TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 66, Issue I, March, 2023

A CRUD IMPLEMENTATION IN JDEVELOPER AND MS ACCESS OF A FLAT DATABASE FOR STORING ROBOT PROGRAMS

Tiberiu Alexandru ANTAL

Abstract: *The paper aims to give a detailed description of how to implement a Java graphical interface for the development of an application that uses a single table for storing robot programs. The research uses the JDeveloper IDE and the MS Access relational database management system, presenting the particularities of an implementation that must ensure basic management, in the sense of CRUD (Create Read, Update, Delete), of data from the application.*
Key words: *CRUD, database, GUI, Java, robot.*

1. INTRODUCTION

1.1 The concept of flat database and CRUD

The term database revolves around the concept of data. Data is a measurement of some parameters of a process. Most real life processes are complex and require some model based approach to be computable. The concept of data model is related to information algebra where subject is defined as a mathematical formalism for defining the data structures and the operators to validate and manipulate the data. The term of database refers to an organized collection (based on some data model) of structured data that is accessible using a software called database management system. At a database level the model is a specification describing how a database is structured and used. The flat database model sometimes called the table model consists of a single table also called as a two-dimensional array of data elements, where all data of a column are assumed to have the same type, and all rows are assumed to be related to one another [1]. The concept of type is defined as a classification of data based on predefined categories described by: domain, operators, axioms and preconditions (some type examples would be - integer, real or string).

The database management system used in this implementation is a Microsoft product called MS Access and the integrated development environment (IDE) used to develop the Java application is JDeveloper, a product of Oracle. While the MS Access database management system is easy to use in the context of Microsoft development environments, JDeveloper is easy to use in the context of database management systems approved by Oracle. The JDeveloper is designed to connect and work with Oracle databases and a number of non-Oracle databases (DB2, MySQL, SQL Server, etc.) at IDE level but it will not connect to MS Access. However, in the context of creating small or medium-sized databases that are not used by many users, MS Access remains a cheap, secure, robust, easy-to-learn and very efficient database management system thanks to the simple manipulation interfaces it provides to the user. MS Access is a database management system based on the relational data model [6] (Relational database Management System or RDBMS) invented by the English computer scientist named Edgar Frank Codd in 1970. According to this data model the data structure used to save the data is the table and the operators use to manipulate and validate the data are part of relational algebra (projection, join, composition, restriction). Each table must have a name and all columns in a

Received: 30.01.23; **Similarities**: 16.02.23: **Reviewed**: 02.03./26.02.23: **Accepted**:20.03.23.

table must have unique names and associated types (like real, integer or string). In [6], E. F. Codd, introduced the concept of normalization in order to minimize redundancy from tables. Normalization is obtained by decomposing a table in two tables with lover redundancy while keeping some common values (in columns called keys) that allow the recomposing the original table. There are various levels of normalization; some of them are called as follows [1], [8]:

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF) etc

In flat databases the number of tables is restricted to one and that structure of the table must obey the First Normal Form, that is, the table must have a primary key and any composite or multi-valued data in cells (row-column intersection) must be eliminated by adding new rows in the table. A primary key is the minimal set columns that can uniquely identify a row in the table. In order to demonstrate the presented concepts, consider the data from Table 1 where a set of statements are presented grouped in order to associate them to a certain program.

Table 1

A presentation table for the program-statement association stored as "text processor document".

program1	statement11 statement12 statement13
program2	statement21 statement22
program3	statement31 statement32 statement33 statement34

Although visually speaking a table is observed, this has nothing to do with the table concept used in the relational data model. Reorganizing it according to the requirements of the relational data model will lead, step by step, to a new table where:

- at the intersection of a row and a column the multi-valued data in cells must be eliminated by providing individual rows and columns for each piece of data (see Table 2 and Table 3);
- the columns must have a name and type (see Table 4);
- a primary key must be provided for the rows in the table (see Table 5);
- the table must have a name (see Table 5).

Table 2

Column two reorganized in individual rows.

program1	statement11
	statement12
	statement13
program2	statement21
	statement22
program3	statement31
	statement32
	statement33
	statement34

Table 3

Column one data reorganized in rows to match the rows in column two.

program1	statement11
program1	statement12
program1	statement13
program2	statement21
program2	statement22
program3	statement31
program3	statement32
program3	statement33
program3	statement34

Table 4

Columns received name and type.

Program (type string)	Statement (type string)
program1	statement11
program1	statement12
program1	statement13
program2	statement21
program2	statement22
program3	statement31
program3	statement32
program3	statement33
program3	statement34

Table 5

The table receives the Programs name and a new column with the name ID and integer type for the primary key values.

ID	Program	Statement

(type integer, PK)	(type string)	(type string)
1	program1	statement11
2	program1	statement12
3	program1	statement13
4	program2	statement21
5	program2	statement22
6	program3	statement31
7	program3	statement32
8	program3	statement33
9	program3	statement34

1.2 JDBC design and architecture

In 1996, Sun released the first version of Java Database Connectivity (or JDBC) as a vendor independent Application Programming Interface (or API) for connecting programs written in Java to relational databases using the Structured Query Language (or SQL). As Java was designed to be platform independent, using the vendor independent JDBC, the same Java program could run on different processors and operating systems without being rewritten. Compared to the database applications written in a proprietary database language, using a database management system available only from a single vendor and on a single platform, this approach was a huge breakthrough. JDBC has been updated several times but still has no visual database development tools. Up to this point JDBC 4.3 released in 2017, under JDK SE 9, is the latest stable version.

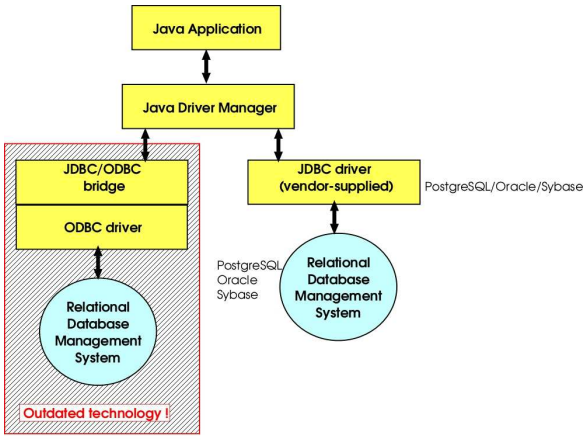

Fig. 1. - JDBC to database communication architecture.

In Figure 1 we can also see the ODBC abbreviation which stands for Open Database Connectivity that was originally a proprietary Sybase Open Client API which MS SQL Server inherited when Microsoft bought the Sybase source code and the rights to produce their own Windows based RDBMS from that codebase from Sybase Corp. As the first Microsoft's ODBC API was released in 1992 - a C programming language interface for database interaction - this was already available on the database market before de first JDBC API [2] was released. JDBC designers tried to adapt this new architecture to deal with the database vendors that already provided ODBC drivers for they databases. To access ODBC databases using the JDBC interfaces a JDBC/ODBC bridge program must be used. Both JDBC and ODBC are based on the same idea: applications are using the API to talk to the driver manager, which, in turn, use the driver to talk to the database. This means that most of the programmers will only use the API to work with the database. The JDBC driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers [7]:

1. translates JDBC to ODBC and relies on the ODBC driver to communicate with the database; the JDK only contains the JDBC/ODBC bridge which requires the correctly configured ODBC driver;
2. native-API driver: partially java driver and partially native code; it communicates with the client API of the database and needs some platform specific code in addition to the Java library;
3. network protocol driver (middleware driver): a pure Java client library that uses a database independent protocol to communicate database request to a server component which translates the request to database specific protocol;
4. thin driver (pure Java driver): a pure Java library that translates JDBC calls directly to database specific protocol.

The JDBC/ODBC bridge driver that was part of Java 5, 6, and 7 has no longer support in Java 8. This means that after upgrading to Java 8 and try to connect to a MS Access database (or other

databases that require an ODBC driver instead of a JDBC driver) you will get the error: 'java.lang.ClassNotFoundException: sun.jdbc.odbc.JdbcOdbcDriver'. One of the free solutions available as an open-source Java JDBC driver (type 4) implementation that allows Java developers and JDBC client programs to read/write Microsoft Access databases is named UcanAccess and can be found at [5].

1.3 CURD in SQL for JDBC

The term CRUD is an acronym of four fundamental operations to obtain persistent database applications:

- Create - create data;
- Read - read data;
- Update - update data;
- Delete - erase data.

JDBC communicates with the database using the SQL language. It is not the purpose of this work to give a description of the SQL language; however a brief SQL introduction is going to be performed using the table from Figure 1 in order to perform CURD operations on the table. It should be noted that the ID column (field) in Table1 is PK and it has the AutoNumber type. In MS Access this type is automatically incremented and is used to create an identity column which uniquely identifies each record.

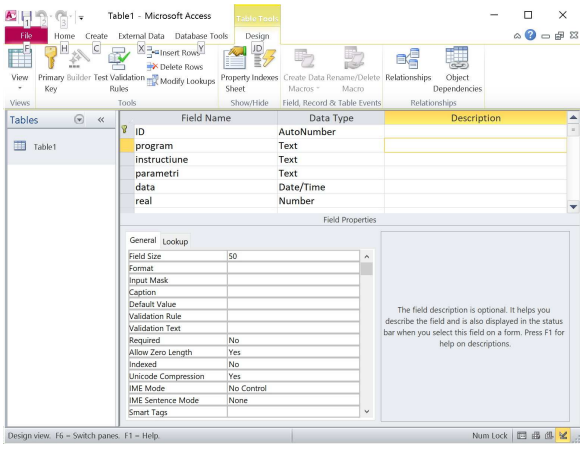

Fig. 2. - MS Access table structure used in the flat database.

In **C**RUD operations the 'C' is an acronym for create, which means to add or insert data into the SQL table. At SQL language level the INSERT INTO command is used to insert rows in a table. A typical insertion in Table1 would look like this:

INSERT INTO Table1([program], [instructiune],[parametri],[data],[real]) VALUES ('program4','grip','open',#2022-12-28 21:25:22#,0.0)

The 'R' in C**R**UD operations stands for read, which means retrieving or fetching the data from a table. For this the SELECT SQL command is used on Table1. To retrieve all the records from the table an asterisk (*) is used in the SELECT. There is also an option of retrieving only those records which satisfy a particular condition by using the WHERE clause and to sort the records ascending using the ORDE BY clause in SELECT.

*SELECT * FROM Table1*

*SELECT * FROM Table1 ORDER BY program*

*SELECT * FROM Table1 WHERE program = 'program1'*

In CR**U**D operations the 'U' is an acronym for update, which means making changes to the records already present in the table. The SQL command for this is UPDATE and it will modify the data present in Table1. The modification of a row from Table 1 using the WHERE clause that is uniquely identified by the value in the ID column is written as follows:

UPDATE Table1 SET [program]='program4', [instructiune]='grip',[parametri]='close',[data]=#2022-12-28 21:25:22# WHERE [ID] = 39

The 'D' in CRU**D** is the acronym for delete, which means removing records from a table. We can delete all the rows from tables using the SQL DELETE command. There is also an option to remove only specific records that satisfy a particular condition by using the WHERE clause in a DELETE query as shown in the following example:

*DELETE * FROM Table1 WHERE [ID]=39*

2. JDBC PROGRAMMING CONCEPTS

2.1 UcanAccess driver installation in JDeveloper

The communication between the Java application and the desktop RDMS MS Access is based on the third party type 4 JDBC driver called UcanAccess. When downloaded from [5] the "UCanAccess-5.0.1.bin.zip" archive will contain all the necessary files for the installation. As type 4 drivers are written in Java and stored in jar files, the term of installation is only about configuring the Integrated Development Environment (or IDE), in this case JDeveloper, to work with the classes provided as JDBC drivers. The safest way to work with the diver is to unzip it to a directory, keeping the default directory name [UCanAccess-5.0.1.bin], into the application project directory created by JDeveloper. The following step is to add to the *Project Properties* at *Libraries and Classpath* the *jar* files shown in Figure 3:

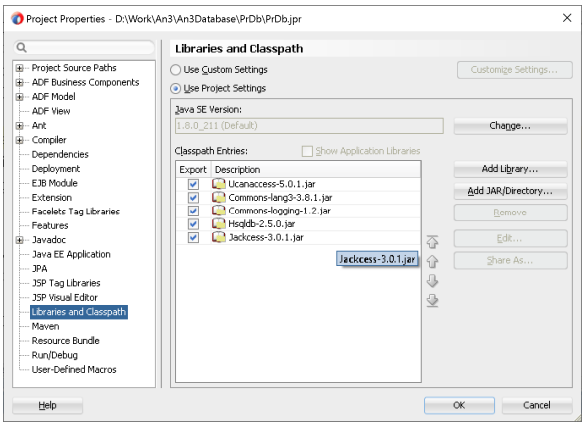

Fig. 3. - The five jar files added to the JDeveloper Libraries and Classpath to work with UcanAccess JDBC driver.

2.2 JDBC programming basics

The classes used to program JDBC are contained in the java.sql and javax.sql packages [3], [4]. Programming consists of building objects based on these classes to achieve the required actions at database level. Normally the first step is about specifying the data source which would identify the database by describing the location of the data using syntax similar to a Uniform Resource Locator (or URL). The second step is about using a registered driver to create a connection to the database. The driver registration is made using the static class *Class.forName(String name)* and the typical code for a MS Access database is:

```
Class.forName("net.ucanaccess.jdbc.UcanaccessDriver"); /* often not required for Java 6 and later (JDBC 4.x) */
```

In the case on MS Access the registration is not required starting from Java 6, so only the connection object must be instantiated to start a session with the database by using:

```
Connection conn=DriverManager.getConnection("jdbc:ucanaccess:// d:\Work\Database\PrDb\test.mdb);
```

To execute a SQL command a *Statement* object must be created as follows:

```
Statement stat = conn.createStatement();
```

Then, an SQL *SELECT* command stored as a string must be created and the *executeQuery()* method of the *Statement* must be called as:

```
String sql = "SELECT * FROM Table1 ORDER BY program;
ResultSet rs = stat.executeQuery(sql);
```

However, if the SQL command to be executed is: *INSERT*, *UPDATE* or *DELETE* the *executeUpdate()* method must be called.

```
String sql = "DELETE * FROM Table1 WHERE [ID]=39";
stat.executeUpdate(sql);
```

The *executeUpdate()* method returns the number of rows affected by the SQL command, while the *executeQuery()* returns an object set of the *ResultSet* type. Initially, the ResultSet object is positioned before the first row in the result. The *next()* method returns true if there is a next row in the result otherwise returns false. After executing the *next()* method, the *ResultSet* object updates the internal pointer to the current row.

The basic loop for traversing the result set look like this:

```
while (rs.next())
{
    do something with the current row
}
```

When positioned on a row from the record set object in order to inspect the values of a filed several accessor methods are available depending on the type conversions to be made, for example *getDouble()* or *getString()*. Each accessor has two forms, one that takes a numeric argument (the column number) and one that takes a string (the column name) in order to access a certain field from the current row as shown in the example:

```
String program = rs.getString(2);
double x = rs.getDouble("real");
```

3. JDEVELOPER IMPLEMENTATION OF THE FLAT DATABASE

The structure of the application is presented in Figure 4, where *FrDatabase*, *FrInsert* and *FrUpdate* are *JFrames* created in JDeveloper based on [9] - [11].

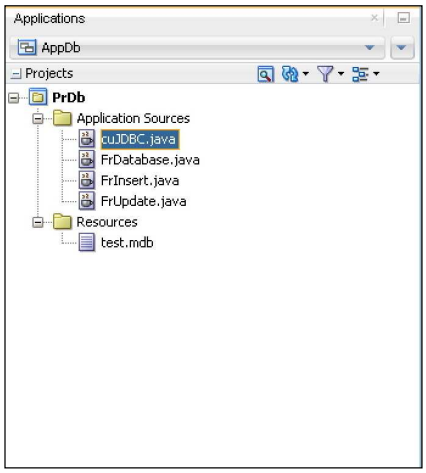

Fig. 4. - The three *JFrames* that implement the CRUD operation on *Table1* table from *test.mdb* MS Access database.

The **read** part form the C**R**UD suite of operations is visible on the main frame in Figure 5. The rows from *Table1* are read into a *JTable* object and can be sorted (using checkboxes) or filtered (using the listbox) based on the program names in the [*program*] field.

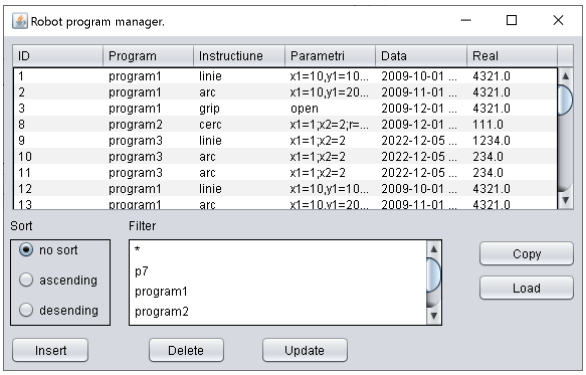

Fig. 5. - The main window of the Java application for storing robot programs in the MS Access database.

The **create** action from **C**RUD is implemented at basic level in Figure 6, where a dedicate *JFrame* named *FrInsert* is collecting the specific data for the new row. Form the main window the *Insert* button must be clicked for this. The code behind the button is:

```
fi = new FrInsert();
fi.setVisible(true);
fi.setThis(this); //stored in parent field of fi
```

The *setThis()* method transfers a reference from the *FrDatabase* main window (which stays opened) to the *FrInsert* window which stores it in the *parent* field.

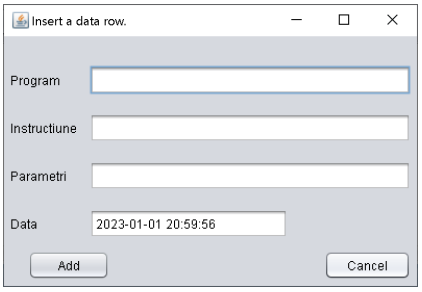

Fig. 6. - The insert JFrame to comply the create part from CRUD.

The code on the *Add* button is:

```
String sqlins = "INSERT INTO Table1" +
"([program],[instructiune],[parametri],[data],
[real]) VALUES("+
        """+jTextField1.getText()+"","+
```

```
        "'"+jTextField2.getText()+"',"+
        "'"+jTextField3.getText()+"',"+
        "#"+jTextField4.getText()+"#,0.0)";
parent.runSQL(sqlins);
this.dispose();
```

The code assembles an SQL string based on the input values and runs the *runSQL()* method from the main window based on the reference passed to the *FrInsert JFrame*. Please keep in mind that the presented sequence does not check the right matching or the correctness of the values to be inserted just for the sake of simplicity and clarity of the code. The *runSQL()* method's code is given bellow:

```
public void runSQL(String sql) {
  Connection con;
  Statement st;
  try {
    String url = UcanaccessDriver.URL_PREFIX + filename + ";newDatabaseVersion=V2003";
    con = DriverManager.getConnection(url, "", "");
    st = con.createStatement();
    st.executeUpdate(sql);

    fillTable("SELECT * FROM Table1;");
    fillListBox("SELECT DISTINCT program FROM Table1;");
  } catch (SQLException e) {
    System.out.println("Exception in runSQL: " + e);
  }
}
```

This runs the SQL command then updates the contents of the *JTable* as well as the contents of the *ListBox* (which will extend automatically if a new program name is added to the database). Increasing the productivity of code insertion can be improved by two buttons, the *Copy* button, which inserts the current row from the table, and the *Load* button, which loads a program from a text file on the disk into the database. If a statement is just copied them it can be modified easily using the *Update* button. If a sequence of statements is stored in a text file this can be uploaded to the database using the *JFileChooser()* from Figure 7. After the upload the name of the file is going to be the name of the program as shown in Figure 8 (the inserted rows are shown selected).

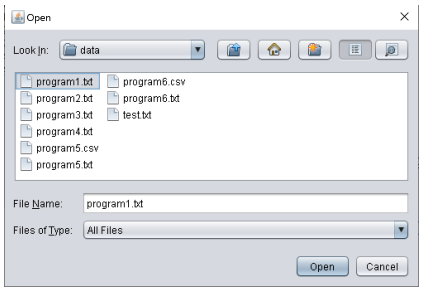

Fig. 7. - The *JFileChooser()* object used to select a program file.

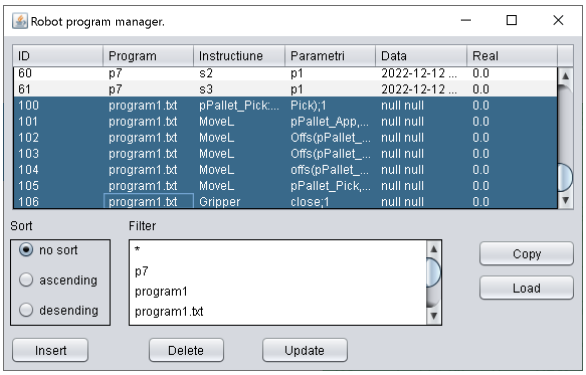

Fig. 8. - The main window contents after loading a file to the database.

The **update** procedure in C**U**RD is implemented with the help of a distinct *JFrame* named *FrUpdate*. This will retrieve the data of the current row and display it in the update form as shown in Figure 9. The update procedure could change the entire content of the row but will always keep the same *ID*.

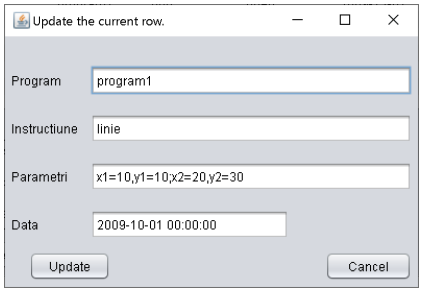

Fig. 9. - The update JFrame to comply the update part from CRUD.

The **delete** in the CRU**D** refers the process of erasing rows in the table. This doesn't need a separate window as the row or rows can be

selected directly from the main window. The code form the *Delete* button is:

```
DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
String sql;

int[] selected = jTable1.getSelectedRows();
for (int i = selected.length - 1; i >= 0; --i) {
  String ID = jTable1.getModel().getValueAt(selected[i], 0).toString();
  model.removeRow(selected[i]);
  sql = "DELETE * FROM Table1 WHERE [ID]=" + ID;
  runSQL(sql);
}
```

The selected row or rows *ID* is returned by the *getSeletedRows()* method of the *JTable* and stored in the *selected* integer table. The *for* loop will visit and delete one by one each corresponding row with the same *ID* in the database using the DELETE SQL command and the *removeRow()* method will update the state of the rows shown in the main window, as deleted rows from the table must no longer be visible in the view presented by the *JTable*.

4. REFERENCES

[1] ANTAL Tiberiu Alexandru, *Microsoft Access 97 şi 2000 în 14 cursuri*, Editura Todesco, 2000, p. 299, ISBN 973-99779-6-0.

[2] ANTAL, T. A., *ACCESS to an ORACLE DATABASE using JDBC*, Acta Technica Napocensis, Series: Applied Mathemathics and Mechanics, Nr. 47, Vol. III, 2004, p.63-68, ISSN 1221-5872.

[3] ANTAL, T. A., *Elemente de Java cu JDeveloper - îndrumător de laborator*, Editura UTPRES, 2013, p.150, ISBN: 978-973-662-827-6.

[4] ANTAL, T. A., *Java - Iniţiere - îndrumător de laborator*, Editura UTPRES, 2013, p. 246, ISBN: 978-973-662-832-0.

[5] https://ucanaccess.sourceforge.net/site.html

[6] https://dl.acm.org/doi/pdf/10.1145/362384.362685

[7] ANTAL, T .A., *Using Oracle JDeveloper 10g to build a Microsoft Access database Java interface*, Acta Technica Napocensis, Series: Applied Mathemathics and Mechanics, Nr. 50, Vol. VII, 2007, p.33-38, ISSN 1221-5872.

[8] ANTAL Tiberiu Alexandru, *Proiectarea paginilor Web cu HTML, VBScript şi ASP - ediţia a II-a*, Editura RISOPRINT, 2006, p.264, ISBN 973-751-349-5.

[9] HORSTMANN, C. S., *Core Java SE 9 for the Impatient - Second Edition*, Addison-Wesley, 2018, p. 538. ISSN 978-0-13-469472-6.

[10] SCHILD, H. Java: *The Complete Reference, Eleventh Edition*, McGraw-Hill Education, 2019, p. 1208, ISBN: 978-1-260-44023-2.

[11] HORSTMANN, C. S., CORNELL, Core *Java 2: Volume II – Advances Features, Seventh Edition*, Prentice Hall, 2005, ISBN13: 9780131118263.

O implementare CRUD în mediul de dezvoltare JDeveloper folosind sistemul de gestionare a bazelor de date relaţionale MS Access a unei baze de date cu un singur tabel pentru stocarea programelor robot

Lucrarea îşi propune să dea o descriere detaliată a modului de implementare a unei interfeţe grafice Java pentru dezvoltarea unei aplicaţii ce foloseşte un singur tabel pentru stocarea de programe robot. Cercetarea utilizează mediul JDeveloper şi sistemul de gestiune a bazelor de date MS Access prezentând particularităţile unei implementări care trebuie să asigure gestionarea elementară, în sensul de CRUD (creare, citire, actualizarea şi ştergere), a datelor din aplicaţie.

ANTAL Tiberiu Alexandru, Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@bavaria.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.