enableMessages) {
240             System.out.println(query);
241         }
242 
243         while (resultSet.next()) {
244             String print = "";
245 
246             for (int i = 0; i < fields.length; i++) {
247                 print += "|" + resultSet.getString(fields[i]) + "|\t";
248             }
249 
250             System.out.println(print);
251         }
252     }
253 
254     public ArrayList selection(String table, boolean putQuotationMarksOnTheFields,
255             String fields[], String condition) {
256         ArrayList resultsList = new ArrayList();
257 
258         try {
259             String query = "SELECT ";
260             if (putQuotationMarksOnTheFields) {
261                 query += returnFieldsNames(fields);
262             } else {
263                 query += returnValues(fields, putQuotationMarksOnTheFields);
264             }
265             query += " FROM " + table;
```

```
266
267             if (!condition.equals("")) {
268                 query += " WHERE " + condition;
269             }
270
271             ResultSet resultSet = this.queryDB.executeQuery(query);
272
273             if (this.enableMessages) {
274                 System.out.println(query);
275             }
276
277             resultSet.beforeFirst();
278             while (resultSet.next()) {
279                 String[] row = new String[resultSet.getMetaData().getColumnCount()];
280                 for (int i = 0; i < resultSet.getMetaData().getColumnCount(); i++) {
281                     row[i] = resultSet.getString(i + 1);
282                 }
283                 resultsList.add(row);
284             }
285         } catch (SQLException ex) {
286             System.out.println("Exceção SQL: " + ex);
287         }
288         return resultsList;
289     }
290
291     public ResultSet selection(String table, String fields[],
292             boolean putQuotationMarksOnTheFields, String condition,
293             boolean pointerToFirstRecord) throws SQLException {
294         String query = "SELECT ";
295         if (putQuotationMarksOnTheFields) {
296             query += returnFieldsNames(fields);
297             if (dbms == 1) {
298                 query += " FROM \"" + table + "\"";
299             } else {
300                 query += " FROM " + table + " ";
301             }
302         } else {
303             query += returnValues(fields, putQuotationMarksOnTheFields);
304             query += " FROM " + table;
305         }
306
307         if (!condition.equals("")) {
308             query += " WHERE " + condition;
309         }
310
311         ResultSet resultSet = this.queryDB.executeQuery(query);
312         if (pointerToFirstRecord) {
313             resultSet.next();
314         }
315
316         return (resultSet);
317     }
318
319     public ResultSet selection(String query) throws SQLException {
320         ResultSet resultSet = this.queryDB.executeQuery(query);
321         resultSet.next();
322
323         return (resultSet);
324     }
325
326     public void printJoinSelection(String tables[], String fields[],
327             String condition) throws SQLException {
328         String query = "SELECT ";
329         query += returnFieldsNames(fields);
330         query += " FROM " + returnFieldsNames(tables);
331
332         if (!condition.equals("")) {
333             query += " WHERE " + condition;
334         }
335
336         ResultSet resultSet = this.queryDB.executeQuery(query);
337
338         if (this.enableMessages) {
339             System.out.println(query);
340         }
341
342         for (int i = 0; i < fields.length; i++) {
343             int dotPosition = fields[i].indexOf('.');
344             fields[i] = fields[i].substring(dotPosition + 1, fields[i].length());
345         }
346
347         while (resultSet.next()) {
348             String print = "";
349
350             for (int i = 0; i < fields.length; i++) {
351                 print += "|" + resultSet.getString(fields[i]) + "|\t";
352             }
353
```

```
354             System.out.println(print);
355         }
356     }
357 
358     public String returnValues(String values[], boolean putQuotationMarks) {
359         String vals = "";
360 
361         if (putQuotationMarks) {
362             for (int i = 0; i < values.length - 1; i++) {
363                 vals += "\'" + values[i] + "\', ";
364             }
365 
366             vals += "\'" + values[values.length - 1] + "\'";
367         } else {
368             for (int i = 0; i < values.length - 1; i++) {
369                 vals += values[i] + ", ";
370             }
371 
372             vals += values[values.length - 1];
373         }
374 
375         return vals;
376     }
377 
378     public String returnFieldsNames(String values[]) {
379         boolean putQuotationMarks;
380         putQuotationMarks = dbms == 1;
381         String vals;
382         if (putQuotationMarks) {
383             vals = "\"";
384 
385             for (int i = 0; i < values.length - 1; i++) {
386                 vals += values[i] + "\", \"";
387             }
388 
389             vals += values[values.length - 1] + "\"";
390         } else {
391             vals = "";
392 
393             for (int i = 0; i < values.length - 1; i++) {
394                 vals += values[i] + ", ";
395             }
396 
397             vals += values[values.length - 1];
398         }
399 
400         return vals;
401     }
402 
403     public String returnSetValues(String fields[], String values[],
404             boolean putQuotationMarks) {
405 
406         String vals = "";
407 
408         for (int i = 0; i < values.length - 1; i++) {
409             if (putQuotationMarks) {
410                 vals += "\"" + fields[i] + "\" = "
411                         + (values[i].equals("") ? "\'\'" : "\'"
412                         + values[i] + "\'") + ", ";
413             } else {
414                 vals += "" + fields[i] + " = "
415                         + (values[i].equals("") ? "\'\'" : "\'"
416                         + values[i] + "\'") + ", ";
417             }
418         }
419 
420         if (putQuotationMarks) {
421             vals += "\"" + fields[fields.length - 1] + "\" = "
422                     + (values[values.length - 1].equals("") ? "\'\'" : "\'"
423                     + values[values.length - 1] + "\'");
424         } else {
425             vals += "" + fields[fields.length - 1] + " = "
426                     + (values[values.length - 1].equals("") ? "\'\'" : "\'"
427                     + values[values.length - 1] + "\'");
428         }
429         return vals;
430     }
431 
432     /**
433      * Converte um vetor de inteiros para um formato aceitável por um campo do
434      * tipo "array" do PostgreSQL.
435      *
436      * @param Vetor de Inteiros
437      * @return "'{ val1, val2, ... }'"
438      */
439     public String convertToStringArray(int v[]) {
440 
441         String vetor = "";
```

```
442
443         vetor += "{";
444
445         for (int i = 0; i < v.length; i++) {
446             if ((i < (v.length - 1)) && (!((i > 0) &&
447                     (String.valueOf(v[i]).equals("0"))))) {
448                 vetor += String.valueOf(v[i]) + ",";
449             } else {
450                 vetor += String.valueOf(v[i]);
451                 break;
452             }
453         }
454         vetor += "}";
455
456         return (vetor);
457     }
458
459     /**
460      * Converte um campo do tipo "array" do PostgreSQL (String) num vetor de
461      * inteiros.
462      *
463      * @param "String" - Ex.: {0,80,17,71,13,0}
464      * @return "int []" - Ex.: [0,80,17,71,13,0]
465      */
466     public int[] convertToIntArray(String v) {
467         v = v.trim();
468         int vetor[] = new int[v.split(",", -1).length];
469
470         int j = 0;
471         String n = "";
472         for (int i = 0; i < v.length(); i++) {
473             if ((v.charAt(i) != '{') && (v.charAt(i) != '}')) {
474                 if ((v.charAt(i) != ',') && (v.charAt(i) != ' ')) {
475                     n += v.substring(i, i + 1);
476                 } else {
477                     vetor[j++] = Integer.parseInt(n);
478                     n = "";
479                 }
480             }
481         }
482
483         return (vetor);
484     }
485
486     public void closeDBConnection() throws SQLException {
487         this.connectionDB.close();
488     }
489
490     public boolean isEnableMessages() {
491         return enableMessages;
492     }
493
494     public void setEnableMessages(boolean enableMessages) {
495         this.enableMessages = enableMessages;
496     }
497
498     @Override
    public void finalize() throws SQLException, Throwable {
500         super.finalize();
501         closeDBConnection();
502     }
503
504     private Object getFieldValue(String tableName, String fieldName,
505             String condition) throws SQLException {
506         boolean putQuotationMarks;
507         Object value = null;
508         String fields[] = {fieldName};
509
510         /* Se o banco não for PostgreSQL, não colocar
511          aspas no campo e tampouco na tabela.       */
512         putQuotationMarks = (dbms == 1);
513
514         ResultSet rs = selection(tableName, fields,
515                 putQuotationMarks, condition, true);
516
517         if (rs.first()) {
518             value = rs.getObject(fieldName);
519         }
520
521         return (value);
522     }
523
524     public String getStringFieldValue(String tables, String fields,
525             String condition) throws SQLException {
526         String value = "";
527         Object val;
528
529         val = getFieldValue(tables, fields, condition);
```

```
        if (val != null) {
            value = val.toString();
        }

        return (value);
    }

    public int getIntFieldValue(String tables, String fields, String condition)
            throws SQLException {
        int value = 0;
        Object val;

        val = getFieldValue(tables, fields, condition);
        if (val != null) {
            value = Integer.valueOf(val.toString());
        }

        return (value);
    }

    public float getFloatFieldValue(String tables, String fields, String condition)
            throws SQLException {
        float value = 0;
        Object val;

        val = getFieldValue(tables, fields, condition);
        if (val != null) {
            value = Float.valueOf(val.toString());
        }

        return (value);
    }

    public boolean getBooleanFieldValue(String tables, String fields,
            String condition) throws SQLException {
        boolean value = false;
        Object val;

        val = getFieldValue(tables, fields, condition);
        if (val != null) {
            value = Boolean.valueOf(val.toString());
        }

        return (value);
    }

    public int getNumberOfRowsInTable(String table, boolean putQuotation)
            throws SQLException {
        ResultSet rs;

        if (putQuotation) {
            rs = selection("SELECT count(*) "
                    + "  FROM \"" + table + "\"");
        } else {
            rs = selection("SELECT count(*) "
                    + "  FROM " + table);
        }
        int value = rs.getInt(1);

        return value;
    }
}
```