INTRODUCTION TO

SQLJ

- Revised Spring 2005 -

1 What is SQLJ?

SQL for Java (SQLJ) is a way to seamlessly embed SQL syntax directly in the Java programming language. It uses several classes included in the JDBC package but also have some classes of its own. This is done through an additional pre-compiler that interprets the SQLJ specific syntax and creates source files in Java. These files are then compiled as any other Java application through the *javac* compiler.

To be able to compile and execute a Java program that uses the SQLJ framework you need to have three things. First, you need to have the *Java Standard Development Toolkit* (SDK) installed. This environment is free of charge and could be downloaded from: http://java.sun.com/j2se/1.4.2/download.html. Secondly, you need to have the SQLJ pre-compiler (*sqlj.exe*) and the SQLJ Java package (*sqlj.zip*, usually found under *c:\SQLLIB\Java*). Last, you need to have the program *nmake.exe* and the batch-file *embprep.bat* (usually found in *c:\MyProg*).

When writing a SQLJ application you save the source code as a *.sqlj* file. To produce a Java source file out of this you use the pre-compiler. The resulting *.java* file can then be compiled through the default Java compiler *javac*. This produces the *.class* file that could be executed by the virtual machine. The *embprep.bat* file is especially made for your course and need to be modified in case it is used for other purposes.

2 Makefile

A makefile is a set of instructions on how to compile an application. In these instructions you are able to define a sequence of commands that are to be executed when triggering the *nmake* application. Consider the following:

```
crazy:
  cls
  dir
```

Note! Every command must be entered on a single row and started with a tab character.

When executing the command *nmake crazy* in a Windows Command Prompt it'll issue the command *cls* followed by *dir*. It is also possible to declare parameters in a *makefile* by using the commands:

```
DIRNAME=sample
```

DIRNAME is the name of the parameter and *sample* is its value. If you want to use this parameter you type:

```
crazy:
  cls
  dir $(DIRNAME)
```

This would issue the command *cls* followed by *dir sample*

In makefiles it is also possible to use condition constructs. You can define files that have to be available before an attempt to access them is made. Do this by adding the file names according to the example below:

```
crazy : file1.txt file1.bak
  cls
  dir $(DIRNAMN)
```

If you use the statement *nmake crazy*, the nmake application will try to find *file1.txt*. If the file is missing the application will try to create it according any available instructions. If no such instructions exist it will return a ”Fatal: ’file1.txt’ does not exist – don’t know how to make it” error. The following example shows how to specify instructions for file creation:

```
# Create the constant DN
DN=sample

#Specifying the crazy command
crazy : file1.txt file1.bak
  dir $(DN)

#Instructions to create file1.txt
file1.txt :
  dir >file1.txt

#Instructions to create file1.bak
file1.bak : file1.txt
  copy file1.txt file1.bak
```

This will set the constant *DN* to the value *sample* and test that the files *file1.txt* and *file1.bak* exists before trying to issue the command *dir sample*. If one of the files is missing it will be made by piping a *dir* command into *file1.txt* and by copying this file into *file1.bak*. Every row that starts with the # character is treated as a remark.

3 An example (Appa):

On the location \\db-srv-1\studkursinfo\IS4 vt2005\mysqljtest on the local network you’ll find the file *Appa.sqlj* and the *makefile*. *Appa.sqlj* is the complete source code of this example and there you’ll see all syntax used in this introduction in its context. *Makefile* is the file containing the instructions used together with the nmake program to create your running Java application.

1. First of all, create a new folder (preferably on your d: drive) and name it something convenient like for example *mySQLJ*.
2. Copy the source file *Appa.sqlj* and the makefile from the network to your new folder.

Now, the *Appa.sqlj* file is not compiled yet, as you’ll see if you open it in a text editor like for example notepad. In fact it’s not even a complete Java source file. If you try to compile it through javac all you’ll get is a handful of problems. *Appa.sqlj* is still in its SQLJ specific high level programming language, which is easily readable and meaningful for the human eye.

3. Compile the *.sqlj* file by using the *SQLJ pre*-compiler to create a new *.java* file. Probably the easiest way to do this is by placing yourself in the newly created folder using a *Windows Command Prompt* (in this case *d:\mySQLJ*) and issue the command: **`nmake Appa`**. This will execute the following commands from the makefile:

```
# Use the java compiler
CC= javac

# To connect to another database update the DATASOURCE variable.
# User ID and password are optional. If you want to use them,
# update TESTUID with your user ID, and TESTPWD with your password.
```

```
DATASOURCE=sample
TESTUID=
TESTPWD=

# Build and run the following SQLJ application with these commands:
# By Michael Persson
#      nmake  Appa
#      java  Appa
#

Appa.java : Appa.sqlj
  sqlj Appa.sqlj

Appa.class : Appa.java Appa_SJProfile0.ser

Appa : Appa.class
  $(CC) Appa.java
  embprep Appa $(DATASOURCE) $(TESTUID) $(TESTPWD)
```

To run the application we need to have the *Appa.class* file to address the Java virtual machine. If *Appa.class* is missing we need to have the *Appa.java* and *Appa_SJProfile0.ser* files. *Appa.java* can be created if the *Appa.sqlj* file is available. The three constants that are used (DATASOURCE, TESTUID, TESTPWD), specifies the database information needed in the final application.

The source code specified in the *.sqlj* files is very similar to plain Java. All that differs are the database specific commands which in SQLJ are denoted by a beginning *#sql* markup on each row. These are the rows the pre-compiler turn into Java when turning the file from *.sqlj* to *.java*.

3.1 Explanation of the source code

Open up *Appa.sqlj* to view the application source code. Exactly as the SQLJ pre-compiler, Java compiler and the virtual machine, we'll go through the file, starting at the top.

```
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
…
```

These are statements that include already compiled Java methods and constructs from other packages. The star '*****' states that we shall include all classes in the *sqlj.runtime* and *sqlj.runtime.ref* package paths.

```
…
#sql iterator Appa_Cursor1 (String empno, String firstnme);
#sql iterator Appa_Cursor2 (String);
…
```

These statements define two SQLJ data types that later can be used to define cursor variables. The first one is called *Appa_Cursor1* and the other is called *Appa_Cursor2*. *Appa_Cursor1* define the two string arguments as *empno* and *firstnme* while *Appa_Cursor2* only defines the argument as a string value.

Further down in *Appa.sqlj* is the main method that starts with defining two cursor handles that are defined just like any other Java variable with the *Appa_Cursor* data types we specified in the previous step:

```
…
Appa_Cursor1 cursor1;
```

```
Appa_Cursor2 cursor2;
…
```

The next part handles the database connection. Note the URL *jdbc:db2:sample* specifying the path to the DB2 database *sample*. Also note the use of *DefaultContext* that specifies the database connection and handles the execution of SQL queries, and the *con* variable that supplies a handle to this connection. If the *con* variable is set to auto commit the queries, they are applied directly into the database when carried out. If set to false, an explicit *commit* command has to be issued for the changes to be effective.

```
…
#sql cursor1 = {SELECT empno, firstnme FROM db2admin.employee};
…
```

This select statement has to fetch the same fields as the arguments specified in *Appa_Cursor1*. The fetched rows have to have just as many columns, have the same column names in the exact same order as the arguments given by *Appa_Cursor1*. Using any other construct will fail to retrieve the information we want to collect. Since *Appa_Cursor1* is specified to take two string arguments (*empno* and *firstnme*) using *cursor1* that's specified to this type works just fine. However, using the *cursor2* variable wouldn't work. To pull out the information from the data collection after it has been fetched, we can do the following:

```
…
while (cursor1.next()) {
  str1 = cursor1.empno();
  str2 = cursor1.firstnme();
  …
}
cursor1.close();
…
```

The WHILE-statement iterates through the whole collection of data. Retrieving row after row for each loop by the *next* method and accessing the row's column by specifying a method in the cursor called after their name. When the data collection is exhausted, the *next* method returns false and the *close* method closes the connection to the database.

Simpler SQL queries that return no or only a single row of data don't have to use cursors. Consider the following:

```
…
#sql {UPDATE db2admin.employee SET firstnme ='SHILI' WHERE empno =
'000010'};
…
```

This would update the employee number 000010 by setting his/her first name to SHILI. It's also possible to denote variables inside the query by using the colon qualifier character ':' in front of the variable name. To put the integer value from a select statement into the variable *count1* would look like:

```
…
#sql { SELECT count(*) into :count1 FROM db2admin.employee };
…
```

Using undeclared cursors like the *Appa_Cursor2* with unspecified argument names need another approach. In this case you could use the FETCH statement that iterates through a collection of data and puts the retrieved values into the given variable. In the example below the first names gathered from the select statement are put into the *str2* variable. The *endFetch* method ends the loop when there are no more rows in the data collection:

```
…
str1 = "000010";
#sql cursor2 = { SELECT firstnme from db2admin.employee where empno =
:str1 };
…
while (true) {
  #sql { FETCH :cursor2 INTO :str2 };
  if (cursor2.endFetch()) break;
  …
}
cursor2.close();
…
```

If you like, SQLJ could also perform transaction management. To rollback a transaction, use the command:

```
…
#sql { ROLLBACK work };
…
```

Basically, using SQLJ is designed to be easy. Creating a Java application from your SQLJ file should be as easy as running the command *nmake Appa* to compile it and *java Appa* to execute it. This would look like:

```
D:\mySQLJ>nmake Appa

IBM(R) Program Maintenance Utility for Windows(R)
Version 3.50.000 Feb 13 1996
Copyright (C) IBM Corporation 1988-1995
Copyright (C) Microsoft Corp. 1988-1991
All rights reserved.

        sqlj Appa.sqlj
        javac Appa.java
        embprep Appa sample

[IBM][SQLJ Driver] SQJ0001W Customizing profile "Appa_SJProfile0".
PROFILE NAME:   Appa_SJProfile0
SOURCE PROGRAM: Appa.sqlj
ENTRY   LINE    MESSAGES

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

SQL0060W  The "SQLJ" precompiler is in progress.
SQL0091W  Precompilation or binding was ended with "0"
          errors and "0" warnings.

D:\mySQLJ>java Appa
Retrieve some data from the database...
Received results:
 empno= 000010 firstname= CHRISTINE
 empno= 000020 firstname= MICHAEL
 empno= 000030 firstname= SALLY
 empno= 000050 firstname= JOHN
 empno= 000060 firstname= IRVING
 empno= 000070 firstname= EVA
 empno= 000090 firstname= EILEEN
 empno= 000100 firstname= THEODORE
 empno= 000110 firstname= VINCENZO
 empno= 000120 firstname= SEAN
 empno= 000130 firstname= DOLORES
 empno= 000140 firstname= HEATHER
```

```
 empno= 000150 firstname= BRUCE
 empno= 000160 firstname= ELIZABETH
 empno= 000170 firstname= MASATOSHI
 empno= 000180 firstname= MARILYN
 empno= 000190 firstname= JAMES
 empno= 000200 firstname= DAVID
 empno= 000210 firstname= WILLIAM
 empno= 000220 firstname= JENNIFER
 empno= 000230 firstname= JAMES
 empno= 000240 firstname= SALVATORE
 empno= 000250 firstname= DANIEL
 empno= 000260 firstname= SYBIL
 empno= 000270 firstname= MARIA
 empno= 000280 firstname= ETHEL
 empno= 000290 firstname= JOHN
 empno= 000300 firstname= PHILIP
 empno= 000310 firstname= MAUDE
 empno= 000320 firstname= RAMLAL
 empno= 000330 firstname= WING
 empno= 000340 firstname= JASON

Retrieve the number of rows in employee table...
There are 32 rows in employee table.

Update the database...
Retrieve the updated data from the database...
Received results:

 empno= 000010 firstname= SHILI

Rollback the update...
Rollback done.
```

4 Writing your first SQLJ application

It's greatly recommended to use your own folder under the *d:* drive as a base when creating the application. Try to reuse as much of the *Appa.sqlj* and the *makefile* as possible, modify them and make adjustments rather then starting from scratch. A few points to keep in mind when doing this:

1. The name of the file and the class must be the same. Renaming the class = renaming the file and vice versa.
2. The instructions in the makefile must correspond to the filenames in use.
3. The path to the database and the login information must be correct both in the makefile and in the application.
4. The arguments specified in the declared SQLJ cursors must correspond to the fields in the SQL queries.