INTRODUKTION TILL SQLJ & MAKEFILE

Vad är SQLJ?

SQLJ är ett nytt sätt att skriva embedded SQL i java. SQLJ använder sig av en del av JDBC, som ingår i java, men har också en del egna klasser. Ett SQLJ program är enklare än motsvarande program med endast JDBC. SQLJ program kan också i många fall vara snabbare än JDBC motsvarigheten.

Vad kräver det?

För att kunna kompilera och köra SQLJ program måste man:

- ha **JDK** installerat på datorn.
- ha SQLJ kompilatorn (sqlj.exe) och SQLJ java biblioteket (sqlj.zip) tillgängliga på datorn (Se katalogen **C:\Program Files\SQLLIB**)
- ha programmen **nmake.exe** och **embprep.bat** tillgängliga på datorn (Se katalogen **c:\MyProg**)

När man skriver ett SQLJ program sparar man det i en **.sqlj** fil. Filen måste sedan kompileras med SQLJ kompilatorn, vilket producerar en **.java** fil som kan sedan kompileras med java kompilatorn. När programmet är färdigtkompilerat kan man köra det genom att köra den skapade **.class** filen. Det gör man med kommandot **java** och *klassnamnet.* Kommandot **embprep** länkar programmet med databasen.

Vissa delar av den här konfigurationen är anpassade för kursen och de produkter som används under kursen. Till exempel **embprep.bat** är specifikt för DB2.

Makefile

En makefile innehåller instruktioner om hur ett program kompileras. I makefile kan man definiera en sekvens kommandon som ska exekveras när användaren anropar **make**-programmet. Man kan till exempel definiera följande rader i makefile:

```
crazy :
        cls
        dir
```

Obs! Varje kommando måste skrivas på ny rad och inleds med ett *tabb-tecken*

Om man nu skriver i ett kommando-fönster **nmake crazy**, då exekveras kommandon **cls** och **dir**.

Man kan också använda *konstanter* i en **makefile**. Man kan definiera följande:

```
DIRNAMN=sample
```

DIRNAMN är namnet på konstanten och **sample** är värdet. Om man nu vill utnyttja konstanten kan man skriva:

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

Om man nu skriver i ett kommando-fönster **nmake crazy**, då exekveras kommandon **cls** och **dir sample**.

Det finns också möjligheten att definiera villkor i en makefile. Man kan definiera vilka filer som måste finnas innan man ska exekvera de specificerade kommandona. Det gör man genom att rada upp filnamnen enligt följande:

```
crazy : fil1.txt fil1.bak
        cls
        dir $(DIRNAMN)
```

Skriver man nu **nmake crazy**, så kommer make-programmet att först försöka hitta **fil1.txt**. Om filen inte finns, kommer programmet att försöka skapa den, om den kan hitta instruktioner om hur man skapar **fil1.txt**. Om det inte finns sådana instruktioner i **makefile**, kommer programmet ge meddelandet " **Fatal: 'fil1.txt' does not exist - don't know how to make it**". Därför kan man lägga till instruktioner i **makefile** för att skapa **fil1.txt** och **fil1.bak**. Här är ett sådant exempel:

```
#Definition av konstanten DN
DN=sample

#Instruktioner för crazy
crazy : fil1.txt fil1.bak
        dir $(DN)

#Instruktioner för fil1.txt
fil1.txt :
        dir >fil1.txt

#Instruktioner för fil1.bak
fil1.bak : fil1.txt
        copy fil1.txt fil1.bak
```

Rader som börjar med **#** är kommentarer!

Appa exemplet

Appa är namnet på SQLJ exemplet som finns att ladda ner från

\\Db-srv-1\StudKursInfo\IS4 Ht2001\SQLj\MySQLjTest
(man hittar dit via "Network Neighborhood")

Där finner man följande två filer:

Appa.sqlj och **makefile**

Skapa en egen katalog och kopiera dit filerna!

Appa.sqlj är det okompilerade SQLJ programmet och **makefile** är filen som innehåller kompileringsinstruktioner för **Appa.sqlj**. Dessa instruktioner är följande och finns i makefile:

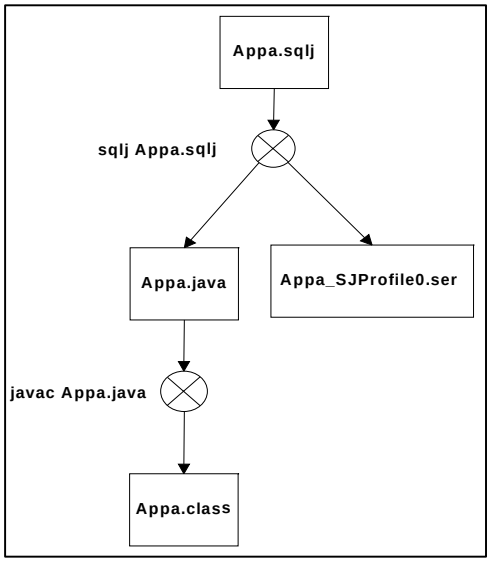
```
# Use the java compiler
CC= javac

# 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)
```

Alltså, för att göra Appa måste det finnas filen **Appa.class**. Om den inte finns, måste **Appa.java** och **Appa_SJProfile0.ser** finnas. Om **Appa.java** inte finns, kan den skapas om **Appa.sqlj** finns. De tre konstanter som används (**DATASOURCE, TESTUID, TESTPWD**), specificerar databasen som Appa programmet jobbar mot. I bilden bredvid ser man vilka filer som skapas med vilka kommandon.



Innehållet av **Appa.sqlj** är väldigt lik java kod. I ett SQLJ program har man möjligheten att använda så mycket java man vill, och ytterligare har man en del extra kommandon tillgängliga. Dessa är databasrelaterade kommandon. Alla kommandon / rader som innehåller

SQLJ syntax börjar alltid med **#sql**. Sådana rader kan till exempel definiera **cursors** och **variabler**, som sedan kan refereras i java som vanliga variabler.
Här är en förklaring av de viktigaste SQLJ kommandon, baserat på innehållet av **Appa.sqlj**:

I början av programmet finns följande två rader:

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

De måste alltid finns med för att SQLJ programmet ska kunna fungera. De specificerar och importerar de två specifika klassbibliotek för SQLJ.

Appa.sqlj fortsätter med följande:

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

Med de här raderna definierar man två typer, som kan sedan användas för att definiera **cursor** variabler. Den första typen heter **Appa_Cursor1** och har två fält. Fältena heter **empno** och **firstnme**, och är båda strängar (av typ **String**). Den andra typen heter **Appa_Cursor2** och har bara ett fält av typen **String**. Att fältet inte har något namn betyder att en **cursor** av den typen måste användas lite annorlunda.

Lite längre ner i **Appa.sqlj** finns metoden **main()**. Den börjar med att definiera två **cursor** variabler:

```
Appa_Cursor1 cursor1;
Appa_Cursor2 cursor2;
```

Variablerna heter **cursor1** och **cursor2** och typerna är de två typer som definierades tidigare.

Nästa del har hand om databaskopplingen. Det viktiga här är värdet som url variabeln har:

```
jdbc:db2:sample
```

Det är en DB2 databas med namn **sample** som kopplas via JDBC.

DefaultContext förekommer några gånger i den del av koden som har med databaskopplingen att göra. **DefaultContext** finns i **sqlj.runtime.ref** biblioteket och krävs för att databaskopplingen ska fungera. Man kan ha flera **Context**, en för varje databaskoppling, men alla SQL satser exekveras alltid på den **Context** som har satts som **DefaultContext**, om man inte specificerar en annan **Context**.

Även följande rad har stor betydelse:

```
con.setAutoCommit(false);
```

con är variabeln som håller i databaskopplingen. Om **AutoCommit** är satt till **true** då gäller varje SQL kommando som man kör direkt efter att man har kört det. Sätter man **AutoCommit** till **false** då måste man skicka ett SQL **COMMIT** kommando för att alla andringar ska bli permanenta i databasen. Har man inte gjort **COMMIT**, försvinner alla ändringar i databasen när programmet avslutas. Man kan också använda SQL kommandot **ROLLBACK** om man vill ångra alla ändringar i databasen.

Följande rad jobbar med variabeln **cursor1**:

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

SELECT satsen, som **cursor1** kopplas till, måste producera samma fält (i antal, namn och ordning) som typen **Appa_Cursor1** (typen av **cursor1**) har (alltså **empno** och **firstnme**). Direkt efter den här raden pekar **cursor1** inte på första raden i resultatet. Det är metoden **next()** som flyttar fram en *iterator* (som **cursor1**) i resultatet. För varje kolumn/fält i resultatet finns också en metod. Varje metod returnerar värdet av fältet i den aktuella raden. För cursor1 finns metoden **empno()** och metoden **firstnme()**. Här är några rader från **Appa.sqlj** som använder alla dess metoder:

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

När man är klar med **cursor1** kan man stänga den med metoden **close()**.

För enklare SQL satser som returnerar endast en rad eller inget alls, kan man använda ett enkelt **#sql** kommando. Här är två exempel av sådana kommandon som finns i **Appa.sqlj**:

```
#sql { SELECT count(*) into :count1 from db2admin.employee };
#sql { UPDATE db2admin.employee set firstnme = 'SHILI' where empno
= '000010' };
```

Observera att man kan använda vanliga variabler i SQL satserna genom att endast sätta ett kolon-tecken (**:**) framför variabelns namn (till exempel **count1**).

cursor2 är av typ **Appa_Cursor2**, som är definierad att bara ha ett **String**-fält utan namn. Följande rader visar hur man kan använda en sådan cursor/iterator:

```
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();
```

En **FETCH** sats tar en iterator (**cursor2**) och skickar placerar de aktuella värdena från iteratorn i variabler (**str2**). En **FETCH** sats börjar med att flytta iteratorn till nästa rad. Det kan därför hända att en **FETCH** misslyckas om det inte fanns flera rader. Metoden **endFetch()** returnerar **true** om iteratorn inte pekar på någon rad.

Slutligen kan man från ett SQLJ-program köra andra databaskommandon som till exempel **ROLLBACK**:

```
#sql { ROLLBACK work };
```

Programmet **Appa.sqlj** kan man kompilera med kommandot **nmake Appa**. I exemplet används **M:\jdb038\MySqljProg** som katalogen där **Appa.sqlj** och **makefile** finns:

```
M:\jdb038\MySqljProg>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.
```

För att köra programmet använder vi kommandot **java Appa**:

```
M:\jdb038\MySqljProg>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.
```

Att skriva ett eget SQLJ program!

Det rekommenderade arbetssättet är att man skapar en katalog (till exempel **NySqljProg**) och kopierar dit **makefile** och **Appa.sqlj** som man sedan kan modifiera. Här är en lista med några viktiga punkter som man bör tänka på när man skapar ett nytt SQLJ-program:

- Filnamnet och klassnamnet måste stämma överens
- Det måste finnas instruktioner i **makefile** som matchar programmets namn
- Databasnamnet och inloggningsinformationen måste vara korrekt både i **makefile** och i programmet.
- Namnen på kolumner i iteratorvariabler/cursorvariabler måste stämma överens med namnen i Iteratortypdefinitioner.

Ytterligare information

SQLJ java bibliotekets dokumentation:
http://www.labunix.uqam.ca/oracle/sqlj/doc/runtime/javadoc/packages.html

Generell information om SQLJ:
http://www.sqlj.org/