CSC309 Programming on the Web
Spring 2009

Due: 11:59 PM, March 14th, 2009 (worth 20%)

Overview

In the first assignment, you have developed the front end of the media web site. In this assignment, you will be building the backend (database layer) for the same application. A SQL script is provided for you to create the tables in a PostgreSQL database. Also, a set of stored procedures are provided which implement the various data operations required by the web site. You need to understand the database schema underlying these tables to be able to work with the database and also the stored procedures.

This assignment involves the development of the following components:

1) **DBApp** A Java command line application

This command line application will connect to a pre-determined message queue upon startup, listen for incoming request messages (encoded in xml), execute the request by calling the appropriate stored procedure and return the result to the requester (also encoded in xml). The responses are placed in another pre-determined message queue.

2) **MQDriver** A second Java command line application

This command line application is for testing purposes. You will load a set of XML files from the hard disk (from the directory <xml-requests-log>), en-queue them into the message queue, wait for responses and save these responses on the hard disk (in the directory <xml-responses-log>). Upon startup, the program should automatically load the files from <xml-requests-log> and en-queue them and start listening for responses.

In this assignment, you will be using several software applications and libraries. A description of each follows.

Software

- Database:
  - You will use PostgreSQL database to store the data and stored procedures.
  - PostgreSQL is available for download from http://www.postgresql.org/
  - On CDF, PostgreSQL is installed on dbsrv3.cdf.toronto.edu. Each student has a separate database named csc309h-<username> where <username> stands for your CDF user name. No password is required to connect to the database locally.
  - The JDBC PostgreSQL driver is located in /local/packages/jdbc-postgresql/ of dbsrv3.cdf.
- Message Queue:
  - You will use ActiveMQ as the message broker for the website.
  - ActiveMQ is available for download from http://activemq.apache.org/
  - On CDF, ActiveMQ is installed in /u/csc309h/lib/apache-activemq-5.2.0

    - To connect from Java to your running ActiveMQ, you will need this jar file: /u/csc309h/lib/apache-activemq-5.2.0/lib/activemq-core-5.2.0.jar
    - You will need to start ActiveMQ before sending and receiving messages from/to it. The provided startup files include the ActiveMQ startup scripts (*you will need to change the default 1099 port to your own port. Port numbers are mentioned in the ports file – see working on cdf below*)
    - You will also need to define 2 point-to-point queues in your ActiveMQ instance, one for requests and another for responses. You can do that either from the Java application, or by launching the ActiveMQ console and manually creating these 2 queues.
    - Documentation is here: http://activemq.apache.org/maven/activemq-core/apidocs/ (look under JMS Client – DBApp and MQDriver in this context are both a JMS Client)
    - ActiveMQ tutorial: http://sacrosanctblood.blogspot.com/2008/06/activemq-510-tutorial.html
- JAXB library:
    - You will use JAXB to bind XML messages to Java Objects.
    - Why? This library will allow you to generate XML from Java class and initialize a Java class from XML. Hence, you don't have to write code that parses XML and initialize a Java class or code to serialize a Java class to XML.
    - JAXB library is available as part of the JDK 1.6 (already installed on CDF)
    - JAXB tutorial: http://java.sun.com/webservices/docs/1.4/tutorial/doc/JAXBWorks2.html
- DBCP library:
    - You will use the DBCP library to cache and reuse database connections.
    - Why? To connect to the database, you will need to create a JDBC connection. This is an expensive operation that allocates significant resources on the Java side and also on the database side. One of the common mistakes is to create a connection for every request and close it after the request is executed. Instead, use a connection pool to create a set of connections and keep using them throughout the lifetime of the program. The max number of connections should not typically exceed 10-15 connections. If all the connections are already being used for other requests, your code should block till a connection is available. A database connection pool, e.g. DBCP, will give you this functionality.
    - On CDF, DBCP is installed under /u/csc309h/lib/dbcp-1.2.2
    - To use DBCP in your Java program, you will need this jar file: /u/csc309h/lib/dbcp-1.2.2/commons-dbcp-1.2.2.jar
    - Website: http://commons.apache.org/dbcp/
    - API documentation: http://commons.apache.org/dbcp/apidocs/index.html
    - Short tutorial: http://www.freshblurbs.com/jakarta-commons-dbcp-tutorial

Startup Files

- In the assignments folder on blackboard, you will find a2-startupfiles.rar This file include the following directories
  - <database> contains sql files for creating tables and defining stored procedures
  - <activemq> contains startup scripts for ActiveMQ
  - <xml> contains sample xml messages which you will need to work with

Working on CDF

- A list of students and assigned ports are provided in the assignments section (available from course website → Assignments).
- You should only use the assigned activeMQ ports to you to avoid interrupting other students. 4 ports have been allocated to each student (*you only need one to get the ActiveMQ running and another one if you are going to run the console*).
- You should use the assigned database to you!

Design Issues and Tips

- Each user session is identified by a unique number which is passed with every XML message in the <session-id> tag. This number is generated when the user login. Session times out if the user is inactive (i.e. does not send any request).
- Returning an XML response with multiple records is done following a consistent format with the tag <record id=some-counter> </record id> (check the XML document for examples).
- There is approximately 50 operations supported by the database stored procedure and a corresponding XML request and response for each. How do you implement all of these in the shortest time ?! Even a better question: how can we design a framework that supports adding/modifying both stored procedures and XML content without having to change the Java code every time such a change happen ? One possible approach is to document the mapping between stored procedures and xml tags in Java properties file (.prop) and your program uses this mapping file to learn which stored procedure to call, and which parameter should be passed which XML tag content, and also how to make the XML response. For example:

```
service-name = stored-procedure-name
stored-procedure-name.inputparam.1 = request-xml-tag-name
stored-procedure-name.inputparam.2 = request-xml-tag-name
stored-procedure-name.inputparam.3 = request-xml-tag-name
stored-procedure-name.inputparam.4 = request-xml-tag-name
stored-procedure-name.inputparam.5 = request-xml-tag-name
stored-procedure-name.outputparam.1 = response-xml-tag-name
stored-procedure-name.outputparam.2 = response-xml-tag-name
stored-procedure-name.outputparam.3 = response-xml-tag-name
```

```
stored-procedure-name.outputparam.4 = response-xml-tag-name
```

Using the login request to demonstrate how you use the above:

```
login= LoginUser
LoginUser.inputparam.1= email-id
LoginUser.inputparam.2= password
LoginUser.inputparam.3= user-ip
```

Obviously, the above needs refinement. You might need to add few other things.

Deliverable Directory Structure

<a2-team-name>

README.TXT	(file containing your team members, name, ids, cdf login. In addition to any missing features you didn't have time to implement)
runDBApp	(shell script to run the DBApp java application. Should include in the classpath the jars you are using, e.g. /u/csc309h/lib/activemq-core-5.2.0.jar)
runMQDriver	(shell script to run the MQDriver java application)
<db>	(directory containing all database related scripts: ddl, stored procedures, sample inserts, etc…)
<xml-requests-log>	(directory containing requests in xml)
<xml-responses-log>	(directory containing responses in xml)
<config>	(directory containing any configuration file for your java programs or the programs/libraries you use)
<lib>	(directory containing any additional library you are using)
<bin>	(java class files, you could have your own packages under this directory)
<src>	(java source files, you could have your own packages under this directory)
MQDriver.java	(under <src>)
DBApp.java	(under <src>)
Makefile	(under <src> Compiles your application and place executables under <bin>)

Submissions

- Your submission must work on CDF.
- Using one of the team members cdf account, submit a single zipped file using the following command: **submit -N assign2 csc309h a2-teamname.zip**
- The zip file should contain all your work including the parent directory (named after your team with prefix a2-).
- Do a **man submit** if you want more information about how to the submit command works.