This project has four major parts broken into six tasks:

First, the student is provided with detailed instructions on building a relational database using Netbeans and Java DB. The student is 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. The student is required to run the code and see the impact on these data. In addition, the student is 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 JSP page 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 an Android application that interacts with a database on a server.

This project will hopefully lead to careful considerations concerning WSDL, interoperability, locking, transaction handling, entity beans and distributed transactions. It is also important to think about the protocols being employed in the
distributed parts of this project. These topics will be addressed in the reflection questions related to this project.
Task 0. Getting Started with JavaDB in Netbeans

0.0 Build a simple Derby Database.

0. Open Netbeans 6.X
1. Select Services Tab.
2. Expand Databases.
3. Right click Java DB.
4. Choose Create Database.
5. Enter Database Name (Account), Username, Password. Remember these.
6. Under Databases, Right click jdbc:derby://localhost...
7. Choose Connect.
8. Expand 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. 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.

2. The project name is P5T0DatabaseDemoProject.
3. Right click project and select New/Java Class.
4. Enter package name databasedemoproject.
5. Name the class Account.
6. Enter Account.java as shown below.
7. Right click project select New Java/Java Class.
8. The name is RecordNotFoundException and the package is databasedemoproject.
9. Enter RecordNotFoundException as shown below.
10. Right click P5T0DatabaseDemoProject node.
11. Choose Set Configuration.
12. Choose Customize.
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?
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.
1. Right click the P5T0DatabaseDemoProject project node.
2. Select New/Java class/.
3. Name the class DBClient.java.
4. Name the package databasedemoproject.
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?
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 P5T0DBWebServiceProject.
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 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. 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/P5T0DBWebServiceClientProject/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 “P5T0DatabaseDemoProject”.
Submit a web service project named “P5T0WebServiceProject”.
Submit a web service client named “P5T0WebServiceClientProject”.

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_avail >= 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 P5T1ScreenShots.zip.
Task 2. A web service interacting with a database using Netbeans

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

```java
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:

```java
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 P5T2ScreenShots.zip. In addition, submit to blackboard your web service project named P5T2WebServiceProject.

**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 based web application. You will need to add an HTML front end to this web application. The browser user will be prompted for values and those values will be used to update the database.
From within a JSP page, one can very simply access the parameters passed from an HTML form by using the built in JSP request object.

If you prefer, you may use a servlet instead of a JSP page. JSP pages are automatically converted to servlets.

The web application’s appearance and design is up to you.

**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 P5T3Screen Shots.zip. In addition, submit to blackboard your web service project named PST3WebServiceWebApplicationClientProject.

**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. 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 P5T4ConsoleWebServiceClientProject and the screen shots will be named P5T4Screen Shots.zip.

**Task 5. An Android device books a trip**

Provide an Android application that displays a button labeled “BookTrip” that allows a user to interact with the database that you built in Task 1. Each time that the
BookTrip button is pressed a trip will be booked – reserving a room, a seat on a plane, and a car. We will always be using American Air Lines, the Pittsburgh Hilton and the Pittsburgh Car Rental. The exact design of this application is up to you. We are not using Android’s browser in this task.

**Task 5 Summary**

This will be an Eclipse project named P5T5AndroidClient. Include screen shots demonstrating the interaction. These will be in a file named P5T5ScreenShots.zip.

**Overall Summary**

<table>
<thead>
<tr>
<th>Task Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>DB Tutorial</td>
</tr>
<tr>
<td>T1</td>
<td>New DB Construction</td>
</tr>
<tr>
<td>T2</td>
<td>Web Service to DB</td>
</tr>
<tr>
<td>T3</td>
<td>Web App to Web Service</td>
</tr>
<tr>
<td>T4</td>
<td>Console app to Web Service</td>
</tr>
<tr>
<td>T5</td>
<td>Android app to Web app</td>
</tr>
<tr>
<td></td>
<td>Reflection Questions</td>
</tr>
</tbody>
</table>
/* Account.java */
package databasedemoproject;

// 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) {

        }
    }
}
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 {

    statement.close();

}

}
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 databasedemoproject;

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/AccountRDBMS","mm6","sesame");

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

// move some money around from a to b to c to a
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("Accounts 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 Transferred");
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();
}
catch(Exception re) {
 System.out.println("Problem doing rollback. Exception " + re);
}
e.printStackTrace();