CPRG 307 Stored Program Calls and SQL*Loader Information for Assignment 2

Stored Procedures – Called from Java

Use `CallableStatement` when making procedure or function calls to the database. Note that this is done through a JDBC connection.

After a connection has been made, we will call this *conn*, you can then prepare the statement:

```
CallableStatement cstmt = conn.prepareCall("{call procedureName}");
```

This statement can then be executed by:

```
cstmt.execute();
```

If the procedure has parameters (following good programming standards we will assume these are all IN type parameters), the notation is slightly different:

```
CallableStatement cstmt = conn.prepareCall("{call procedureName(?, ?)} ");
```

This would indicate a procedure with two parameter values. The parameter values are then set depending on the data type (look at the `CallableStatement` API for more options):

```
cstmt.setString(1, param1);
cstmt.setString(2, param2);
```

The execution is performed the same as before.

Stored Functions – Called from Java

As with stored procedures, calling stored functions uses `CallableStatement` with a JDBC connection. The syntax is slightly different:

```
CallableStatement cstmt = conn.prepareCall("{? = call functionName}");
```

If this function had IN type parameters, they would be dealt with as noted for stored procedures. With the function return value, and OUT type of parameters, we must register these values:

```
cstmt.registerOutParameter(1, OracleTypes.VARCHAR);
```

Review the `CallableStatement` API for type options. One hint: Oracle does not have a true Boolean data type available. We use it in stored program units, but there is no Boolean database data type. When registering a return value as Boolean, the value that is returned is actually an integer value. The easiest way that I have found is to return VARCHAR2 values and evaluate the return value in Java.

The execution is performed the same as before.

To view the results returned by a function using OracleTypes.VARCHAR, an example would be:

```
char result = cstmt.getString(1).charAt(0);
```

SQL*Loader - General

SQL*Loader is an Oracle utility that loads bulk data from a data file into one or more database tables. To use SQL*Loader, a "control file" needs to be create that will provide information on the data file to load, how the data file records are delimited, and what table(s) this information is going to go into.

For information on SQL*Loader, and the control files, do an internet search. One location is the following (but there are many more):

http://psoug.org/reference/sqlloader.html

The command line call would be:

```
sqlldr userid=username/password control=path/controlfile_name
log=path/filename
```

Replace:

username	– the database user that the data will be loaded with
password	– the password of the database user
path	– the complete directory path where the file is/will be located
controlfile_name	– the name of the control file to be used in the load. Usually with a *ctl* extension
filename	– the name of the log file where load information will be placed. Usually with a *log* extension

An SQL*Loader control file example is below. The record structure presented is identical to what you will be creating for your assignment.

```
LOAD DATA
INFILE '/home/oracle/Desktop/A2/payroll.txt'"STR '
'"
REPLACE
INTO TABLE payroll_load
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(payroll_date DATE "Month dd, yyyy",
 employee_id,
 amount,
 status)
```

Using a Linux environment for the database does require one aspect to be added to the control file that is not needed if the database is in a Windows environment (this is highlighted above). In your assignment, your Java application will build this file so, to build the *INFILE* line, the following structure should be added to your code (where *textFile* is a string variable that holds the path and name of the delimited file to be loaded into the database):

```
"INFILE " + "'" + textFile + "'" + "\"STR '\r\n'\""
```

SQL*Loader – Called From Java

For your assignment, your Java program will initiate the SQL*Loader process which is an external utility to the database the application is connected to.

Using `Runtime.exec()` is a great way to call external processes from Java. Below is an example of how to utilize `Runtime.exec()`.

```
try
{
        Runtime rt = Runtime.getRuntime();
        Process proc = rt.exec(cmd);
        exitValue = proc.waitFor();
}
```

Notes:

cmd	– is a string that holds the command that is to be executed. For this assignment, this would be the complete SQL*Loader command. This would be defined someplace in your code before calling the method.
exitValue	– is an integer that is defined in your code. The waitFor() method, waits for the process that is executing to terminate before going onto the next command. It returns an integer value based on whether the process that was executed was successful, an exit value of 0 means that the process had no errors and everything worked okay. Any other exit value is an indication that there was a problem (this could be a problem with the way SQL*Loader is being called, or with the data being processed). For this assignment, make sure you are getting the 0 exit value or there is a problem with your SQL*Loader command, or how you have created the delimited file.