Breaking the PL/SQL Barrier for Procedures
Calling Java Routines from Validation/Derivation Procedures

Thomas Struzik
DBMS Consulting
12 October 2010
Validation/Derivation Procedures

•Acknowledgements

- Many thanks to the OHSUG for this opportunity to present for the OHSUG Template Focus Group.
- Many thanks to the OHSUG Planning and Review Committee and OHSUG Template Focus Group Chairs for their infinite patience in receiving and expeditious review of this presentation
- Many thanks to everyone who participated in the development of presentation.

Java Stored Procedures

- For years, enterprise customers have introduced customized PL/SQL library functions which can be used across an organizations' validation/derivation procedures multiple times for greater efficiency and code consistency. But Java code is also prevalent in the Oracle RDBMS, and especially in the Oracle 11g RDBMS.
- Do there exist core Java routines and programs embedded in the base Oracle 11g RDBMS which would be useful for validation and derivation procedures, and if so, is it technically possible to call these Java routines and programs?
- This presentation seeks to answer these questions with some practical examples which will hopefully lead to a new way of thinking in building validation/derivation procedure custom libraries.

Why use Java for writing stored procedures and functions?

- Capitalize on the skills of existing Java developers within your organization
- Employ object-oriented and reusable code
- Oracle 11g provides a native Java compiler which enhances performance of Java byte code [1]

[1] 1-12 Oracle Database Java Developer's Guide

Verify that JVM is installed

```
SQL> select * from all_registry_banners;
--------------------------------
JServer JAVA Virtual Machine Release 11.1.0.7.0 - Production
...
```

Step 1. Create Java class

- Java classes can be loaded into an 11g database either as source files (*.java) or class files (*.class)
- Java classes can include any Java libraries (Apache, etc.)
- Classes that are going to actually access SQL data should import both `java.sql.*` and `oracle.jdbc.*`

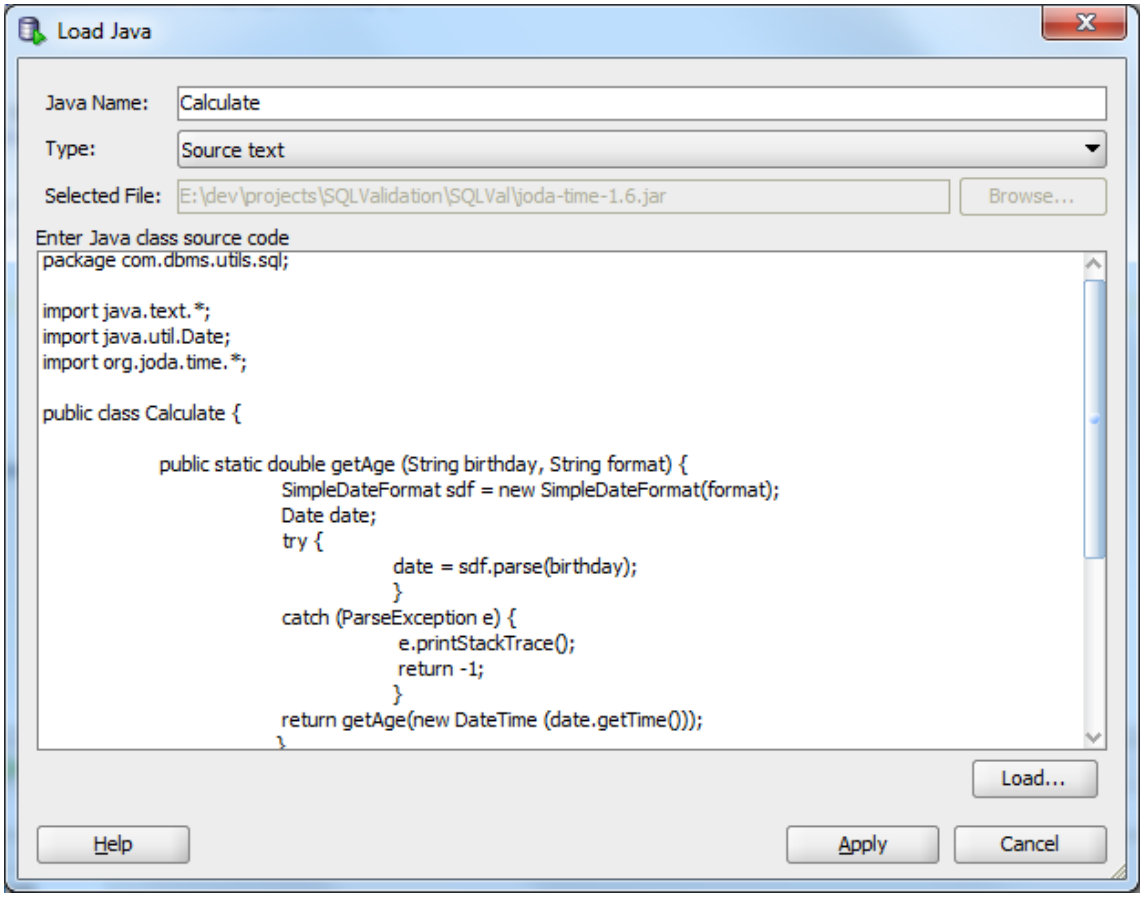
Step 2. Load Java class

- Classes can be loaded either via the `loadjava` utility

```
loadjava: Usage: loadjava [-definer] [-encoding encoding] [-force]
[-genmissing] [-genmissingjar jar] [-grant grants] [-help]
[-nousage] [-noverify] [-oci8] [-order] [-resolve] [-nativecompile]
    [-resolver resolver][-schema schema] [-synonym] [-thin]
[-tableschema schema] [-user user/password@database]
[-verbose] classes..jars..resources..properties...
```

Step 2. Load Java class (cont)

or with the built-in utilities in Oracle SQL Oracle SQL Developer



Step 3. Load required libraries

- Use the loadjava utility to upload any required jar files
- Example:

```
loadjava -schema MY_SCHEMA -user USER_NAME/PASSWORD@SERVICE NAME
   e:\temp\joda-time-1.6.jar
```

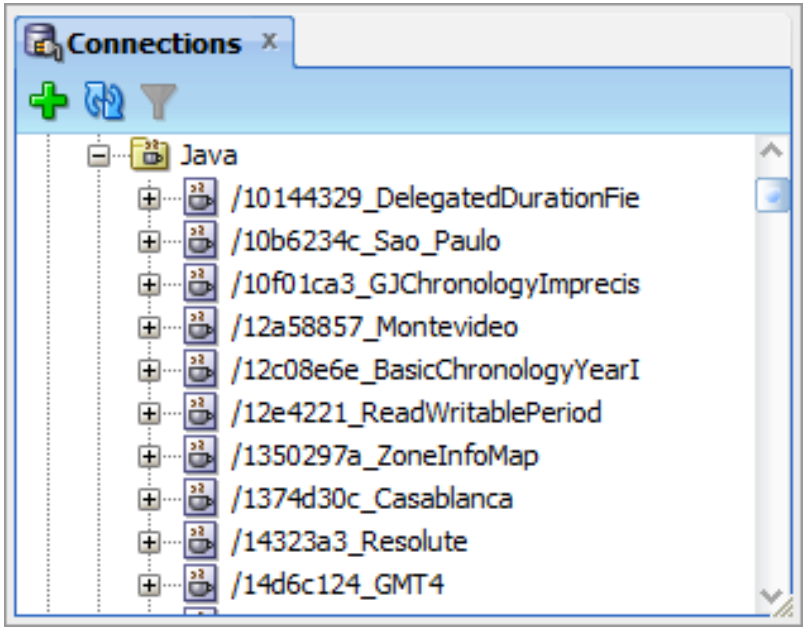
- The above command loads the Joda Java Date and Time API into MY_SCHEMA

Step 3. Load required libraries (cont)

- Classes can be verified either by SQL:

```
SQL> SELECT object_name, object_type, status
FROM USER_OBJECTS WHERE object_type LIKE 'JAVA%';
```

- Or via the Oracle SQL Developer:



Step 4. Publish Java class

- Classes and methods must be made available to the database by wrapping them in an SQL function or procedure

```
SQL> CREATE OR REPLACE FUNCTION <my function>
     RETURN <return value>
     AS LANGUAGE JAVA NAME
   '<qualified class name>.<function name>() return <return value>';
```

Step 5. Call Java class via SQL

- Function or procedure can now be used within SQL calls:

```
SQL> SELECT <MY FUNCTION>(<MY PARAMETERS>) FROM DUAL;
```

Example 1. Validation

- In this example a function has been created that allows for testing values against a regular expression to validate user input.
- Regular expressions are stored in a database table in name-value pairs as shown in *table 1*.
- Function signature
  - SQL
    - `FUNCTION VALIDATOR (val1 IN VARCHAR2, val2 IN VARCHAR2) RETURN BOOLEAN`
  - Java
    - `public static boolean validate(String value, String patternName)`

Example 1. Validation (cont)

NAME	REGEX	VALID VALUES
MIN_MAX_AGE	^[1-9]{1}$|^[1-9]…	1 - 113
PHONE_NUMBER_US	^(?:\([2-9]\d{2}…	5305551212
PHONE_NUMBER_MOBILE_INDIA	^((\+){0,1}91(\s)…	+919847444225
EMAIL_ADDRESS	(\w[-._\w]*\w@\w…	test@test.com
VALID_DATE_TIME	"^(((((0?[13578])|…	04/01/2003 10:01:23 am
NATIONAL_DRUG_CODE	\d{4}-\d{4}-\d{2}…	1234-5678-90

Table 1.
Regular expressions used to valid fields

Example 1. Validation (cont)

Partial code listing

```
public static boolean validate(String value, String patternName) {
            Pattern pattern = Pattern.compile(getPattern(patternName));
            Matcher matcher = pattern.matcher(value);
            return matcher.matches();
            }

    private static String getPattern(String patternName) {
            System.out.println("Looking up pattern: " + patternName);

            String pattern = null;

            try {
                        Connection conn = DriverManager.getConnection("jdbc:default:connection:");
                        conn.setAutoCommit (false);
                        String sql = "SELECT PATTERN_TEXT FROM validation_patterns WHERE PATTERN_NAME = ?";
                        PreparedStatement pstmt = conn.prepareStatement(sql);
                        pstmt.setString(1, patternName);
                        ResultSet rs = pstmt.executeQuery();
                        if (rs.next()) {
                                    pattern = rs.getString("PATTERN_TEXT");
                                    }
                        rs.close();
                        pstmt.close();
                        }
            catch (Exception e) {
                        System.err.println("VALIDATION ERROR: UNKNOWN PATTERN_NAME:" + patternName);
                        e.printStackTrace();
                        }
            System.out.println("Returning pattern: " + pattern);
            return pattern;
            }
```

Example 1. Validation (cont)

Results

```
SQL> select VALIDATOR('49981-*007-01', 'NATIONAL_DRUG_CODE') from
   dual;

VALIDATOR('49981-*007-01','NATIONAL_DRUG_CODE')
-----------------------------------------------
                                              1

SQL> select VALIDATOR('49981*-007-01', 'NATIONAL_DRUG_CODE') from
   dual;

VALIDATOR('49981*-007-01','NATIONAL_DRUG_CODE')
-----------------------------------------------
                                              0
```

Example 1. Validation (cont)

Results

```
SQL> select VALIDATOR('31', 'MIN_MAX_AGE') from dual;

VALIDATOR('31','MIN_MAX_AGE')
-----------------------------
                            1

SQL> select VALIDATOR('3a1', 'MIN_MAX_AGE') from dual;

VALIDATOR('3A1','MIN_MAX_AGE')
------------------------------
                             0

SQL> select VALIDATOR('101', 'MIN_MAX_AGE') from dual;

VALIDATOR('101','MIN_MAX_AGE')
------------------------------
                             1
```

Example 2. Validation

- In this example a function has been created that calculates a person's age using the Joda Date and Time API.
- This demonstrates using a 3rd party API within the Oracle 11g JVM
- Function signature
  - SQL
    - `FUNCTION GET_AGE (val1 IN VARCHAR2, val2 IN VARCHAR2) RETURN NUMBER`
  - Java
    - `public static double getAge (String birthday, String format)`

Example 2. Validation (cont)

Partial code listing

```
public static double getAge (String birthday, String format) {
          SimpleDateFormat sdf = new SimpleDateFormat(format);
          Date date;
          try {
                  date = sdf.parse(birthday);
                  }
          catch (ParseException e) {
                   e.printStackTrace();
                   return -1;
                  }
          return getAge(new DateTime (date.getTime()));
         }

    public static double  getAge(DateTime birthday) {
          DateTime now = new DateTime();
          MutablePeriod mutablePeriod = new
          MutablePeriod(birthday.getMillis(), now.getMillis());
          int years = mutablePeriod.getYears();
          double months = mutablePeriod.getMonths()/12d;
          return Math.round ((years + months)*100.0)/100.0;
          }
```

Example 2. Validation (cont)

Results

```
SQL> select GET_AGE('16-DEC-1974','dd-MMM-yyyy') from dual;

GET_AGE('16-DEC-1974','DD-MMM-YYYY')
------------------------------------
                               35.58

SQL> select GET_AGE('4-MAY-1924','dd-MMM-yyyy') from dual;

GET_AGE('4-MAY-1924','DD-MMM-YYYY')
-----------------------------------
                              86.17
```

Conclusions

- Java provides an alternative to PL/SQL when coding within Oracle 11g. This allows organizations to leverage their Java expertise within the database with minimal effort.
- Java also allows using object-oriented, reusable code that can be shared amongst the organization.

Sources consulted

- http://www.oracle.com/technology/tech/java/jsp/faq_jvm_java_stored_procedures.html
- Oracle® Database 2 Day + Java Developer's Guide 11g Release 1 B28765-01
- Oracle® Database Java Developer's Guide 10g Release 1 (10.1) Part No. B12021-02
- Joda time 1.6 API - http://joda-time.sourceforge.net/
- Feuerstein, Steven and Bill Pribyl. Oracle PL/SQL Programming. Sebastopol: O'Reilly, 2009.
- RegExLib.com - Regular Expression Library

Question and Answers

Thomas Struzik
thomas.struzik@clinicalserver.com
+1(866) 241-3774 x720