A SQL COOKBOOK USING BSF4OOREXX

Julia Kirisits

Matriculation No.: 1251486

Winter term 2015/16

Vienna University of Economics and Business

Course: Project Seminar of Information Systems

Course-No.: 0182

Specialisation: Business Information Systems

Instructor: ao.Univ.Prof. Dr. Rony G. Flatscher

Declaration of Authorship

I, Julia Kirisits, hereby declare

- that I have written this seminar paper without any help from others and without the use of documents and aids other than those stated in the references
- that I have mentioned all the sources used and that I have cited them correctly according to established academic citation rules
- that the topic or parts of it are not already the object of any work or examination of another course unless this explicitly stated.

Date: 04 February 2016

Signature:

Table of contents

List of Figures

List of Tables

Code Listings

List of Abbreviations

GUI	Graphical User Interface
JDK	Java Development Kit
SQL	Structured Query Language
ver	version

1 Introduction

1.1 Overview

This paper was developed within the course “Project Seminar of Information Systems” in winter term 2015/16.
Its objective is to provide a lightweight code of practice for handling and SQL databases through BSF4ooRexx.
In this cookbook, BSF4ooRexx as an extension to the programming language ooRexx is used to automate SQL operations.
To begin, I will explain the most important tools, I’m dealing with in this cookbook. That is the ooRexx language, BSF4ooRexx and the demonstrated databases.
Second, I will guide you through some easy starters and last show more ambitious examples of using SQL databases through BSF4ooRexx.

1.2 About BSF4ooRexx

BSF4ooRexx is the short name for a project named “The Bean Scripting Framework for ooRexx” by ao.Univ.Prof. Dr. Rony G. Flatscher.
BSF4ooRexx is an external Rexx function package, which allows to create a bridge between ooRexx and Java.[1]

Flatscher describes the project objectives as following:

> “Developing an external function library which allows the typeless scripting language ooRexx (…) to communicate with the strictly typed Java. This allows Rexx to use all of the functionality of Java.”[2]

1.2.1 About ooRexx

“ooRexx” is the abbreviation for “Open Object Rexx”, where “Rexx” means “Restructured Extended Executer”.
The classic REXX was originally developed at IBM by Mike Cowlishaw and mainly targeting mainframe computers. Object REXX – an object-oriented version of REXX – has been developed by IBM in late 1990s. In 2004, IBM handed over Object Rexx to the open source community.

Since then it is distributed as “Open Object Rexx” under Common Public License (CPL).
The Open Object Rexx project is managed and maintained by the Rexx Language Association.[3]
Open Object Rexx offers the advantages of both, classic Rexx as a well-grounded and human-oriented programming language and all the powerfulness of an object oriented programming language.
Here are few of the many advantages of Open Object Rexx, as they can be found on the project website:[4]

- Human-oriented syntax
- English-like commands
- Easy to learn
- Portable to multiple environments
    - E.g. Linux, Unix, Windows
- Fully object oriented

1.3 The databases

This cookbook is written to work with three different Database Management Systems, i.e. MySQL, PostgreSQL and SQLite.
It allows the readers of this paper to take the code “as it is” and use it on their own database, which shows the portability of the ooRexx-Code.
The DBMSs chosen, are open-source, very popular and each has its specific strength.
MySQL is one of the most popular open-source RDBMS (relational database management system).
As a client-server RDBMS, MySQL is deployed on many webserver-infrastructures (e.g. to be used in web applications like Wordpress or TYPO3).[5], [6]
PostgreSQL is an open-source object-relational database management system.
PostgreSQL supports both, object-oriented and relational database functionality.
It is known as a very reliable DBMS, especially because of its ACID-conformity.[7], [8]
SQLite is, in contrast to the other databases, a file-based databases.
SQLite is very standards-aware, fast and efficient, but doesn’t support user authentication.
The best known application of SQLite is on mobile Devices, especially Android.[7]

2 Preparatory Work

Before we can start cooking, we have to do some preparatory work.

In general we need following components:

Figure 1: components needed to run recipes

To get the recipes in this cookbook working, take the following steps as a preparation:

Figure 2: Workflow of preparatory work

Due to the fact that the setup of a RDMBS can vary greatly in different environments, we will pass over this step and move on to the installation of BSF4ooRexx. It is more suitable if you choose that setup, which meets your demands best.

2.1 Install BSF4ooRexx

2.1.1 Install on Linux systems

Note: This description refers to Ubuntu systems as an example

Step 1: Install ooRexx

- Download the appropriate ooRexx-Archive from the official Sourceforge-Site[1]
- In case of Ubuntu this would be a file named like "ooRexx-**ver**.ubuntu3110.x86_x64.deb"
- If the download has finished, install ooRexx by opening the file with your preferred package manager tool.
    - For example, if you prefer a GUI installation in Ubuntu, with "Ubuntu Software Center".

Step 2: Install JRE

- The Open JDK is provided from the official Ubuntu repositories.
    - Further information: [9]
- So, the easiest way is to install the Open JDK by your preferred package manager.

Step 3: Install BSF4ooRexx

- To install BSF4ooRexx from the scratch
    - Download the installation archive from the BSF4ooRexx-Sourceforge-Site[1]
    - Unzip the installation file and change to subfolder "./bsf4ooRexx/install/linux".
    - Run file "install.sh" as superuser
    - BSF4ooRexx will be installed automatically
- If you were successful, a small pop-up written in BSF4ooRexx should appear.
- Furthermore, there are some interesting sample-files in "/opt/BSF4ooRexx/samples".

2.1.2 Install on Windows

Installation on Windows systems is very similar to a Linux installation.

Step 1: Install ooRexx

- Download the Windows installer from the ooRexx Sourceforge site.
- The Windows package is named like "ooRexx-**ver**.windows.x86_64.exe" or "ooRexx-**ver**.windowsx86_32.exe"
- Attention: *Rexx bitness must be the same as Java bitness*
- After the download has finished, run the ooRexx-Setup.
- A graphical user interface will guide you through the installation

Step 2: Install Java (if not yet installed)

- Download the current version of Java Runtime Environment
- If you install Java after ooRexx, take care of using the same bitness.
- Again, a GUI will guide you through JAVA installation procedure

Step 3: Install BSF4ooRexx

- Download installation archive from BSF4ooRexx project site on Sourceforge[1]
- Unzip the archive and change to subfolder ".\bsf4ooRexx\install\Windows"
- Run program "install.cmd"
- If you were successful, a small pop-up written in BSF4ooRexx should appear

2.2 Install JDBC driver

- Download the JDBC driver for your RDBMS (see section 'Useful Links')

2.2.1 Setting CLASSPATH-Variable in Linux

- In order to enable the bridge between BSF4ooRexx and Java, the Java Classpath-Variable and the paths to the JDBC-drivers have to be set
- In Linux systems, set the path in the startup file
    - cf. "Update the PATH Variable (Solaris and Linux)"at Oracle's Java Tutorials[10]
- Insert the path to JAVA_HOME (if necessary)
- Insert the path to your JDBC-drivers

2.2.2 Setting CLASSPATH-Variable in Windows

- Go to 'System properties' >> Change to tab 'Advanced' >> Click button 'Environment Variables'
- In column 'Variables' select 'PATH' and click 'Edit'-button.
- Insert the path to your Java binary files (if necessary)
- Insert the path to your JDBC-drivers.[10]

3 Basic Formula

To get all of the following recipes working, it's necessary to use one basic formula for the database connection in each program.

This works more or less for all three types of databases in the same way.
Just pay attention to the following points:

- There is no user authentication in SQLite
- In MySQL, backticks have to be used WITHIN the double-quotes, if your table or database contains special characters
- At the end of every programming example, there must be one line with "::REQUIRES BSF.CLS"
- Depending on your operating system, the database URL may has to be changed

3.1 For MySQL

```
1  ::ROUTINE db_conn public
2
3  /* Define your user-id and password to access the MySQL-DB*/
4  uid = "yourUserID"
5  pw = "yourPassword"
6  /*Define the url to the database*/
7  -- for example for MySQL via jdbc on default-port 3306
8  url = "jdbc:mysql://localhost:3306/rexx_db";
9  /*Define the name of SQL-table, which contains the data (if
   neccessary)*/
10 table = "`veg_table`"
11
12 /*Get the JDBC-Driver */
13 mydriver=.bsf~new('com.mysql.jdbc.Driver')
14 man=bsf.loadClass("java.sql.DriverManager")
15 man~registerDriver(mydriver)
16
17 /*Building the Connection*/
18 conn=man~getConnection(url, uid, pw)
19 statement=conn~createStatement
20
21 tmpColl = .array ~of(table, statement)
22
23 RETURN tmpColl
24 ::REQUIRES BSF.CLS
```

Source code 1: program 'jdbc-mysql.rxj'

In the code-snippet the connection-process is declared as a public routine called 'db_conn'. UserID, password and url are assigned to variables and passed on when building a connection.

As you can see, in this case, the database is located on localhost and is named 'rexx_db'. When the connection is built and new statement[11] is created, the statement and the variable for the table are saved in an ooRexx-array (line 21).

If an external program calls on this routine ('db_conn'), the 'tmpColl' variable will be returned.

This allows us connecting to the database in a various number of programs without the effort of retyping always the same lines of code.

NOTE: Pay attention to Line 24: Without the functions from 'BSF.CLS' this program won't work.

3.2 For PostgreSQL

```
 1 ::ROUTINE db_conn public
 2
 3 /*Establish connection to postgreSQL-database */
 4
 5 uid = "yourUserID"
 6 url = "jdbc:postgresql://localhost/rexx_db"
 7 pw = "yourPassword"
 8 table = "veg_table"
 9 mydriver=.bsf~new('org.postgresql.Driver')
10 man=bsf.loadClass("java.sql.DriverManager")
11 man~registerDriver(mydriver)
12 conn=man~getConnection(url, uid, pw)
13 statement=conn~createStatement
14
15 tmpColl = .array ~of(table, statement)
16
17 RETURN tmpColl
18 ::REQUIRES BSF.CLS
```

Source code 2_ program 'jdbc-postgresql.rxj'

The basic formula for PostgreSQL is largely the same as for MySQL.

The only thing to change is the URL (line 6) and the JDBC driver class.

3.3 For SQLite

```
 1 ::ROUTINE db_conn public
 2
 3 /*Establish connection to postgreSQL-database */
 4
 5 url = "jdbc:sqlite://home/testi/SQLite/rexx_db";
 6 table = "veg_table"
 7 mydriver=.bsf~new('org.sqlite.JDBC')
 8 man=bsf.loadClass("java.sql.DriverManager")
 9 man~registerDriver(mydriver)
10 conn=man~getConnection(url)
11 statement=conn~createStatement
12
13 tmpColl = .array ~of(table, statement)
14
15 RETURN tmpColl
16 ::REQUIRES BSF.CLS
```

Source code 3: program 'jdbc-sqlite.rxj'

Because of the missing user authentication, the basic formula for SQLite databases differs more from the codes above. Apart from that, the only things changed are the URL and the JDBC driver class.

4 Starters

When preparatory work is done, we are able to start the culinary trip into the SQL-BSF4ooRexx-World.

First of all, to make you familiar with the topic, I'll serve you some light and crisp starters.

4.1 Pre-defined table

The Starters are based on a pre-defined demo-table called 'veg-table'.

It has the following structure:

```
mysql> show columns from veg_table;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| veg_no    | int(11)     | NO   | PRI | NULL    | auto_increment |
| veg_label | varchar(50) | NO   |     | NULL    |                |
| veg_color | varchar(20) | NO   |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0,00 sec)
```

Figure 3: structure of table 'veg_table' in MySQL

If you are working on PostgreSQL, use SERIAL as datatype instead of INTEGER.

For SQL script see Appendix B – SQL scripts.

The table 'veg-table' is pre-filled with following records:

```
+--------+-------------+-----------+
| veg_no | veg_label   | veg_color |
+--------+-------------+-----------+
|      1 | carrot      | orange    |
|      2 | tomato      | red       |
|      3 | sweet corn  | yellow    |
|      4 | chili       | red       |
|      5 | cucumber    | green     |
|      6 | pepper      | red       |
|      7 | lettuce     | green     |
|      8 | aubergine   | purple    |
|      9 | avocado pear| green     |
+--------+-------------+-----------+
```

Figure 4: pre-filled records in 'veg_table' in MySQL

4.2 Starter No. 1 – fine SELECTion of vegetables

After doing the preparatory work and database setup, I would say: Let's start cooking!

To show all the entries of the SQL table called 'veg_table', use the "SELECT"-Statement:

```
 1 tmpColl = db_conn()
 2 table = tmpColl[1]
 3 Statement = tmpColl[2]
 4
 5 /* Show all entries of "veg-table" in MySQL */
 6
 7 rSet=Statement~executeQuery("SELECT * FROM " table ";")
 8 SAY .endofline || "Showing all results of veg-table"
 9 SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
10 DO WHILE rset~next
11 SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
12 END
```

Source code 4: Select statement in program 'starters-mysql.rxj'

"rSet" is the variable in which the Result-Set is saved.

It is querying all columns from the table.

The DO WHILE clause says that the instructions it contains, are being repeated as long as the actual rSet-object has a successor.

In Line 7, the program gets the string of the specific columns and displays it in a terminal.

Info: This example works for the other databases i.e. PostgreSQL and SQLite in the same way.

4.3 Starter No. 2 – Be more picky

Due to the fact, that ID-Numbers and colours are not so interesting, we only want to show the column "veg_label" by doing the following:

```
15 /* Only show specific column of "veg-table" in MySQL */
16
17 rSet=Statement~executeQuery("SELECT `veg_label` FROM" table ";")
18 SAY .endofline || "Showing only column veg_label"
19 SAY left("VEG-LABEL", 15)
20 DO WHILE rset~next
21 SAY left(rset~getString("veg_label"), 15)
22 END
```

Source code 5: Select statement with specified column in 'starters-mysql.rxj'

As you may noticed, this is the same program code as in “Starter No. 1”, with the exception of Line 17 where the SQL-Statement is defined.
That means, you can reuse the code of “Starter No. 1” and only replace the SQL statement in Line 3 with the newer one.

4.3.1 About backticks and co.

If your fieldnames contain special characters or reserved keywords, pay attention to the following hints:

MySQL	PostgreSQL & SQLite

• Use backticks around fieldnames	• In general, use double-quotes around fieldnames
	• Problem: The SQL statements are already double-quoted.
	  • The Rexx interpreter thinks that statement is finished.
	• Solution:
	  • Use two doube-quotes around fieldnames
	  • Or use single-quotes for rexx instruction
	  • e.g. 'SELECT "veg_label" FROM'

Table 1: escape characters MySQL vs. PostgreSQL & SQLite

To check' this out, look at the code-snippet of "Starter No. 2" for PostgreSQL and SQLite below.

```
1 /* Only show specific columns of "veg_table" */
2
3 rSet=Statement~executeQuery("SELECT ""veg_label"" FROM "|| table ||";")
```

Source code 6: Select statement with specified column in 'starters-postgresql.rxj' and 'starters-sqlite.rxj'

4.4 Starter No. 3 – Choose your favourite colour

Now let's have yet another view of our veg_table.

In case of someone doesn't like green vegetable, we only want to query red vegetable:

MySQL:

```
25 /* Only show results with specific "veg-color" in MySQL */
26
27 rSet=Statement~executeQuery("SELECT * FROM" table "WHERE `veg_color`=
   'red' ;")
28
29 SAY .endofline || "Showing only veg with red color"
30 SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
31 DO WHILE rset~next
32 SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
33 END
```

Source code 7: Select statement with WHERE clause in program 'starters-mysql.rxj'

PostgreSQL & SQLite:

```
17 rSet=Statement~executeQuery("SELECT ""veg_label"" FROM" table";")
```

Source code 8: Select statement with WHERE clause in program 'starters-postgresql.rxj'

4.5 Starters together

You may create a new program for every starter or do it like me and put them together in one file.

Therefore, you will find below again the lines of code, but in the same form as in the Appendix.

To shorten it up, I'll now leave out the middle part of the code

```
 1 tmpColl = db_conn()
 2 table = tmpColl[1]
 3 Statement = tmpColl[2]
 4
 5 /* Show all entries of "veg-table" in MySQL */
 6
 7 rSet=Statement~executeQuery("SELECT * FROM " table ";")
 8 SAY .endofline || "Showing all results of veg-table"
 9 SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
10 DO WHILE rset~next
11 SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
12 END
13
14
..
..
..
..
..
34
35
36 ::REQUIRES 'jdbc-mysql.rxj'
37 ::REQUIRES BSF.CLS
```

Source code 9: first and last lines of program 'starters-mysql.rxj'

The most important things are at the beginning and at the end of the program.

In line 1, the public routine 'db_conn' (from the basic formula) is called and the object returned is assigned to variable 'tmpColl'.

The first value of this array is assigned to the variable 'table', the second to 'Statement'.

Again, the program requires the 'BSF.CLS' class PLUS the file, in which the routine 'db_conn' can be found (for example 'jdbc-mysql.rxj').

Running this program, will lead to following output:

```
Showing all results of veg-table
 VEG-NO VEG-LABEL       VEG-COLOR
      1 carrot          orange
      2 tomato          red
      3 sweet corn      yellow
      4 chili           red
      5 cucumber        green
      6 pepper          red
      7 lettuce         green
      8 aubergine       purple
      9 avocado pear    green

Showing only column veg_label
VEG-LABEL
carrot
tomato
sweet corn
chili
cucumber
pepper
lettuce
aubergine
avocado pear

Showing only veg with red color
 VEG-NO VEG-LABEL       VEG-COLOR
      2 tomato          red
      4 chili           red
      6 pepper          red
```

Figure 5: Output of 'starters-mysql.rxj'

5 Main dishes

In this section, you will find four fine selected main courses.

The first one, yet includes DML statement only.

All the following recipes are mostly dealing with DDL statements.

5.1 First main course

The first main course contains a few short SQL-Statements served alongside with a small Rexx-function as a side dish.

5.1.1 INSERT a new table entry

First of all, add a new entry by using the “INSERT INTO” – Statement.

If you are working on a MySQL table, it looks like the following:

```
5 /* Add a new vegetable to the "veg-table" */
6 rSet = Statement~executeUpdate("INSERT INTO" table "VALUES (DEFAULT,
  'potatoes', 'white');")
```

Source code 10: INSERT statement in program ‘2nd-maincourse-mysql.rxj’

5.1.1.1 PostgreSQL

In PostgreSQL the example from above works in the same way.

In the example below it is additional being declared in which rows we want to insert the data.

```
6 /* Add a new vegetable to the "veg-table" */
7 rSet = Statement~executeUpdate("INSERT INTO" table "(veg_label,
  veg_color) VALUES ('potatoes', 'white');" )
```

Source code 11: INSERT statement in program ‘1st-maincourse-postgresql.rxj’

5.1.2 Do a Colour UPDATE

After adding the previous entry, we noticed that our potatoes are yellow coloured, not white.

So we have to change this entry in the database.

MySQL:

```
12 /* Change color of potatoes to yellow */
13 rSet = Statement~executeUpdate("UPDATE" table "SET `veg_color`='yellow'
   WHERE `veg_label`='potatoes';")
```

Source code 12: UPDATE statement in program '1st-maincourse-mysql.rxj'

PostgreSQL & SQLite:

```
13 /* Change color of potatoes to yellow */
14 rSet = Statement~executeUpdate("UPDATE" table "SET veg_color='yellow'
   WHERE veg_label='potatoes';")
```

Source code 13: INSERT statement in program '1st-maincourse-postgresql.rxj' and '1st-maincourse-sqlite.rxj'

5.1.3 DELETE carefully

As we do not necessarily need the potatoes in our database, we can delete them again.

MySQL:

```
19 /* Delete all entries for potatoes in "veg-table" */
20 rSet = Statement~executeUpdate("DELETE FROM" table "WHERE `veg_label` =
   'potatoes';")
```

Source code 14: DELETE FROM statement in program '1st-maincourse-mysql.rxj'

5.1.4 Get table content

To make the process above traceable within one file, it would be necessary to write a SELECT-query after each SQL-instruction.

As I always get tired of writing the same code multiple times, I created a small Rexx function named "getTable":

```
28 getTable:
29 rSet=Statement~executeQuery("SELECT * FROM " table "ORDER BY veg_no
   ASC;")
30
31 SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
32 DO WHILE rset~next
33 SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
   15) left(rset~getString("veg_color"), 10)
34 END
35
36 RETURN
```

Source code 15: method 'getTable' in program '1st-maincourse-mysql.rxj'

It works for all of the demonstrated databases (i.e. MySQL, PostgreSQL and SQLite).

5.1.5 Finishing first main course

Running the first main course leads to following output:

```
Check whether INSERT was successful
 VEG-NO VEG-LABEL       VEG-COLOR
      1 carrot          orange
      2 tomato          red
      3 sweet corn      yellow
      4 chili           red
      5 cucumber        green
      6 pepper          red
      7 lettuce         green
      8 aubergine       purple
      9 avocado pear    green
     10 potatoes        white

Check whether UPDATE was successful
 VEG-NO VEG-LABEL       VEG-COLOR
      1 carrot          orange
      2 tomato          red
      3 sweet corn      yellow
      4 chili           red
      5 cucumber        green
      6 pepper          red
      7 lettuce         green
      8 aubergine       purple
      9 avocado pear    green
     10 potatoes        yellow

Check whether DELETE was successful
 VEG-NO VEG-LABEL       VEG-COLOR
      1 carrot          orange
      2 tomato          red
      3 sweet corn      yellow
      4 chili           red
      5 cucumber        green
      6 pepper          red
      7 lettuce         green
      8 aubergine       purple
      9 avocado pear    green
```

Figure 6: Output of '1st-maincourse-mysql.rxj'

5.2 Second main course

5.2.1 ALTER table

The second main course starts with a basic change in the “veg_table”.

To be more precise, add a column by using the “ALTER TABLE” clause, which is the first DDL statement in this cookbook.

```
 6 /* Add a new column to "veg-table" */
 7 rSet=Statement~executeUpdate("ALTER TABLE" table "ADD grows_in_AT
   boolean;")
```

Source code 16: ALTER TABLE statement in program ‘2nd-maincourse-mysql.rxj’

5.2.2 UPDATE table

As already mentioned, the DDL statements only deal with the database schemas, not with the data stored in a database.

So we need to put data in our new column by updating already existing records.

```
10 /* Insert VALUES into new column */
11 rSet =Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '1';")
12
13
14 /* Change VALUES of some specific records */
15 rSet = Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '0'
   WHERE veg_no = 4 OR veg_no = 5 OR veg_no = 8 OR veg_no = 9;")
```

Source code 17: UPDATE table statement in program ‘2nd-maincourse-mysql.rxj’

The statement in Line 11 sets the value for ‘grows_in_AT’ for all existing records to TRUE.

Statement in Line 15 changes specific records to FALSE.

In the end the ‘veg_table’ should look like that:

```
Showing all results of veg-table
 VEG-NO VEG-LABEL       VEG-COLOR  Grows in AT
      1 carrot          orange     true
      2 tomato          red        true
      3 sweet corn      yellow     true
      4 chili           red        false
      5 cucumber        green      false
      6 pepper          red        true
      7 lettuce         green      true
      8 aubergine       purple     false
      9 avocado pear    green      false
testi@Ubuntu-Vbox:~/Schreibtisch/MySQL$
```

Figure 7: output of ‘2nd-maincourse-mysql.rxj’

5.3 Third main course – Self-made

The third main course works with a self-made table, which does not yet exist and has to be created in the first step.

5.3.1 CREATE TABLE

First of all, a table labelled “taste_table” must be created.

It contains two columns with following properties:

Column name	**taste_id**	**taste_label**
Data type	INT	varchar(20)
NOT NULL	TRUE	FALSE
	PRIMARY KEY	
	AUTO_INCREMENT	

Table 2: structure of new table 'taste_table'

The column “taste_id” should be serial number and also act as the primary key for the table.

5.3.1.1 CREATE TABLE in MySQL

```
5 /* Create new table in rexx-db */
6 rSet = Statement~executeUpdate("CREATE TABLE `taste_table` (`taste_id`
  INT NOT NULL AUTO_INCREMENT PRIMARY KEY, `taste_label` varchar(20) )")
```

Source code 18: CREATE statement in program ‘selfmade-mysql.rxj’

5.3.1.2 CREATE TABLE in PostgreSQL

```
5 /* Create new table in rexx_db */
6 rSet = Statement~executeUpdate("CREATE TABLE taste_table (taste_id
  SERIAL NOT NULL PRIMARY KEY, taste_label varchar(20) );")
```

Source code 19: CREATE statement in program ‘selfmade-postgresql.rxj’

In PostgreSQL use datatype ‘SERIAL’ and omit the ‘AUTO_INCREMENT’ for column ‘taste_id’.

5.3.1.3 CREATE TABLE in SQLite

```
 5 /* Create new table in rexx_db */
 6 rSet = Statement~executeUpdate("CREATE TABLE taste_table (taste_id
   integer NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, taste_label
   varchar(20) );")
```

Source code 20: CREATE statement in program 'selfmade-sqlite.rxj'

In SQLite, additionally, use the 'UNIQUE' attribute for column "taste_id".

5.3.2 RENAME table

Now, rename table "taste_table" to "taste" and redefine variable "table" to the new table name.

5.3.2.1 Rename table in MySQL

```
 9 /* Rename table */
10 rSet = Statement~executeUpdate("RENAME TABLE `taste_table` TO `taste`")
11
12 /*Change variable table to "taste"*/
13 table ="taste"
```

Source code 21: RENAME statement in program 'selfmade-mysql.rxj'

5.3.2.2 Rename table in PostgreSQL & SQLite

In both, PostgreSQL and SQLite the RENAME instruction differs from that in MySQL.

Here you have to use the "ALTER TABLE" instruction, followed by the "RENAME TO" instruction.

```
 9 /* Rename table */
10 rSet = Statement~executeUpdate("ALTER TABLE taste_table RENAME TO
   taste;");
11
12 /*Change variable table to "taste"*/
13 table = "taste"
```

Source code 22: RENAME statement in program 'selfmade-postgresql.rxj' and 'selfmade-sqlite.rxj'

5.3.3 Fill table with records

Once again, the table must be filled with records.

We are going to fill the column 'taste_label' with a few flavours.

In a second step, verify that the table was created successfully.

```
16 /* Fill with records */
17 rSet = Statement~executeUpdate("INSERT INTO taste (taste_label) VALUES
   ('sweet'), ('sour'), ('hot'), ('spicy');")
18
19 /*Check whether table was created successfully */
20 rSet=Statement~executeQuery("SELECT * FROM" table ";")
21
22 SAY "Showing new created table 'taste'"
23 SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
24 DO WHILE rset~next
25 SAY right(rset~getString("taste_id"), 9)
   left(rset~getString("taste_label"), 12)
26 END
```

Source code 23: INSERT statement in program 'selfmade-mysql.rxj'

5.3.3.1 Fill table in SQLite

In SQLite, use NULL instead of DEFAULT for the omitted column "taste_id", which is auto increment.

```
16 /* Fill with records */
17 rSet = Statement~executeUpdate("INSERT INTO taste (taste_id,
   taste_label) VALUES (NULL, 'sweet'), (NULL, 'sour'), (NULL, 'hot'),
   (NULL, 'spicy');")
```

Source code 24: INSERT statement in program 'selfmade-sqlite.rxj'

5.3.4 TRUNCATE table

Because vegetable usually cannot be classified into strict flavours, we will TRUNCATE all records of table “taste” again and verify the success of this action.

```
29 /* Use TRUNCATE to delete all entries of table "taste" */
30 Statement~executeUpdate("TRUNCATE TABLE" table ";")
31
32 /*Check whether records have been dropped successfully*/
33 rSet=Statement~executeQuery("SELECT * FROM" table ";")
34
35 SAY .endofline "Showing table 'taste' without records"
36 SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
37 DO WHILE rset~next
38 SAY right(rset~getString("taste_id"), 9)
   left(rset~getString("taste_label"), 12)
39 END
```

Source code 25: TRUNCATE statement in program ‘selfmade-mysql.rxj’

5.3.4.1 DELETE records in SQLite

In SQLite, use the DELETE-statement to wipe all table records.

```
30 /* Use DELETE FROM to delete all entries of table "taste" */
31 Statement~executeUpdate("DELETE FROM "|| table ||";")
```

Source code 26: DELETE statement in program ‘selfmade-sqlite.rxj’

5.3.5 DROP table

With no records, the table “taste” is quite useless and can be deleted completely:

```
42 /*USE DROP TABLE to delete table "taste" itself */
43 Statement~executeUpdate("DROP TABLE" table ";")
```

Source code 27: DROP TABLE in program ‘selfmade-mysql.rxj’

5.3.6 Output of the self-made course

If all instruction of the third main course are packed in one program, the output should be following:

Figure 8: output of program 'selfmade-mysql.rxj'

6 Bonus sweet - GUI

As a bonus sweet, I created a small GUI using JAVA Swing to test the JDBC connection.

```
 1 userData = .directory~new -- a directory which will be passed to Rexx
   with the event
 4 rexxCloseEH = .RexxCloseAppEventHandler~new
 5 rpCloseEH = BsfCreateRexxProxy(rexxCloseEH, ,
   "java.awt.event.WindowListener")
 6 userData~rexxCloseEH=rexxCloseEH -- save Rexx event handler for later
   use
 9 --create Flow-Layout
10 layout = .bsf~new("java.awt.FlowLayout")
11
12 --create Frame
13 frame = .bsf~new("javax.swing.JFrame", "JDBC Database Connection Test")
14 frame~addWindowListener(rpCloseEH) -- add RexxProxy event handler
15 frame~setLayout(layout)
16
18 -- create label and button
20 lblInstruction = .bsf~new("javax.swing.JLabel", "By clicking the OK-
   Button, I will start testing your connection to the database. ")
21
22 btnOK = .bsf~new("javax.swing.JButton", "OK");
23 frame~getRootPane~setDefaultButton(btnOK) --set btnOK as default button
24
26 --add a ActionListener to the button
27 OkEh = BsfCreateRexxProxy(.RexxStartEventHandler~new, userData,
   "java.awt.event.ActionListener")
28 btnOK~addActionListener(OkEh);
29
31 --add label and button to frame
32 frame~~add(lblInstruction)~~add(btnOK)
33
34 --show the frame
35 frame~pack()
36 frame~setVisible(.true)
38 userData~frame = frame
39 rexxCloseEH~waitForExit
40
43 /* WindowListener */
44 ::CLASS RexxCloseAppEventHandler
45         ::attribute closeApp
46
47         ::METHOD init
48                 expose closeApp
49                 closeApp = .false;
50
51         ::METHOD windowClosing
52                 expose closeApp
53                 closeApp = .true;
54
55         ::METHOD unknown
56
57         ::METHOD waitForExit
58                 expose closeApp
59                 guard on when closeApp = .true -- blocks (waits) until
   control variable is set to .true
```

Source code 28: Usage of Java Swing in program 'connection_test_gui.rxj'

Most Code of this program is to build the GUI.

By using BSF4ooRexx functionality, it is possible to get access to Java Swing classes and use them like an ooRexx object.

So just to explain in some words, what is happening:

- Line 1 creates a collection of type '.directory' for later use
- Line 4 to Line 6
    - Through BSF4ooRexx, a Rexx Proxy is created, which allows to treat a Rexx object like a Java object.
    - In Detail:
        - The variable 'RexxCloseEH' points to a new instance of ooRexx class 'RexxCloseAppEventHandler' (defined in line 44).
        - Then through the BSF4ooRexx function 'BsfCreateRexxProxy' the variable 'RexxCloseEH' is passed over as an argument to the Java class 'java.awt.event.WindowListener'
        - This is necessary to keep control over Java Swing objects.
            - The class 'RexxCloseAppEventHandler' is watching the program's state – if it is started, if the program should exit or if something unknown happens.
        - Both, the ooRexx proxy and the Rexx app event handler are saved in variables for later use.
- Line 9 to line 23
    - Now the GUI is being defined
    - Every time a Java Component is needed, a new instance of the defined class is created through the BSF4ooRexx function '.bsf~new'
        - For instance, in line 13, a new instance of a Java Swing frame is created, having the Title 'JDBC Database Connection Test'
    - In this way, we create a new instances for the layout, the label, the button.
- Line 27 to line 28
    - Now we have created a button, but it doesn't have any function yet.
    - To add a functionality, we create again a Rexx Proxy. But in this case, for the class 'RexxStartEventHandler'.
    - Then we add this (Java) ActionListener to the button.

- Line 31 to line 39
    - Add components to the frame and make it visible
- Line 44 to line 60
    - Class 'RexxCloseAppEventHandler'
        - Controls the frame and state of the program

The most important class is the '**RexxStartEventHandler'**.

If its method 'ActionPerformed' is called following tasks will be performded:

- The public routine 'db_conn' is called
    - Make sure, the 'jdbc-**db**.rxj' is in the same folder as the GUI program.
- If there were no exception or errors, a message-box will pop-up and by clicking its OK-Button, the program will exit.

```
61 /* ActionListener */
62 ::CLASS RexxStartEventHandler
63
64         ::METHOD actionPerformed
65                 use arg eventObject, slotDir
66                 CALL db_conn
67
68                 .bsf.dialog~messageBox("The JDBC connection to your
   database was built successfully!", ,"information")
69                 slotDir~userData~frame~setVisible(.false);
70
71                 slotDir~userData~rexxCloseEH~closeApp = .true
72         Exit
```

Source code 29: Actionlistener to check' SQL connection in program 'connection_test_gui.rxj'

And as in all examples before, at the end of the program two directives are required:

```
73
74
75
76 ::REQUIRES 'jdbc-mysql.rxj'
77 ::REQUIRES BSF.CLS
```

Source code 30: last lines of program 'connection_test_gui.rxj'

6.1 GUI output

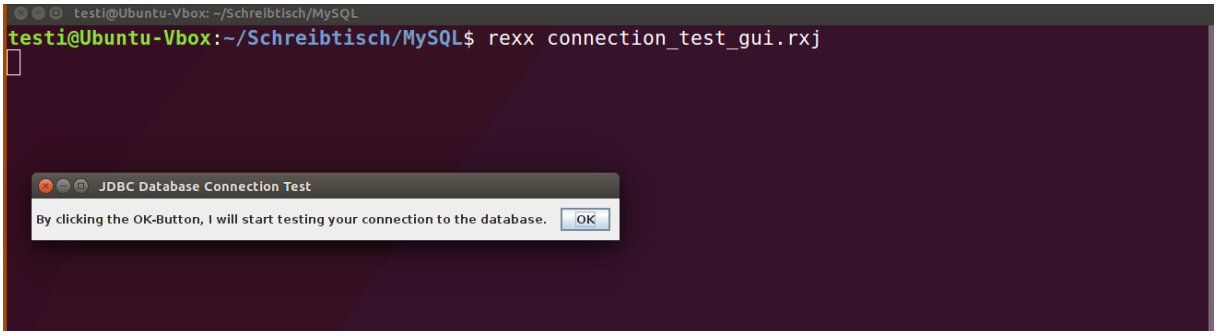

Figure 9: Output 1 of program 'connection_test_gui.rxj'

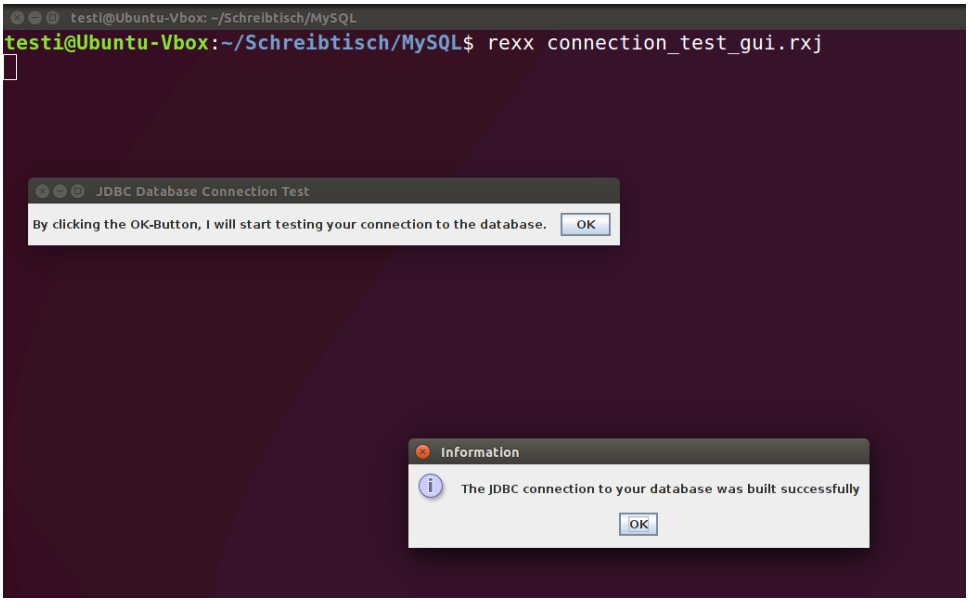

Figure 10: Output 2 of program 'connection_test_gui.rxj'

7 Roundup and outlook

As a conclusion, I can say that working on this project was kind of challenging, but also very interesting and fascinating.
The human-friendly programming language ooRexx with the BSF4ooRexx package is a powerful toolkit to easily automate daily operations.
In particular, it allows an easy and cost-efficient way to automate SQL operations with the option of providing a graphical user interface to the user.
The access to the widespread Java Class Library allows a wide variety of possible application and the combination of them.

This cookbook should allow an easy entry into the world of BSF4ooRexx in SQL environments.
Although, there are still some missing components. For example, there is no error handling in the examples at all. Another critical point is the storing of login information as plaintext. A graphical user interface with a password field or some encryption methods would be necessary for safety operation. These components are not yet included in this cookbook, some of them can be implemented easily on demand and other may have to be developed from the scratch.

Apart from all these technical points of view, I hope you have fun by reading and experiencing this cookbook.

8 Appendix A – ooRexx listings

8.1 'jdbc-mysql.rxj'

```
::ROUTINE db_conn public

/* Define your user-id and password to access the MySQL-DB*/
uid = "yourUserID"
pw = "yourPassword"
/*Define the url to the database*/
-- for example for MySQL via jdbc on default-port 3306
url = "jdbc:mysql://localhost:3306/rexx_db";
/*Define the name of SQL-table, which contains the data (if
neccessary)*/
table = "`veg_table`"

/*Get the JDBC-Driver */
mydriver=.bsf~new('com.mysql.jdbc.Driver')
man=bsf.loadClass("java.sql.DriverManager")
man~registerDriver(mydriver)

/*Building the Connection*/
conn=man~getConnection(url, uid, pw)
statement=conn~createStatement

tmpColl = .array ~of(table, statement)

RETURN tmpColl
::REQUIRES BSF.CLS
```

Based on [12], [13]

8.2 'jdbc-postgresql.rxj'

```
::ROUTINE db_conn public

/*Establish connection to postgreSQL-database */

uid = "yourUserID"
url = "jdbc:postgresql://localhost/rexx_db"
pw = "yourPassword"
table = "veg_table"
mydriver=.bsf~new('org.postgresql.Driver')
man=bsf.loadClass("java.sql.DriverManager")
man~registerDriver(mydriver)
conn=man~getConnection(url, uid, pw)
statement=conn~createStatement

tmpColl = .array ~of(table, statement)

RETURN tmpColl
::REQUIRES BSF.CLS
```

Based on [12]–[14]

8.3 ‘jdbc-sqlite.rxj’

```
::ROUTINE db_conn public

/*Establish connection to postgreSQL-database */

url = "jdbc:sqlite://home/testi/SQLite/rexx_db";
table = "veg_table"
mydriver=.bsf~new('org.sqlite.JDBC')
man=bsf.loadClass("java.sql.DriverManager")
man~registerDriver(mydriver)
conn=man~getConnection(url)
statement=conn~createStatement

tmpColl = .array ~of(table, statement)

RETURN tmpColl
::REQUIRES BSF.CLS
```

Based on [12], [13], [15]

8.4 ‘starters-mysql.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]

/* Show all entries of "veg-table" in MySQL */

rSet=Statement~executeQuery("SELECT * FROM " table ";")
SAY .endofline || "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END


/* Only show specific column of "veg-table" in MySQL */

rSet=Statement~executeQuery("SELECT `veg_label` FROM" table ";")
SAY .endofline || "Showing only column veg_label"
SAY left("VEG-LABEL", 15)
DO WHILE rset~next
SAY left(rset~getString("veg_label"), 15)
END


/* Only show results with specific "veg-color" in MySQL */

rSet=Statement~executeQuery("SELECT * FROM" table "WHERE `veg_color`=
'red' ;")

SAY .endofline || "Showing only veg with red color"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END


::REQUIRES 'jdbc-mysql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.5 ‘starters-postgresql.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]

/* Show all entries of "veg-table" in PostgreSQL */

rSet=Statement~executeQuery("SELECT * FROM " table ";")
SAY .endofline || "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END


/* Only show specific columns of "veg_table" in PostgreSQL */

rSet=Statement~executeQuery("SELECT ""veg_label"" FROM" table";")
SAY .endofline || "Showing only column veg_label"
SAY left("VEG-LABEL", 15)
DO WHILE rset~next
SAY left(rset~getString("veg_label"), 15)
END


/* Only show results with specific "veg-color" in PostgreSQL */

rSet=Statement~executeQuery("SELECT * FROM "|| table ||" WHERE
""veg_color"" = 'red' ;")
SAY .endofline || "Showing only veg with red color"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END


::REQUIRES 'jdbc-postgresql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.6 ‘starters-sqlite.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]

/* Show all entries of "veg-table" in SQLite */
rSet=Statement~executeQuery("SELECT * FROM " table ";")

SAY "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END

/* Only show specific columns of "veg-table" in SQLite */
rSet=Statement~executeQuery("SELECT ""veg_label"" FROM" table ";")

SAY .endofline || "Showing only column veg_label"
SAY left("VEG-LABEL", 15)
DO WHILE rset~next
SAY left(rset~getString("veg_label"), 15)
END


/* Only show results with specific "veg-color" in SQLite */
rSet=Statement~executeQuery("SELECT * FROM "|| table ||" WHERE
veg_color= 'red' ;")

SAY .endofline ||  "Showing only veg with red color"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END


::REQUIRES 'jdbc-sqlite.rxj'
::REQUIRES BSF.CLS
```

[16]

8.7 ‘1st-maincourse-mysql.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]

/* Add a new vegetable to the "veg-table" */
rSet = Statement~executeUpdate("INSERT INTO" table "VALUES (DEFAULT,
'potatoes', 'white');")

SAY "Check whether INSERT was successful"
CALL getTable table


/* Change color of potatoes to yellow */
rSet = Statement~executeUpdate("UPDATE" table "SET `veg_color`='yellow'
WHERE `veg_label`='potatoes';")

SAY .endofline || "Check whether UPDATE was successful"
CALL getTable table


/* Delete all entries for potatoes in "veg-table" */
rSet = Statement~executeUpdate("DELETE FROM" table "WHERE `veg_label` =
'potatoes';")

SAY .endofline || "Check whether DELETE was successful"
CALL getTable table
EXIT



getTable:
rSet=Statement~executeQuery("SELECT * FROM " table "ORDER BY veg_no
ASC;")

SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END

RETURN

::REQUIRES 'jdbc-mysql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.8 ‘1st-maincourse-postgresql.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]


/* Add a new vegetable to the "veg-table" */
rSet = Statement~executeUpdate("INSERT INTO" table "(veg_label,
veg_color) VALUES ('potatoes', 'white');" )

SAY "Check whether INSERT was successful"
CALL getTable


/* Change color of potatoes to yellow */
rSet = Statement~executeUpdate("UPDATE" table "SET veg_color='yellow'
WHERE veg_label='potatoes';")

SAY .endofline || "Check whether UPDATE was successful"
CALL getTable


/* Delete all entries for potatoes in "veg-table" */
rSet = Statement~executeUpdate("DELETE FROM" table "WHERE veg_label =
'potatoes';")

SAY .endofline || "Check whether DELETE was successful"
CALL getTable
EXIT


getTable:
rSet=Statement~executeQuery("SELECT * FROM" table " ORDER BY veg_no
ASC;")

SAY "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END


RETURN

::REQUIRES 'jdbc-postgresql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.9 ‘1st-maincourse-sqlite.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]


/* Add a new vegetable to the "veg-table" */
rSet = Statement~executeUpdate("INSERT INTO" table "(veg_label,
veg_color) VALUES ('potatoes', 'white');" )

SAY "Check whether INSERT was successful"
CALL getTable


/* Change color of potatoes to yellow */
rSet = Statement~executeUpdate("UPDATE" table "SET veg_color='yellow'
WHERE veg_label='potatoes';")

SAY .endofline || "Check whether UPDATE was successful"
CALL getTable


/* Delete all entries for potatoes in "veg-table" */
rSet = Statement~executeUpdate("DELETE FROM "|| table ||" WHERE
veg_label = 'potatoes';")

SAY .endofline || "Check whether DELETE was successful"
CALL getTable
EXIT


getTable:
rSet=Statement~executeQuery("SELECT * FROM" table " ORDER BY veg_no
ASC;")

SAY "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
DO WHILE rset~next
SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10)
END

RETURN

::REQUIRES 'jdbc-sqlite.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.10 ‘2nd-maincourse-mysql.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]


/* Add a new column to "veg-table" */
rSet=Statement~executeUpdate("ALTER TABLE" table "ADD grows_in_AT
boolean;")


/* Insert VALUES into new column */
rSet =Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '1';")


/* Change VALUES of some specific records */
rSet = Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '0'
WHERE veg_no = 4 OR veg_no = 5 OR veg_no = 8 OR veg_no = 9;")


/* Check whether ALTER and UPDATE were successful */
rSet=Statement~executeQuery("SELECT * FROM " table ";")

SAY "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
left("Grows in AT", 12)
DO WHILE rset~next

IF rset~getString("grows_in_AT") = '1'  THEN giAT='true'
                                        ELSE giAT = 'false'

SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10) left(giAT, 12)
END

::REQUIRES 'jdbc-mysql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.11 ‘2nd-maincourse-postgresql.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]


/* Add a new column to "veg-table" */
rSet=Statement~executeUpdate("ALTER TABLE" table "ADD grows_in_AT boolean;")


/* Insert VALUES into new column */
rSet =Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '1';")


/* Change VALUES of some specific records */
rSet = Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '0' WHERE veg_no = 4 OR veg_no = 5 OR veg_no = 8 OR veg_no = 9;")


/* Check whether ALTER and UPDATE were successful */
rSet=Statement~executeQuery("SELECT * FROM " table "ORDER BY veg_no ASC;")

SAY "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10) left("Grows in AT", 12)
DO WHILE rset~next

IF rset~getString("grows_in_AT") = 't'  THEN giAT='true'
                                        ELSE giAT = 'false'

SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"), 15) left(rset~getString("veg_color"), 10) left(giAT, 12)
END
SAY

::REQUIRES 'jdbc-postgresql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.12 ‘2nd-maincourse-sqlite.rxj’

```
tmpColl = db_conn()
table = tmpColl[1]
Statement = tmpColl[2]


/* Add a new column to "veg-table" */
rSet=Statement~executeUpdate("ALTER TABLE" table "ADD grows_in_AT
boolean;")


/* Insert VALUES into new column */
rSet =Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '1';")


/* Change VALUES of some specific records */
rSet = Statement~executeUpdate("UPDATE" table "SET grows_in_AT = '0'
WHERE veg_no = 4 OR veg_no = 5 OR veg_no = 8 OR veg_no = 9;")


/* Check whether ALTER and UPDATE were successful */
rSet=Statement~executeQuery("SELECT * FROM " table ";")

SAY "Showing all results of veg-table"
SAY right("VEG-NO", 7) left("VEG-LABEL", 15) left("VEG-COLOR", 10)
right("Grows in AT", 12)
DO WHILE rset~next

IF rset~getString("grows_in_AT") = '1'  THEN giAT='true'
                                        ELSE giAT = 'false'

SAY right(rset~getString("veg_no"), 7) left(rset~getString("veg_label"),
15) left(rset~getString("veg_color"), 10) right(giAT, 12)
END


::REQUIRES 'jdbc-sqlite.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.13 ‘selfmade-mysql.rxj’

```
tmpColl = db_conn()
Statement = tmpColl[2]


/* Create new table in rexx-db */
rSet = Statement~executeUpdate("CREATE TABLE `taste_table` (`taste_id`
INT NOT NULL AUTO_INCREMENT PRIMARY KEY, `taste_label` varchar(20) )")


/* Rename table */
rSet = Statement~executeUpdate("RENAME TABLE `taste_table` TO `taste`")

/*Change variable table to "taste"*/
table ="taste"


/* Fill with records */
rSet = Statement~executeUpdate("INSERT INTO" table "(`taste_id`,
`taste_label`) VALUES (default, 'sweet'), (default, 'sour'), (default,
'hot'), (default, 'spicy')")

/*Check' whether table was created successfully */
rSet=Statement~executeQuery("SELECT * FROM" table ";")

SAY "Showing new created table 'taste'"
SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
DO WHILE rset~next
SAY right(rset~getString("taste_id"), 9)
left(rset~getString("taste_label"), 12)
END


/* Use TRUNCATE to delete all entries of table "taste" */
Statement~executeUpdate("TRUNCATE TABLE" table ";")

/*Check whether records have been dropped successfully*/
rSet=Statement~executeQuery("SELECT * FROM" table ";")

SAY .endofline "Showing table 'taste' without records"
SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
DO WHILE rset~next
SAY right(rset~getString("taste_id"), 9)
left(rset~getString("taste_label"), 12)
END


/*USE DROP TABLE to delete table "taste" itself */
Statement~executeUpdate("DROP TABLE" table ";")

::REQUIRES 'jdbc-mysql.rxj'
::REQUIRES BSF.CLS
```

Cf. [16]

8.14 ‘selfmade-postgresql.rxj’

```
tmpColl = db_conn()
Statement = tmpColl[2]


/* Create new table in rexx_db */
rSet = Statement~executeUpdate("CREATE TABLE taste_table (taste_id
SERIAL NOT NULL PRIMARY KEY, taste_label varchar(20) );")


/* Rename table */
rSet = Statement~executeUpdate("ALTER TABLE taste_table RENAME TO
taste;");

/*Change variable table to "taste"*/
table = "taste"


/* Fill with records */
rSet = Statement~executeUpdate("INSERT INTO taste (taste_label) VALUES
('sweet'), ('sour'), ('hot'), ('spicy');")

/*Check whether table was created successfully */
rSet=Statement~executeQuery("SELECT * FROM" table ";")

SAY "Showing new created table 'taste'"
SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
DO WHILE rset~next
SAY right(rset~getString("taste_id"), 9)
left(rset~getString("taste_label"), 12)
END


/* Use TRUNCATE to delete all entries of table "taste" */
Statement~executeUpdate("TRUNCATE TABLE" table ";")

/*Check whether records have been dropped successfully*/
rSet=Statement~executeQuery("SELECT * FROM" table ";")

SAY .endofline || "Showing table 'taste' without records"
SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
DO WHILE rset~next
SAY right(rset~getString("taste_id"), 9)
left(rset~getString("taste_label"), 12)
END
SAY


/*USE DROP TABLE to delete table "taste" itself */
Statement~executeUpdate("DROP TABLE "|| table ||";")

::REQUIRES 'jdbc-postgresql.rxj'
::REQUIRES BSF.CLS
```

[16], [17]

8.15 ‘selfmade-sqlite.rxj’

```
tmpColl = db_conn()
Statement = tmpColl[2]


/* Create new table in rexx_db */
rSet = Statement~executeUpdate("CREATE TABLE taste_table (taste_id
integer NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, taste_label
varchar(20) );")


/* Rename table */
rSet = Statement~executeUpdate("ALTER TABLE taste_table RENAME TO
taste;");

/*Change variable table to "taste"*/
table = "taste"


/* Fill with records */
rSet = Statement~executeUpdate("INSERT INTO taste (taste_id,
taste_label) VALUES (NULL, 'sweet'), (NULL, 'sour'), (NULL, 'hot'),
(NULL, 'spicy');")
-- Hint: In SQLite we have to use 'NULL' instead of 'DEFAULT'

/*Check whether table was created successfully */
rSet=Statement~executeQuery("SELECT * FROM" table ";")

SAY "Showing new created table 'taste'"
SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
DO WHILE rset~next
SAY right(rset~getString("taste_id"), 9)
left(rset~getString("taste_label"), 12)
END


/* Use DELETE FROM to delete all entries of table "taste" */
Statement~executeUpdate("DELETE FROM "|| table ||";")
--Hint: In SQLite the command for wiping data is DELETE FROM

/* Check whether records have been deleted successfully */
rSet=Statement~executeQuery("SELECT * FROM "|| table ||";")

SAY .endofline || "Showing table 'taste' without records"
SAY right("TASTE-ID", 9) left("TASTE-LABEL", 12)
DO WHILE rset~next
SAY right(rset~getString("taste_id"), 9)
left(rset~getString("taste_label"), 12)
END
SAY .endofline


/*USE DROP TABLE to delete table "taste" itself */
Statement~executeUpdate("DROP TABLE "|| table ||";")

::REQUIRES 'jdbc-sqlite.rxj'
::REQUIRES BSF.CLS
```

Cf. [16],[18], [19]

8.16 'connection_test_gui.rxj'

```
 1 userData = .directory~new -- a directory which will be passed to Rexx
   with the event
 2
 3
 4 rexxCloseEH = .RexxCloseAppEventHandler~new
 5 rpCloseEH = BsfCreateRexxProxy(rexxCloseEH, ,
   "java.awt.event.WindowListener")
 6 userData~rexxCloseEH=rexxCloseEH -- save Rexx event handler for later
   use
 7
 8
 9 --create Flow-Layout
10 layout = .bsf~new("java.awt.FlowLayout")
11
12 --create Frame
13 frame = .bsf~new("javax.swing.JFrame", "JDBC Database Connection Test")
14 frame~addWindowListener(rpCloseEH) -- add RexxProxy event handler
15 frame~setLayout(layout)
16
17
18 -- create label and button
19
20 lblInstruction = .bsf~new("javax.swing.JLabel", "By clicking the OK-
   Button, I will start testing your connection to the database. ")
21
22 btnOK = .bsf~new("javax.swing.JButton", "OK");
23 frame~getRootPane~setDefaultButton(btnOK) --set btnOK as default button
24
25
26 --add a ActionListener to the button
27 OkEh = BsfCreateRexxProxy(.RexxStartEventHandler~new, userData,
   "java.awt.event.ActionListener")
28 btnOK~addActionListener(OkEh);
29
30
31 --add label and button to frame
32 frame~~add(lblInstruction)~~add(btnOK)
33
34 --show the frame
35 frame~pack()
36 frame~setVisible(.true)
37
38 userData~frame = frame
39 rexxCloseEH~waitForExit
40
41
42
43 /* WindowListener */
44 ::CLASS RexxCloseAppEventHandler
45         ::attribute closeApp
46
47         ::METHOD init
48                 expose closeApp
49                 closeApp = .false;
50
51         ::METHOD windowClosing
52                 expose closeApp
53                 closeApp = .true;
54
```

```
55          ::METHOD unknown
56
57          ::METHOD waitForExit
58                  expose closeApp
59                  guard on when closeApp = .true -- blocks (waits) until
control variable is set to .true
60
61  /* ActionListener */
62  ::CLASS RexxStartEventHandler
63
64          ::METHOD actionPerformed
65                  use arg eventObject, slotDir
66                  CALL db_conn
67
68                  .bsf.dialog~messageBox("The JDBC connection to your
   database was built successfully!", ,"information")
69                  slotDir~userData~frame~setVisible(.false);
70
71                  slotDir~userData~rexxCloseEH~closeApp = .true
72          Exit
73
74
75
76  ::REQUIRES 'jdbc-mysql.rxj'
77  ::REQUIRES BSF.CLS
```

Cf. [13], [16]

9 Appendix B – SQL scripts

9.1 MySQL script for 'veg_table'

```
-- phpMyAdmin SQL Dump
-- version 4.4.13.1deb1
-- http://www.phpmyadmin.net


SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Datenbank: `rexx_db`
--

-- --------------------------------------------------------

--
-- Tabellenstruktur fÃ¼r Tabelle `veg_table`
--

CREATE TABLE IF NOT EXISTS `veg_table` (
  `veg_no` int(11) NOT NULL AUTO_INCREMENT,
  `veg_label` varchar(50) CHARACTER SET latin1 NOT NULL,
  `veg_color` varchar(20) CHARACTER SET latin1 NOT NULL,
  PRIMARY KEY (veg_no)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

--
-- Daten fÃ¼r Tabelle `veg table`
--

INSERT INTO `veg_table` (`veg_label`, `veg_color`) VALUES
('carrot', 'orange'),
('tomato', 'red'),
('sweet corn', 'yellow'),
('chili', 'red'),
('cucumber', 'green'),
('pepper', 'red'),
('lettuce', 'green'),
('aubergine', 'purple'),
('avocado pear', 'green');


/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

9.2 PostgreSQL script for ‘veg_table’

```
--
-- PostgreSQL database dump
--

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;

SET search_path = public, pg_catalog;

SET default_tablespace = '';

SET default_with_oids = false;

--
-- Name: veg_table; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE veg_table (
    veg_no integer NOT NULL,
    veg_label character varying(50),
    veg_color character varying(20)
);


ALTER TABLE veg_table OWNER TO postgres;

--
-- Name: veg_table_veg_no_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

CREATE SEQUENCE veg_table_veg_no_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE veg_table_veg_no_seq OWNER TO postgres;

--
-- Name: veg_table_veg_no_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres
--

ALTER SEQUENCE veg_table_veg_no_seq OWNED BY veg_table.veg_no;


--
-- Name: veg_no; Type: DEFAULT; Schema: public; Owner: postgres
--

```

```
ALTER TABLE ONLY veg_table ALTER COLUMN veg_no SET DEFAULT
  nextval('veg_table_veg_no_seq'::regclass);


--
-- Data for Name: veg_table; Type: TABLE DATA; Schema: public; Owner:
  postgres
--

INSERT INTO veg_table VALUES (1, 'carrot', 'orange');
INSERT INTO veg_table VALUES (2, 'tomato', 'red');
INSERT INTO veg_table VALUES (3, 'sweet corn', 'yellow');
INSERT INTO veg_table VALUES (6, 'pepper', 'red');
INSERT INTO veg_table VALUES (7, 'lettuce', 'green');
INSERT INTO veg_table VALUES (4, 'chili', 'red');
INSERT INTO veg_table VALUES (5, 'cucumber', 'green');
INSERT INTO veg_table VALUES (8, 'aubergine', 'purple');
INSERT INTO veg_table VALUES (9, 'avocado pear', 'green');


--
-- Name: veg_table_veg_no_seq; Type: SEQUENCE SET; Schema: public;
  Owner: postgres
--

SELECT pg_catalog.setval('veg_table_veg_no_seq', 32, true);


--
-- Name: veg_table_pkey; Type: CONSTRAINT; Schema: public; Owner:
  postgres; Tablespace:
--

ALTER TABLE ONLY veg_table
    ADD CONSTRAINT veg_table_pkey PRIMARY KEY (veg_no);


--
-- Name: public; Type: ACL; Schema: -; Owner: postgres
--

REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM postgres;
GRANT ALL ON SCHEMA public TO postgres;
GRANT ALL ON SCHEMA public TO PUBLIC;


--
-- PostgreSQL database dump complete
--

```

9.3 SQLite script for ‘veg_table’

```
BEGIN TRANSACTION;
CREATE TABLE "veg_table" (
        `veg_no`        INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        `veg_label`     TEXT,
        `veg_color`     TEXT
);
INSERT INTO `veg_table` VALUES (1,'carrot','orange');
INSERT INTO `veg_table` VALUES (2,'tomato','red');
INSERT INTO `veg_table` VALUES (3,'sweet corn','yellow');
INSERT INTO `veg_table` VALUES (4,'chili','red');
INSERT INTO `veg_table` VALUES (5,'cucumber','green');
INSERT INTO `veg_table` VALUES (6,'pepper','red');
INSERT INTO `veg_table` VALUES (7,'lettuce','green');
INSERT INTO `veg_table` VALUES (8,'aubergine','purple');
INSERT INTO `veg_table` VALUES (9,'avocado pear','green');
COMMIT;
```

10 Useful Links

MySQL JDBC connector: https://dev.mysql.com/downloads/connector/j/

PostgreSQL JDBC connector: https://jdbc.postgresql.org/download.html

SQLite JDBC connector: https://bitbucket.org/xerial/sqlite-jdbc

11 Bibliography

[1] "BSF4ooRexx - Browse /GA at SourceForge.net." [Online]. Available: http://sourceforge.net/projects/bsf4oorexx/files/GA/. [Accessed: 12-Jan-2016].

[2] R. G. Flatscher, "BSF4ooRexx - Bean Scripting Framework for ooRexx," *WU FIDES*. [Online]. Available: https://bach.wu.ac.at/d/research/projects/1973/#abstract. [Accessed: 27-Dec-2015].

[3] "Rexx," *Wikipedia, the free encyclopedia*. 15-Sep-2015.

[4] "Open Object Rexx." [Online]. Available: http://www.oorexx.org/about.html. [Accessed: 27-Dec-2015].

[5] "MySQL," *Wikipedia, the free encyclopedia*. 17-Dec-2015.

[6] "What are the top MySQL features? What is MySQL?," *SearchITChannel*. [Online]. Available: http://searchitchannel.techtarget.com/feature/What-are-the-top-MySQL-features-What-is-MySQL. [Accessed: 28-Dec-2015].

[7] "SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems," *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Accessed: 12-Jan-2016].

[8] "PostgreSQL," *Wikipedia, the free encyclopedia*. 14-Jan-2016.

[9] "Java - Community Help Wiki." [Online]. Available: https://help.ubuntu.com/community/Java. [Accessed: 03-Feb-2016].

[10] "PATH and CLASSPATH (The Java™ Tutorials > Essential Classes > The Platform Environment)." [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/environment/paths.html. [Accessed: 09-Jan-2016].

[11] "Statement (Java Platform SE 7 b99)." [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html. [Accessed: 01-Dec-2015].

[12] M. Stoppacher, "Project Rome in combination with BSF4ooRexx," Wirtschaftsuniversität Wien, Wien, 2010.

[13] U. Kmon and J. Kirisits, "[Aj2014w] G01 Abschlussprojekt 'PAY-BOY.'" [Online]. Available: http://alice.wu-wien.ac.at/pipermail/aj2014w/2015-February/000070.html. [Accessed: 31-Jan-2016].

[14] "Connecting to the Database." [Online]. Available: https://jdbc.postgresql.org/documentation/80/connect.html. [Accessed: 01-Feb-2016].

[15] "Java JDBC using SQLite/Connecting - Wikibooks, open books for an open world." [Online]. Available: https://en.wikibooks.org/w/index.php?title=Java_JDBC_using_SQLite/Connecting&oldid=2611110. [Accessed: 01-Feb-2016].

[16] "Rexx Runtime Objects." [Online]. Available: http://www.oorexx.org/docs/rexxref/c23113.htm#ENDOFLINE. [Accessed: 01-Feb-2016].

[17] "PostgreSQL Data Type." [Online]. Available: http://www.tutorialspoint.com//postgresql/postgresql_data_types.htm. [Accessed: 01-Feb-2016].

[18] "SQLite - TRUNCATE TABLE Command." [Online]. Available: http://www.tutorialspoint.com/sqlite/sqlite_truncate_table.htm. [Accessed: 01-Feb-2016].

[19] “SQLite Query Language: INSERT.” [Online]. Available: https://www.sqlite.org/lang_insert.html. [Accessed: 01-Feb-2016].