## The Project

In this project, you will write an *application program* in Java using JDBC that will update the York River Bookstore (YRB) database. The database is the very same database from Project #2. You each are working with your own copy of the database.

### 1.The Task

The task that the store managers have to do is to add today's purchase records into the database. You have been asked to automate this task with an *application program*, let us call it *AddPurchase*. You are to make this in Java using JDBC; so, AddPurchase.java. The app should connect with EECS's PostgreSQL database server at **db** with the YRB database to record the purchases.

### 2.The Specification

The app will be called from the command line, accepting flags and parameters,

% java AddPurchase -c <cid> -b <club> -t <title> -y <year> [-w <when>] [-q <qnty> ] [-u <user> ]

* *cid*: the customer id who made the purchase
* *club*: which club that the puchase is made
* *title, year*: which book the customer purchased
* *whenp*(optional): when the purchase is made. if not provided, use the system current time.
* *qnty* (optional): the number of copies of the book in this purchase. The default is 1.
* *user*(optional): which *user* and *database* the app is connecting with and to, respectively. This should default to your user name (which is also your database's name)

**Important**: flags and parameters come in pairs but may in different order.

### 3.Error Messages:

The app should provide an error message back to the user for each of the following cases. (Your Java program should finish without failing in error itself in these cases!)

* The customer (*cid*), the *club*, or the book (*title* & *year*) does not exist: if it does not exist in the corresponding table, the app should state this and not make any changes to the database.
* The customer (*cid*) doesn't belong to that *club*: if the customer is not a member of the given club, the app should state this and not make any changes to the database.
* The *club* doesn't offer the book (*title* & *year*): if the club does not offer the book, the app should state this and not make any changes to the database.
* *whenp* is not today: if the new purchase is not made in today (the day performing your app), the app should state this and not make any changes to the database.
* *qnty* value is improper: if the *qnty* is not a positive integer, the app should state this and not make any changes to the database.

### 4.Result

Given no failure mode occurs, your app should add a tuple into the *yrb\_purchase* table with the specified parameters.

## The Driver

Your app needs a driver to set up the connection to our Postgres database server on db(.eecs.yorku.ca). The driver version we are using is *postgresql-42.2.14.jar*.

## The Authentication

On setting up the *database connection*, the program has to provide the *host*, *port*, *user*, *database*, and *password*. But putting one's *password* in program source is extremely bad practice. So, we will *not* allow it here. Additionally, you are writing your program to be general, and not to *hardcode* the *user* / *database* name into the program. User *wxfu*, say, ought to be able to take your program later and execute, say,

% java AddPurchase -c 2 -b AAA -y 1997 -t 'Richmond Underground' -u wxfu -q 2 -w '2020-03-15'

and have it work!

The best way to set it up would be via SSL and certificates, to provide a “drop-through” authentication. But that is not feasible for this project.

Instead, we shall use a .pg\_pass file in your home directory on PRISM. Refer to the guide, [*psql*: *PostgreSQL's shell client*](https://www.eecs.yorku.ca/~godfrey/guide/psql.md) (a guide to using psql with PRISM's **DB**), to set this up. Create a file named ".pgpass" in your home directory on PRISM with perms 600. In this file, you will have a format as follow:

HOST:PORT:DATABASE:USER:PASSWD

E.g.,

DB:5432:wxfu:wxfu:my\_fake\_password

A quick way to test if your ".pgpass" file works is just call

% psql -h db

on a PRSIM machine. If no password requires anymore, it means the ".pgpass" file works.

We can then use a Java package *pgpass* courtesy of *technology16* at [GitHub - technology16/pgpass: Simple Java .pgpass file loader](https://github.com/technology16/pgpass) under the *Apache License 2.0*. A copy of this is compiled and attached. Thus, you can

import pgpass.\*;

and call

String passwd = PgPass.get("db", "\*", user, user);

for your program to fetch the *password* from '~/.pgpass' of the person invoking the program.

## The Java 'classpath'

If you have put the JAR file for the driver and the compiled java package *pgpass* under a directory *your\_dirc*, you need to let your java compiler know where it is.

You can specify this on the command line when you invoke the compile:

% javac -cp 'your\_dirc/\*:your\_dirc/:.' AddPurchase.java

Or, add it to your 'CLASSPATH'.

Working with csh or csh-related (e.g, tcsh),

% setenv CLASSPATH ${CLASSPATH}:'your\_dirc/\*:your\_dirc/'

if the environment variable already exists, or, say,

% setenv CLASSPATH % 'your\_dirc/\*:your\_dirc/:.'

if it doesn't.

Of course, you can modify your command shell's *init* file (e.g., .cshrc) so that this is done automatically for each new shell you launch.

If you are a sh or bash user (or zsh, etc.),

% export CLASSPATH=${CLASSPATH}:'your\_dirc/\*:your\_dirc/'

if the environment variable already exists, or

% export CLASSPATH='your\_dirc/\*:your\_dirc/:.'

if not.

And, of course, you could modify your command shell's *init* file (e.g., .bash).

The IDE **ECLIPSE** has a menu option for adding paths to its internal CLASSPATH.

We can't cover all the cases here, you should seek out documentations for your environment.