School of Engineering and Computer Science

SWEN 304 Database System Engineering

Project 2

Due Date and Time: Friday 11 October, 23:59 pm

This project will build your understanding of database transaction processing and locking using JDBC.

The project is worth 10% of your final grade, and is marked out of 100.

Submission

You should submit the following two files:

- `LibraryUI.java` and
- `LibraryModel.java`

electronically using the link to the submission system on the course home page. You do not need to hand in paper copies of your program.

Background

To do the project you will use

- PostgreSQL Database Management System to build a database,
- Java programming language to write a transaction program, and
- Java Database Connectivity (JDBC) library to exchange data between your database and your transaction program.

To familiarise yourself with JDBC use the lecture and review notes on JDBC published on the course web page as an introductory resource for learning how to use JDBC in your programs. You can find out more about JDBC from PostgreSQL and Java Tutorial (http://ecs.victoria.ac.nz/technical/java/tutorial/index.html). You will find both manuals in the Technical web pages that are referenced from the SWEN 304 website. Do not print out all of the tutorials.

Short Description of the Problem

In this project, you will implement a simple library system. The system keeps track of books, authors, library customers, and their relationships.

Using your system, a library clerk should be able to retrieve information about books, authors, and customers. Also, the clerk should be able to update the appropriate records when a customer borrows or returns a book. Your program should send polite and understandable messages in a plain language to users.

Your program should be written in such a way that allow multiple instances of same program to access and update a common database without introducing any inconsistencies. Therefore, you should make a real transaction program.

Setting Up the PostgreSQL Database

Your transaction program will need to use Java libraries, PostgreSQL, and the PostgreSQL JDBC driver. To use all of these, you should insert the following command in your `.cshrc` file:

`need comp302tools` or **`need postgresql`**

It should be inserted between the lines

```
need SYSfirst
...
need SYSlast
```

The `comp302tools` package sets up the `CLASSPATH` to include "`postgresql94.jar`" (which contains a JDBC driver manager), and also does a "`need java2 `" and a "`need postgresql`". So if you already have any of these other "need" lines you can delete them (although it won't harm to keep them).

Create your database, named **⟨userid⟩_jdbc** by typing

`> createdb` ⟨`userid`⟩`_jdbc`

Note, this is the only database that should be set up to allow access from a Java program. Please use your ECS username as your ⟨`userid`⟩.

To set up the password for your new database `<userid>_jdbc`, open an existing database (say with the name `<userid>`) by typing:

`> psql` ⟨`userid`⟩

and then proceed with

⟨**`userid`**⟩ **`=>ALTER USER`** ⟨`ECS username`⟩ **`WITH PASSWORD`**
`'`⟨`newpassword`⟩**`';`**

This will set your password for the database `<userid>_jdbc`. Note, this password should NOT be the same as your system password. It should protect your database from possible misuse, but need not to be as secure as your system password. Type

⟨`userid`⟩`=>\q`

to close your old database.

To load your database with schema and data, you should use the file

`Project2_19.data`

which contains `CREATE TABLE` and `INSERT` commands and data to populate your database. You will find the file on the Assignments&Projects course web page. Store it in your private directory. The command

> psql –d ⟨db_name⟩ **-f** ~/Project2_19.data

executes SQL `CREATE TABLE` and `INSERT` commands in the file `/Project2_19.data` and populates your database. Note, ⟨db_name⟩ will be **⟨userid⟩_jdbc.**

For this project, you are not asked to use your database in an interactive mode, but if you need to (to try out a query, for example), do it in the same way that you did in Project 1.

Eclipse

If you are using Eclipse you need to:

1) Select the project you are working on.
2) Select "Project" -> "Properties" from the Menu
3) Select "Java Build Path" from the window that appears
4) Select "Add External Jars"
5) Navigate to "/usr/pkg/lib/java/postgresql94.jar"

The Structure of the Database Transaction Application

The database transaction application has the following components:

- A library database (in the file `Project2_19.data`)
- A Java program named `LibraryUI.java` that contains the code for the GUI and the `main` method.
- A Java file named `LibraryModel.java` that should implement methods that provide functionality to the options offered by the GUI in `LibraryUI.java`.

A dump of the schema and data for the database, and the two java files are in the Assignment web page. You will need to complete the `LibraryModel.java` file, but you should not need to modify `LibraryUI.java` at all.

The Library Database

Table 1 contains descriptions of the library database tables, and Table 2 contains description of the referential integrity constraints.

In the book table, the attribute *NumOfCop* is the number of copies of a book that the library owns, and the attribute *NumLeft* is the number of copies of a book that are currently available in the library. The difference between *NumOfCop* and *NumLeft* indicates how many copies of a book are currently out on loan. A book can be simultaneously loaned up to *NumOfCop* customers.

Attribute	Data Type	Length	Null	Def	Condition
Table name: *Customer*, Primary Key: *CustomerId*					
CustomerID	Integer	4	N	0	≥ 0
L_Name	Char	15	N		
F_Name	Char	15	Y		
City	Char	15	Y		(Wellington, Upper Hutt, Lower Hutt)
Table name: *Cust_Book*, Primary Key: *CustomerId* + *ISBN*					
CustomerId	Integer	4	N	0	≥ 0
DueDate	Date		Y		
ISBN	Integer	4	N	0	≥ 0
Table name: *Book*, Primary Key: *ISBN*					
ISBN	Integer	4	N	0	≥ 0
Title	Char	60	N		
Edition_No	SmallInt	2	Y	1	> 0
NumOfCop	SmallInt	2	N	1	
NumLeft	SmallInt	2	N	1	
Table name: *Author*, Primary Key: *AuthorId*					
AuthorId	Integer	4	N	0	≥ 0
Name	Char	15	Y		
Surname	Char	15	N		
Table name: *Book_Author*, Primary Key: (*ISBN* + *AuthorId*)					
ISBN	Integer	4	N	0	≥ 0
AuthorId	Integer	4	N	0	≥ 0
AuthorSeqNo	Integer	2	Y	1	> 0

Table 1.

	Referencing Relation Schemas				
	Customer	*Cust_Book*	*Book*	*Author*	*Book_Author*
Customer		(*d*, *r*), (*m*, *r*)			
Cust_Book					
Book		(*d*, *r*), (*m*, *r*)			(*d*, *d*), (*m*, *c*)
Author					(*d*, *d*), (*m*, *c*)
Book_Author					

Table 2.

The referential integrity constraints are specified in Table 2. The relation schemas in the column headers are referencing the relation schemas in the row headers. If the corresponding table cell is empty, there is no referential integrity constraint defined between the two relation schemas. If the cell is not empty, then there is a referential integrity constraint defined between the two relation schemas, and the cell contains a pair (*operation, action*). *Operation* is either *d* (delete), or *m* (modify), and *action* can be *r* (restrict), *c* (cascade), *n* (set to null), or *d* (set default).

The LibraryUI.java and LibraryModel.java Files

You should copy `LibraryUI.java` and `LibraryModel.java` into your own directory, compile them both using

```
>javac LibraryUI.java
```

and run using

```
> java LibraryUI
```

You will see that the program runs, setting up a GUI, and asking for a username and database password. However, at present none of the buttons on the GUI do much. They only display some messages in the GUI result area. `LibraryUI.java`, which contains the code for the GUI, is complete and you do not need to change it. `LibraryModel.java` contains a skeleton consisting of stubs of all the methods that provide the functionality for the buttons of the GUI. You need to implement all these methods.

The GUI contains an area for displaying results of queries, and a menu (tabs) with the following options:

- Book
  - Book Lookup
  - Show Catalogue
  - Show Loaned Books
- Author
  - Show Author
  - Show All Authors
- Customer
  - Show Customer
  - Show All Customers
- Borrow Book
- Return Book
- File
  - Exit

When the GUI is constructed, it will create an instance of the `LibraryModel class`, passing it a username and a database password. Each option on the GUI will then call a method of the `LibraryModel` object, and display the string returned by the method in the result area.

Note, you can change the given GUI, but you should not expect to receive extra marks for it.

Demo Program

There is a demo version of the program in the `~comp302/` directory that you can run on your own database. The demo program was initially made by Pavle, but was corrected and decorated by Jerome Dolman. To run the demo version, you need to:

- Create your own `<userid>_jdbc` database,
- Register your new password with PostgreSQL,
- Populate your own `<userid>_jdbc` database using `Project2_19.data` file
- Type:

  ```
  > demo comp302 project2
  ```

  from Unix shell prompt, and

- Enter your <user_name> and the new password in the Authentication Pane that will appear on the screen.

Note:

- This is just a **demo**, you may want to make a much better `LibraryModel.java` program.

What should you do

1. Load your database from the file `Project2_19.data`.
2. Complete the implementation of the `LibraryModel` class:
   a) Define the constructor to open a connection to the database
   b) Define each of the methods to perform an appropriate SQL query (or queries) and return the result of the query as a string to be displayed.

When implementing methods of the `LibraryModel` class, pay attention to commands for handling transactions correctly (for starting and ending a transaction, and locking), and to JDBC commands (for acquiring a driver object, establishing a connection to the database, controlling the transaction environment, and executing SQL statements using `Statement` and `PreparedStatement` objects). For the sake of performance, allow read only queries to read all database items, regardless whether they are locked or not. Most of your methods will need to handle exceptions raised by the JDBC commands. The most important goal of your transaction program is to leave the database in a consistent state after executing each of the requested database operations.

An important part of the database application you are going to implement is the method for the "BorrowBook" transaction. After receiving *ISBN*, *CustomerId*, and *DueDate* values from the GUI that method needs to perform the following actions (assuming that none of the actions fails):

1. Check whether the customer exists (and lock him/her as if the delete option were available).
2. Lock the book (if it exists, and if a copy is available).
3. Insert an appropriate tuple in the *Cust_Book* table.
4. Update the *Book* table, and
5. Commit the transaction (if actions were all successful, otherwise rollback)

This transaction is intended to work in a multi-user transaction-processing environment where there are multiple copies of your program running at the same time. To allow you (and the marker) to test whether your program works correctly, insert an interaction command between steps 3 and 4 above. This interaction command should put up a dialog box and ask the user to click `YES/OK` to continue. The effect of this is to stall the processing of the program while you run another copy of the program and invoke a transaction that should interfere with the transaction that is part way through. This way, you can simulate contention for the same database items, and check whether your program:

- Avoids lost update, dirty and unrepeatable read for database update transactions and
- Allows reading all database items for read only transactions.

Apply the steps, similar to steps 1 to 5 above also when implementing the Return Book function. Of course, this time you need to delete an appropriate tuple from the *Cust_Book* table.

How Will the Project be Marked?

The project contains two parts:

The first part will be worth 85 marks and consists of the following tasks:

- Implement the database [1 mark]
- Implement the Book Lookup Function [5 marks]
  (show book authors sorted according to AuthSeqNo)
- Implement the Show Catalogue function [5 marks]
- Implement the Show Loaned Books function [5 marks]
- Implement the Borrow Book function [35 marks]
- Implement the Exit function [4 marks]
- Implement the Show Author function [5 marks]
- Implement the Show All Authors function [5 marks]
- Implement the Show Customer function [5 marks]
- Implement the Show All Customers function [5 marks]
- Implement the Return Book function [10 marks]

The second part will be worth 15 marks and consists of the following tasks:

- Implement Delete customer function, [6 marks]
- Implement Delete author function, and [3 marks]
- Implement Delete book function. [6 marks]

The **marking** will be performed by markers after your electronic submission.

Also note we expect your code to be tidy and properly indented. Also, we expect to see user friendly and properly structured messages from your program. Please provide a regular exit from your program, particularly in the case of an error, e.g., database connection error. If you do not meet these expectations, there may be some penalties.

Using PostgreSQL on the workstations

We have a command line interface to PostgreSQL server, so you need to run it from a Unix prompt in a shell window. To enable the various applications required, first type either

> need comp302tools

You may wish to add "`need comp302tools`" command to your `.cshrc` file so that it is run automatically. Add this command after the command `need SYSfirst`, which has to be the first `need` command in your `.cshrc` file.

There are several commands you can type at the unix prompt:

> createdb ⟨`userid_jdbc`⟩

Creates an empty database. The database is stored in the same PostgreSQL default school cluster used by all the students in the class. **Note**, for this project, your database has to have the name `userid_jdbc`, where `userid` is your UNIX `userid`. To ensure security, you must define a password, as instructed in this handout. You only need to do this once (unless you get rid of your database to start again).

> **psql** [**-d** ⟨`db name`⟩ **]**

Starts an interactive SQL session with PostgreSQL to create, update, and query tables in the database. The db name is optional (unless you have multiple databases)

> dropdb ⟨`db name`⟩

Gets rid of a database. (In order to start again, you will need to create a database again)

> pg_dump -i ⟨`db name`⟩ > ⟨`file name`⟩

Dumps your database into a file in a form consisting of a set of SQL commands that would reconstruct the database if you loaded that file.

> **psql -d** <database_name> **-f** <file_name>

Copies the file <file_name> into your database <database_name>.

Inside an interactive SQL session, you can type SQL commands. You can type the command on multiple lines (note how the prompt changes on a continuation line). End commands with a ';'

There are also many single line PostgreSQL commands starting with '\' . No ';' is required. The most useful are

\? to list the commands,

\i ⟨file name⟩
loads the commands from a file (e.g., a file of your table definitions or the file of data we provide).

\dt to list your tables.

\d ⟨table name⟩ to describe a table.

\q to quit the interpreter

copy <table_name> **to** <file_name>
Copy your table_name data into the file file_name.

copy <table_name> **from** <file_name>
Copy data from the file file_name into your table table_name.

Note also that the PostgreSQL interpreter has some line editing facilities, including up and down arrow to repeat previous commands.

For longer commands, it is safer (and faster) to type your commands in an editor, then paste them into the interpreter!