95-702 Distributed Systems

Project 2

Assigned: September 21, 2012

Due: Friday, October 5, 11:59:59 PM

Project Topics: Relational Databases, Local Transactions, Web Services

In this project, we will build some small systems that illustrate various ways to interact with a remote database over standard web protocols.

First, the assignment contains detailed instructions on building a relational database using Netbeans and Java DB. Databases are an important component of many distributed systems and we spend some time on the handling of local transactions. We will visit distributed transactions later in the course. You are also provided with code that acts as a wrapper around a single row of the database. The code makes use of JDBC to interact with the database. You are required to run the code and see the impact on these data. In addition, you are shown how to build a SOAP based web service and a web service client that interact with the database.

Second, the student is provided with a new database schema and is required to create a small database with several tables and several constraints on database fields.

Third, the student is asked to write a SOAP based web service that may be used by two web service clients to make updates to the data. One client is a web application running on Glassfish and the other client is a standalone console based application. If a constraint violation occurs, the web service is written to rollback the local transaction. This is done with the rollback and commit features of JDBC connections.

Fourth, the student is asked to write a SOAP based web service that may be used by two web service clients to make updates to the data. One client is a web application running on Glassfish and the other client is a standalone console based application. This web service makes use of the Java Persistence API. The interface to the CRUD operations is designed by the student

This project will hopefully lead to careful considerations concerning WSDL, the concept of interoperability and local JDBC transaction handling with rollback and commit. It is broken into six tasks.

Task 0. Getting Started with JavaDB in Netbeans

0.0 Build a simple Derby Database.

0. Open Netbeans 7.X
1. Select Services Tab.
2. Expand Databases.
3. Right click Java DB.
4. Choose Create Database.
5. Enter Data Base Name (Account), Username, Password. Remember these.
6. Under Databases, Right click jdbc:derby://localhost...
7. Choose Connect. Notice how the icon appears to be connected.
8. Double click the icon associated with jdbc:derby://localhost...
9. Right click App and set as default schema.
10. Expand App and right click Table and Choose Create Table.
11. Name the table AccountTable.
12. Enter AccountID (SMALLINT, primary key, unique).
13. Enter CustomerName (VARCHAR size 40).
14. Enter Amount
    (DOUBLE, Checked constraint 0 <= Amount and Amount <= 15000).
15. Right Click AccountTable and select view data.
16. Click the Insert Record icon.
17. Enter tuples (100, Mary, 14000), (200, Sue, 1490), (20,Sam,3000),
    (30,Ben,500), (15,Joe,5000), (16,Mike,500).

0.1 Write a Java class using JDBC that wraps a row in the database.

0. See the code section below and take a little time to read over Account.java.
1. Select File/New Project/Java/Java Application.
2. The project name is P2T0DatabaseDemoProject.
3. Right click project and select New/Java Class.
4. Enter package name p2t0databasedemoproject.
5. Name the class Account.
6. Copy and paste Account.java from the course schedule.
   (Also shown in the code section at the bottom of this document.)
7. Right click project select New Java/Java Class.
8. The name is RecordNotFoundException.
9. Copy and paste RecordNotFoundException.java from the course schedule.
   (Also shown in the code section at the bottom of this document.)
10. Right click P2T0DatabaseDemoProject node.
11. Choose Set Configuration.
12. Choose Customize.
13. Choose Libraries.
14. Choose Add Jar/Folder.
15. Find and Double click on derbyclient.jar (perhaps located in C:\Program Files\glassfish-3.0.1\javadb\lib\derbyclient.jar).
16. Select OK.
17. Study Account.java. You need to change the user name and password.
18. We are not using the main provided by Netbeans and it may be deleted.
19. Select and then run Account.java.
20. What happens when you run Account.java a second time? Why? Look over the data in the database.

0.2 Write a Java client that interacts with Account objects.

0. See the code section below and take a little time to read over DBClient.java. DBClient.java is also available on the schedule for the class.
1. Right click the P2T0DatabaseDemoProject project node.
2. Select New/Java class/.
3. Name the class DBClient.java.
4. Name the package p2t0databasedemoproject.
5. Enter DBClient.java as shown below.
6. You need to change the user name and password.
7. Right click DBClient.java and select Run
8. What did the code do? Can you run the program a second time? Why?
9. Note that there are three new individuals being created in the database and their initial funds total to $24,100. What happens to the data in the database after an attempt to transfer $100.00 rather than $1.00? What happens after an attempt to transfer $2000.00?
10. Change the parameter passed to setAutoCommit - con.setAutoCommit(true) - and try to move $2000.00 rather than $1.00. What happens now? Look over the data in the database.

0.3 Write a web service that interacts with the database.

Write a web service that takes an ID of a database record and returns the name associated with that record. If there is no name with that ID then return the string "Record Not Found". The web service will operate on the Account table.

Note that your web service method needs to use an internal try/catch block. Do not attempt to declare your method as a thrower of exceptions.

0. Create a new Java Web project named P2T0DBWebServiceProject.
1. Right click the project and add a web service named DBWebService.
2. Provide one method in the service.
   public String getNameGivenID(@WebParam(name = "id") int id)
3. Fill in the body by using code from the Account class to interact with the database. (You may want to copy your existing package into this new project.)
4. Start Glassfish.
5. Right click the project to deploy the web service.
6. Expand the Web Services tab. Right click web service name. Test the new database web service. The browser will execute as a test platform. Copy the URL of the WSDL to be used in 0.4.

0.4 Write a console based web service client.

Write a simple Java program that interacts with a user and the web service. The interaction will be of your own design but need not be complex. Your client will contain documentation describing how it interacts with the user and what it does.

0. File/New Project/Java/Java Application/P2T0DBWebServiceClientProject/Finish
1. Right click this project. Select New Web Service Client. Paste the WSDL URL from 0.3.
2. Note the Web Service Reference node in project tree.
3. Expand Web Service Reference down to desired method.

4. Drag and drop into appropriate source location.
5. Be sure to clean and build your project. This ensures that the generated files will be seen at runtime.

Task 0 Summary

Submit a database project named “P2T0DatabaseDemoProject”.
Submit a web service project named “P2T0WebServiceProject”.
Submit a web service client named “P2T0WebServiceClientProject”.

Task 1. Database development using Netbeans and JavaDB

Build a small database using JavaDB and Netbeans. The database will be called Trip and will consist of three relations with the following schemas:

HOTEL
ID : integer unique key
name : string
location: string
URL : string
rooms_avail: integer checked constraint rooms_avail <= 100 and rooms_aval >= 0
CAR
ID : integer unique key
name : string
location: string
URL : string
cars_avail: integer checked constraint cars_avail <= 10 and cars_avail >= 0

PLANE

ID : integer unique key

name : string

location: string

URL : string

seats_avail: integer checked constraint seats_avail <= 40 and seats_avail >= 0

Populate the database with the following data:

CAR

1 Pittsburgh Rental Pittsburgh PA www.pghrental.com cars_avail = 10

2 LA Rental Los Angeles CA www.larental.com cars_avail = 10

3 NY Rental New York NY www.nyrental.com cars_avail = 2

4 SF Cars San Francisco www.sfcars.com cars_avail=5

HOTEL

1 Hilton Pittsburgh PA www.pghhilton.com rooms_avail = 100

2 Hilton New York NY www.nyhilton.com rooms_avail = 20

PLANE

1 QANTAS Sydney Australia www.qantas.com seats_avail = 40

2 American Pittsburgh PA www.aal.com seats_avail = 32

Task 1 Summary

Take a screen shot of each of the three tables as they appear in Netbeans. So that the grader may verify that you built the database, submit the screenshot(s) to Blackboard. Name the zip file P2T1ScreenShots.zip.

Task 2. A web service interacting with a database using Netbeans

Write a SOAP based web service in Java called BookTripWebService. The web service will run on Glassfish and will make available the following method:

```
public  boolean bookTrip(int hotelID, int numRooms, int carID, int numCars,
        int planeID, int numSeats) {
    :
    :
}
```

Note that the bookTrip method does not throw an exception. It uses try catch blocks instead. A value of false is returned to the client if an exception is generated within bookTrip.

If the booking is a legal one (that is, it does not violate any constraints set in the database) then the database is updated and the method returns true. The update consists of a reduction in the number of seats available, cars available, and rooms available. Otherwise, if the booking violates a database constraint, no change is made to the database and the method returns false.

Note that the bookTrip method does not check the database constraints. This is left up to the database. That is, your code will not check the data for a constraint violation. The database will do that and throw an exception when a violation occurs.

It makes sense that the database constraints are tested once and are tested at the database.

The bookTrip method will be contained in a file called BookTripWebService.java. This class will make good use of three additional classes: Car.java, Hotel.java, and Plane.java. Each of these will be modeled after Account.java that is shown below. Each of the Java classes that you write will have getters and setters as well as the methods

create, read, update, and two variations of delete. These are called “CRUD” operations. Feel free to copy the code from Account.java. But in the end, your Java code will have meaningful names. Do not use the names found in Account.java.

This project uses JDBC and you will need to use derbyclient.jar. See above to see how to include this jar file. With this jar added to the project you will be able to use the JDBC statements:

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
Connection con  = DriverManager.getConnection
        ("jdbc:derby://localhost:1527/Trip","mm6","sesame");
```

In order to implement local transactions, your web service method will make good use of JDBC connections as well as instructions to rollback and commit the transactions.

Task 2 Summary

When you test your web service using the testing capabilities provided by Netbeans, you will be prompted by your browser for the parameters that need to be passed to the bookTrip method. Take a screen shot of this browser view. Also, after the browser visits the web service, input and output SOAP documents are displayed. Take a screen shot of this browser view. The zip file containing screenshots will be named P2T2ScreenShots.zip. In addition, submit to blackboard your web service project named P2T2WebServiceProject.

Task 3. A web application that interacts with the web service

Write a web service client that allows a user to make calls on the bookTrip method. This web service client will be a JSP or servlet based web application. The browser user will be prompted for values and those values will be used to update the database.

If you wish, you may choose to use the Model View Design pattern. A simpler design is also fine.

The browser user should be told if a constraint violation occurs.

Task 3 Summary

Take a screen shot showing the data being entered into an HTML form. Take another one after the web application responds. The zip file containing the screen shots will be named P2T3ScreenShots.zip. In addition, submit to blackboard your web service project named P2T3WebServiceWebApplicationClientProject.

Task 4. A console application that interacts with the web service

Write a web service client that allows a user to make calls on the bookTrip method. This web service client will be a command line application. The user will be prompted for values and those values will be used to update the database. Be sure to inform your user of any constraint violations. Again, the exact interaction is in your hands.

Task 4 Summary

Take a screen shot of the console when it's running. Submit a documented client to blackboard. The project will be named P2T4ConsoleWebServiceClientProject and the screen shots will be named P2T4ScreenShots.zip.

Task 5. Java Persistence API

In the previous tasks, you used JDBC along with wrapper classes (Account, Plane, etc.). This is an example of the principle that we should separate concerns. You were asked to write these wrapper classes yourself. With JPA, these classes are generated for you. This is sometimes called object relational mapping.

Detailed instructions are provided here for building a relational database using Netbeans and Java DB. Instructions are also provided for building a web service that uses JPA to query the database.

5.0 Build another simple Derby Database.

0. Open Netbeans 7.X
1. Select Services Tab.
2. Expand Databases.
3. Right click Java DB.
4. Choose Create Database.
5. Provide a database name (StudentDB), user name (newton) and password (newton). Remember these.
6. A connection should now be available.
7. Right click the connection: jdbc:derby://localhost.../StudentDB
8. Choose Connect.
9. Expand the connection jdbc:derby://localhost.../StudentDB
10. Right click App and set as default schema.
11. Expand App and right click Table and Choose Create Table.
12. Name the table StudentTable. And then click Add column.
13. Enter Student_ID (SMALLINT, primary key, unique).
14. Enter Student_Name (VARCHAR size 40).
15. Enter Scholarship_Amount (DOUBLE, Checked constraint 0 <= ScholarShip_Amount and ScholarShipAmount <= 45000).
16. Right Click StudentTable and select view data.
17. Click the Insert Record icon.
18. Enter tuples (100, Mary, 14000), (200, Sue, 1490), (20,Sam,3000), (30,Ben,500), (15,Joe,5000), (16,Mike,500).

5.1 Use JPA to generate code that talks to the database.

0) From Projects, build a new Java Web Application called ReadStudentDatabaseProject.
1) Right click the project node and select New Entity Classes from Database.
2) Expand the Data Source and select New Data Source.
3) Enter the JNDI name as jdbc/StudentDB and select the connection that we built earlier. You should now see StudentTable under available tables. Select it and add it to the Selected Tables area. Select Next.
4) Provide a package name, e.g., edu.cmu.andrew.mm6. Choose defaults and finish.
5) Explore the Java code that has been generated for you. There should be a new class within the package and the new class will be named Studenttable.
6) We need to make one change to this generated source code. Change a line near the top to read: @Table(name = "APP.STUDENTTABLE") instead of reading: @Table(name = "STUDENTTABLE").

5.2 Write a simple web service that uses JPA.

0) From Projects, right click the ReadStudentDatabaseProject. Choose New Web Service. Call the project StudentListService and use the same package name as before. Select Finish. The StudentListService.java file should be in the same package as the Studenttable.java file.
1) Edit the web service source and remove the hello operation. Select Design to design the interface of the service.
2) Create a new operation called findAllStudentNames. This method takes no arguments but returns a java.util.List.
3) Enter the source mode and enter the following as the service implementation: (Note: This code StudentListService.java is also found on the course schedule.)

```
/** Example database web service using JPA. */
package edu.cmu.andrew.mm6;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceUnit;
import javax.persistence.Query;

@WebService(serviceName = "StudentListService")
public class StudentListService {
    @PersistenceUnit EntityManagerFactory emf;
    /**
     * findAllStudentNames is a web service operation.
     * The service takes no arguments but queries the
     * database and returns a list of names.
     */
    @WebMethod(operationName = "findAllStudentNames")
    public List findAllStudentNames() {
      // build an empty list
      List list = new ArrayList();
      // build a query based on find all records in the table
      Query query =
              emf.createEntityManager().createNamedQuery("Studenttable.findAll");

      // get the result of the query, a list of records
```

```
        List students = query.getResultList();
        // create an iterator over the list of records
        Iterator iter = students.listIterator();
        while(iter.hasNext()) {
            // take a Studenttable record
            Studenttable student = (Studenttable) iter.next();
            // add the students name to the list
            list.add(student.getStudentName());
        }
        // return the list of names
        return list;
    }
}
```

4) Clean and Build the project node of the service.

5) Deploy the project node of the service.

6) Test by right clicking StudentListService under the Web Services folder.

5.3 Build a CRUD web service using JPA and interact with two SOAP clients.

The assignment is to extend the web service so that it has operations for the standard CRUD (create, read, update and delete) operations. In addition, the student is required to build two clients - a web application and a console application. Both of these will make use of the SOAP web service.

The Create operation will allow the user to build a new record in the database.

The Read operation will return the student's name and scholarship amount given the student id.

The Update operation will allow the caller to change the name or scholarship amount given the id.

The delete operation will allow the caller to delete a record given the id.

I had success writing a create() operation by doing the following:

1. In the configuration file, persistence.xml, set “Use Java Transaction API’s” to off. Do this in the design mode when viewing persistence.xml.
2. Use code like this inside your create method:

```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Studenttable student = new Studenttable();
student.setScholarshipAmount(new Double(amount));
student.setStudentName(name);
student.setStudentId(new Short(id+""));
em.persist(student);
em.getTransaction().commit();
```

On the web, do some research on JPA. There are plenty of examples that will help you get the remaining operations running.

Take screen shots of the browser of the web application when it's running. Submit a documented web application/web service client project to blackboard. The project will be named P2T5WebAppWebServiceClientProject and the screen shots will be named P2T5WebAppScreenShots.zip.

Take screen shots of the console application when it is running. Submit a documented console based web service client project to blackboard. The project will be named P2T5ConsoleAppWebServiceClientProject and the screen shots will be named P2T5ConsoleScreenShots.zip.

The design of the user interactions (from the console and web app) is in your hands. In both cases, how it works should be obvious to the evaluator. A simple, clean and attractive user interface will gain the most points.

Project 2 Task Summary

Task Number	Description
T0	JDBC Tutorial including local transaction
T1	New DB Construction using JavaDB and Netbeans
T2	SOAP based Web Service to DB
T3	HTML Web Application to Web Service to DB
T4	SOAP based console application to Web Service to DB
T5	Console and web app clients to SOAP based web service built using JPA

Code: Account.java, RecordNotFoundException.java, and DBClient.java

```
/* Account.java */
package p2t0databasedemoproject;

// Professional Java Server Programming J2EE Edition (modified)
// A simple Account object - wraps a row in the Account table.

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.*;
import java.util.*;
import java.io.*;

public class Account {

private String accountID;
private String customerName;
private double amount;

public String toString() {
   return "ID " + accountID + "\t" + "Name " + customerName + "\t" + "Amount "
+ amount ;
}
```

```
public String getAccountID() {
      return accountID;
}

public String getCustomerName() {
      return customerName;
}

public void setCustomerName(String c) {

      customerName = c;
}

public double getAmount() {

      return amount;
}

public void setAmount( double amt) {
      amount = amt;
}

public void create(String actID, String customerName, double amt, Connection con)
        throws SQLException, Exception {

    accountID = actID;
    this.customerName = customerName;
    this.amount = amt;
```

```
        PreparedStatement statement = null;

        try {


            statement = con.prepareStatement("Insert into app.AccountTable
(accountID," +
                                             "CustomerName, Amount)"  +
                                             "Values ( ?,?,?)");
            statement.setString(1,accountID);
            statement.setString(2,customerName);
            statement.setDouble(3,amt);



            statement.executeUpdate();
        }

        catch(SQLException e) {
            System.out.println("Caught exception in create" + e);
            throw new Exception(e);
        }

        finally {
            if(statement != null) {
                statement.close();


            }
        }
    }
```

```
    public void read(String accountID, Connection con) throws SQLException,
RecordNotFoundException {

        PreparedStatement statement = null;

        try {

            statement = con.prepareStatement("Select customerName, amount FROM
app.AccountTable"
                                    +" where accountID = ?");

            statement.setString(1,accountID);

            ResultSet result = statement.executeQuery();

            if(result.next()) {
                this.accountID = accountID;
                this.customerName = result.getString(1);
                this.amount = result.getDouble(2);

            }
            else {
                System.out.println("Could not read a record");
                throw new RecordNotFoundException();
            }
        }
        finally {
            if(statement != null) {
                statement.close();
```

```
        }
      }
    }

    public void update(Connection con) throws SQLException {

        PreparedStatement statement = null;

        try {
            statement = con.prepareStatement("Update app.accountTable set
customername = ?," +
                                    "amount = ? where accountID = ?");

            statement.setString(1, customerName);
            statement.setDouble(2, amount);
            statement.setString(3,accountID);

            statement.executeUpdate();
        }
        finally {
            if(statement != null) {
               statement.close();
            }
        }
    }


    public void delete(Connection con) throws SQLException {
```

```
    PreparedStatement statement = null;

    try {

        statement = con.prepareStatement("Delete from app.AccountTable Where
AccountID = ?");
        statement.setString(1,accountID);
        int h = statement.executeUpdate();
        System.out.println("Tried to delete " + accountID + " Changed " + h + "
records");
    }
    finally {
        if (statement != null) {
            statement.close();
        }
    }

  }

  public void delete(String accountID, Connection con) throws SQLException {

    PreparedStatement statement = null;

    try {

        statement = con.prepareStatement("Delete from app.AccountTable Where
AccountID = ?");
        statement.setString(1,accountID);
        statement.executeUpdate();
    }
```

```
        finally {
            if (statement != null) {
                statement.close();
            }
        }

    }

    public static void main(String args[]) throws SQLException,
RecordNotFoundException,
    ClassNotFoundException, Exception  {



                Class.forName("org.apache.derby.jdbc.ClientDriver");



                Connection con =

DriverManager.getConnection("jdbc:derby://localhost:1527/AccountRDBMS","mm6","
sesame");

                System.out.println("Built connection");

                // Test code. After running once the database has data.
                // That's why a second execution throws an exception.

                Account personA = new Account();

                System.out.println("Built an account");
```

```
personA.create("1","Mike McCarthy",100.0,con);
System.out.println("Create complete");

Account personB = new Account();
personB.create("2","Sue Smith",45.00,con);

System.out.println("Two Accounts constructed");

ResultSetMetaData rsm = null;

String answer = "";

Statement s = con.createStatement();

ResultSet rs = s.executeQuery("select  * from app.AccountTable");

rsm = rs.getMetaData();

try {

   while(rs.next()) {

      for(int col = 1; col <= rsm.getColumnCount(); col++)
          answer += rs.getString(col);
   }
   con.close();
}
```

```
            catch (SQLException sqle) {
               System.err.println("Exception caught in main:" + sqle);
            }

            System.out.println(answer);
            con.close();

   }
}
```

==========

```
package p2t0databasedemoproject;


/**
 *
 * @author mm6
 */
public class RecordNotFoundException extends Exception{

       public RecordNotFoundException() {

   }
       public String toString() {
          return "Could not find database record";
       }
}
```

```
============================================================

/*
 * 95-702 Distributed Systems DBClient.java
 */
package p2t0databasedemoproject;

import java.sql.Connection;
import java.sql.DriverManager;

public class DBClient {


    private static final String ACCOUNT1 = "123";
    private static final String NAME1= "Cristina Couglin";
    private static final double AMOUNT1 = 10000.0;

    private static final String ACCOUNT2 = "124";
    private static final String NAME2= "Mary Klopot";
    private static final double AMOUNT2 = 14000.0;

    private static final String ACCOUNT3 = "125";
    private static final String NAME3= "Mike McCarthy";
    private static final double AMOUNT3 = 100;

    private static final double TRANSFER_AMOUNT = 1.00;

    public static void main(String args[]) throws Exception {
```

```
    Class.forName("org.apache.derby.jdbc.ClientDriver");

    Connection con =

DriverManager.getConnection("jdbc:derby://localhost:1527/Account","mm6","sesam
e");



    // set up three test accounts
    createAccounts(con);

    // move some money around from a to b to c to a
    // from Cristina to Mary
    transfer(ACCOUNT1, ACCOUNT2, TRANSFER_AMOUNT,con);
    transfer(ACCOUNT2,ACCOUNT3,TRANSFER_AMOUNT,con);
    transfer(ACCOUNT3,ACCOUNT1,TRANSFER_AMOUNT,con);



    }

    private static void createAccounts(Connection con) throws Exception {

        try {

            // Create three new accounts after removing the old
            // versions if any.

            Account account1 = new Account();
            account1.delete(ACCOUNT1, con);
            account1.create(ACCOUNT1,NAME1,AMOUNT1,con);
```

```
            System.out.println(account1);

            Account account2 = new Account();
            account2.delete(ACCOUNT2, con);
            account2.create(ACCOUNT2,NAME2,AMOUNT2,con);
            System.out.println(account2);

            Account account3 = new Account();
            account3.delete(ACCOUNT3, con);
            account3.create(ACCOUNT3,NAME3,AMOUNT3,con);
            System.out.println(account3);

            System.out.println("Acoounts created");
        }
        catch(Exception e) {
            System.out.println("Exception thrown");
            e.printStackTrace();
            throw new Exception(e);
        }

    }


    private static void transfer(String accountIDFrom, String accountIDTo, double
amount, Connection con) {

            try {

              // transfer amount from a to b
```

```
    con.setAutoCommit(false);

    Account accountFrom = new Account();
    accountFrom.read(accountIDFrom,con);

    Account accountTo = new Account();
    accountTo.read(accountIDTo,con);

    accountFrom.setAmount(accountFrom.getAmount() - amount);

    accountTo.setAmount(accountTo.getAmount() + amount);

    accountFrom.update(con);
    accountTo.update(con);

    System.out.println("Funds Transfererred");
    System.out.println("From account " + accountFrom);
    System.out.println("To account " + accountTo);

    con.commit();
}

catch(Exception e) {
    try {
        System.out.println("Transaction aborted - Rolling back changes.");
         con.rollback();
         System.exit(1);
        }
```

```
                catch(Exception re) {
                    System.out.println("Problem doing rollback. Exception " + re);
                }
                e.printStackTrace();
            }

        }
    }
```