# Setup:

In order to use the program, the command **source ~db2leduc/cshrc.runtime** must be issued otherwise the program will throw errors. Also, the program must be unzipped which can be done using **unzip asg2.zip**. Furthermore, the database itself must already be built which can be done using the yrb-create file. Furthermore, I have added a make file and the command **make create** can be issued. This will connect to db2, build the db and disconnect. A similar command **make drop** has been offered to remove the db. After that, to compile the program, one can use **javac CustDB.java** or simply type **make** to have the makefile handle compilation. Finally, to run the program, one can type **java CustDB.** I have also included a **make clean** target that can be used to remove the class file when you are done with it.

# Walkthrough:

Upon instantiation of the program, the user is greeted with some basic options for the program. The user is informed that they can terminate the program at any input using the **exit** command. This will commit any changes to the db and close all connections. Similarly, if the user does not want to commit their changes, they can use the **abort** command which does the same thing except it does a rollback instead of a commit. The other instruction states that for any (y/n) trigger, the user can enter any value starting with a **y** or **Y**, so y, Y, yes, YELLOW would all be all be treated as yes, any other value whether it starts with a n or not will be treated as no.

Next, the program asks for a customer ID, any value that is not a basic integer, real numbers, spaces, characters, etc. will tell the user that the input is invalid and prompt for a new input. If the user inserts a value that is not in the db, it will inform the user that there is no customer with that ID and prompt for another ID. Upon entering a valid ID, the program returns the customer's name and city and asks if that is the correct user if the user selects no, they are given the opportunity to insert a new ID.

If the user selects yes, they are presented with the opportunity to update the customer's user name. If they choose yes, they can then enter the name. If the name exceeds 20 characters, the name will be rejected. A side note, the user cannot change the name to either **exit** or **abort** because they are kill words. In theory a prompt could be used to allow those words to be entered as names but since no one is actually named exit or abort, I figured it was unnecessary. After the name is entered, it is displayed back to the user and they can confirm whether it was entered correctly or not. If the user says no, they are prompted again whether they would like to update the user name. If the user is satisfied with the name change or doesn't want to change the name, they are then asked if they want to update the user city. The process is the same, so I won't bother, the only difference is the city cannot exceed 15 characters. The user cannot update the cid since it is a primary key and doing so would be impossible.

Afterwards, the user is provided a list of categories, the user simply needs to enter the number next to the category to select it. This calls the same method as the input for customer ID, so I won't go into any bad input checking, the only key difference is it does not query the db but checks against the existing size of list already provided by the query that finds all the categories. After the user inputs a number, it informs the user what they chose, if they are satisfied with their choice it moves on, otherwise they can insert a different number.

Here, the user is given a list of every book in the given category along with the year, language and weight of the book. The behavior is the same as that of the category list, so I won't go into great detail. At this point, the user is informed what is the cheapest price available to them from all of their clubs and which club they are actually purchasing from. They are then asked for a quantity of the book that they would like to purchase. The behavior is the same as all other number based inputs, the only limiting factor this time is the size of smallint which is 32,767. Why a customer would need 32,767 copies of one book is beyond me, but the option is there. After confirming the quantity, the customer is given the total cost of the order and is asked if they would like to make a purchase.

At this point, the user is given one last chance to confirm all changes, if they select yes, all changes made are committed to the db, otherwise, the transaction is rolled back. This behavior is similar to calling **exit** or **abort** any time in the program. Whether the program finished, or an **exit/abort** command was issued, the program then asks the user if they would like to restart the program. If they select yes, the program runs again starting from the input for customer ID, otherwise it terminates

# Errors:

While this should not occur in testing unless you're actively trying to break the program. In the event, the user updated the database, either the city or name, then presses ctrl+z during the program causing the program to suspend in the background. If the data had not been committed or rolled back at that point and the user attempts to make another change to name or city, the database will still be locked and will be unable to do so. I have set a timeout that if any query takes longer than 5 seconds, the program will throw an SQLError and terminate(rollback/disconnect). The only way to fix this is to ensure the suspended process is correctly terminated, either being killed by a signal like **pkill -9 java**, or by being restored using **fg** and the corresponding **job** number. Also, the program does not like receiving and end of file command ctrl+d, but I have managed to set it so that It closes the connection and does a rollback should that happen.

```
Enter Customer Name: Josh
You have entered Name: Josh
Is this correct(y/n)? y
SQL#2 failed to update
com.ibm.db2.jcc.am.SqlException: DB2 SQL Error: SQLCODE=-952, SQLSTATE=57014, SQ
LERRMC=null, DRIVER=3.71.22
red 351 %
```

Here we have the program failing to update the customer name due to another version of the program running in the background.

# The program in action:

```
Welcome, You may use "exit" to quit the program and save at any time. Alternativ
ely, you can use "abort" to exit without saving. Note, in the prompt to run the
program again, "abort" will exit but has already commited. For any (y/n) choice,
 any input beginning with a y or Y will be accepted. Alternatively, any other in
put will be treated as n.
Please enter a customer ID:
  is not a valid ID ,please enter a number:
 is not a valid ID ,please enter a number: 4.0
4.0 is not a valid ID ,please enter a number: r
r is not a valid ID ,please enter a number: 4
4  is not a valid ID ,please enter a number: 99
99 is not in the database.
Please enter a customer ID: -1
-1 is not in the database.
Please enter a customer ID: 0
0 is not in the database.
Please enter a customer ID: 4
Customer #4, Name: Suzy Sedwick, City: Williamsburg
Is this correct? n
Please enter a customer ID: 4
Customer #4, Name: Suzy Sedwick, City: Williamsburg
Is this correct? y
Update Customer Name(y/n)?
```

Here we have the start of the program and a standard int based input, here we can see it rejecting a space, return, a real number, a char, an int followed by a space. For 99. -1 and 0, it does query the db but returns an invalid entry in the db. Finally, we can see the user saying no to the input and trying again.

```
Would you like to restart the program (y/n)? y
Please enter a customer ID: exit
Would you like to restart the program (y/n)? y
Please enter a customer ID: abort
Would you like to restart the program (y/n)? n
red 357 %
```

Here, we have the **exit** and **abort** being passed to an int input, both give the user a chance to restart the program or leave. All the other int based inputs behave the same with the exception of warning messages so I'm not going to bother with screen shots of them.

```
Customer #4, Name: Suzy Sedwick, City: Williamsburg
Is this correct? y
Update Customer Name(y/n)? y
Enter Customer Name: Suzy Sed
You have entered Name: Suzy Sed
Is this correct(y/n)? y
Name updated
Update Customer City(y/n)? y
Enter Customer City: Williamsburg, NY
Williamsburg, NY can not be longer than 15 characters
Enter Customer City: Willburg
You have entered City: Willburg
Is this correct(y/n)? y
City updated
1: children
2: cooking
3: drama
4: guide
5: history
6: horror
7: humor
8: mystery
9: phil
10: romance
11: science
12: travel
Choose a category by number: exit
Would you like to restart the program (y/n)? y
Please enter a customer ID: 4
Customer #4, Name: Suzy Sed, City: Willburg
```

```
Please enter a customer ID: 4
Customer #4, Name: Suzy Sed, City: Willburg
Is this correct? YELLOW
Update Customer Name(y/n)? yes
Enter Customer Name: name too long!!!!!!!!!!!!!!!
name too long!!!!!!!!!!!!!!! can not be longer than 20 characters
Enter Customer Name: Suzy
You have entered Name: Suzy
Is this correct(y/n)? BANANA
Update Customer Name(y/n)? y
Enter Customer Name: Suzy
You have entered Name: Suzy
Is this correct(y/n)? y
Name updated
Update Customer City(y/n)? abort
Would you like to restart the program (y/n)? y
Please enter a customer ID: 4
Customer #4, Name: Suzy Sed, City: Willburg
```

In the first screen shot, we have the customer's info being changed, we can also see it rejecting a city name that is too long. Exit is then called and the customer's new info has been saved. In the second image, we can see a name being too long, there are also various inputs being accepted as yes, such as yes, y and YELLOW, and the input BANANA being treated as no. Finally, we can see the abort command exits out and does not save the changes.

```
Customer #4, Name: Suzy Sedwick, City: Williamsburg
Is this correct? y
Update Customer Name(y/n)? n
Update Customer City(y/n)? n
1: children
2: cooking
3: drama
4: guide
5: history
6: horror
7: humor
8: mystery
9: phil
10: romance
11: science
12: travel
Choose a category by number: 3
You have selected drama, is that correct(y/n)? y
1: Title: Flibber Gibber, Year: 2000, Language: English, Weight: 51
2: Title: Oberon, Year: 1963, Language: Greek, Weight: 237
3: Title: Yon-juu Hachi, Year: 1948, Language: Japanese, Weight: 459
4: Title: Tchuss, Year: 1998, Language: German, Weight: 265
5: Title: Please Beam Me Up, Year: 2000, Language: Plutonian, Weight: 250
6: Title: Bobbie Sue Ellen, Year: 1995, Language: English, Weight: 247
7: Title: The Earth is not Enough, Year: 1999, Language: Plutonian, Weight: 393
8: Title: Brats like Us, Year: 1995, Language: English, Weight: 338
9: Title: Press the Red Button!, Year: 1999, Language: Plutonian, Weight: 564
10: Title: Gigabytes still to go, Year: 1997, Language: English, Weight: 227
Enter the number of a book: 2
You have selected Oberon, 1963, is that correct(y/n)? y
Oberon 1963 has price: $22.95 from: VaTech Club
Enter a desired quantity: 3
You have selected 3, is that correct(y/n)? y
Total is: $68.85
Would you like to make a purchase (y/n)? y
Purchase inserted
Would you like to save all changes (y/n)? y
Would you like to restart the program (y/n)? n
red 312 %
```

Here we can see a run through the program, the user is selected, no information is updated and then a category is select. The books are then listed, the user selects a book and is given the lowest price and the club which they can get that price from. The user asks for 3 and is given a total price. At this point the user purchases the book, is asked if they wish to save, saying no would be the equivalent of calling **abort**, and exits out of the program.

```
db2 => select * from yrb_purchase where cid = 4

CID    CLUB            TITLE                     YEAR   WHEN                       QNTY
------ --------------- ------------------------- ------ -------------------------- ------
     4 Oprah           Chats ne sont pas Chiens    1992 2001-06-30-13.58.00.000000      1
     4 Oprah           Eigen Eigen                 1980 2001-08-11-17.40.00.000000      1
     4 Oprah           Food for Dummies            2000 2001-06-30-13.58.00.000000      1
     4 Oprah           Rigor Mortis                1023 2001-06-30-13.58.00.000000      1
     4 YRB Silver      Are my feet too big?        1989 2000-06-13-09.45.00.000000      1

  5 record(s) selected.
```

```
db2 => select * from yrb_purchase where cid = 4

CID    CLUB            TITLE                     YEAR   WHEN                       QNTY
------ --------------- ------------------------- ------ -------------------------- ------
     4 Oprah           Chats ne sont pas Chiens    1992 2001-06-30-13.58.00.000000      1
     4 Oprah           Eigen Eigen                 1980 2001-08-11-17.40.00.000000      1
     4 Oprah           Food for Dummies            2000 2001-06-30-13.58.00.000000      1
     4 Oprah           Rigor Mortis                1023 2001-06-30-13.58.00.000000      1
     4 VaTech Club     Oberon                      1963 2017-11-30-21.50.03.000000      3
     4 YRB Silver      Are my feet too big?        1989 2000-06-13-09.45.00.000000      1

  6 record(s) selected.
```

Here we can see the default entry for the above customer as provided by the yrb-create file and the same query made after the program was run and the purchase was inserted into the table.

# Tools used in the project:

The entirety of the program was coded entirely in Notepad++ which automatically saves to my account on Prism and tested running SSH on red. API tools include:

**java.sql.SQLException;** Used to handle any Exceptions thrown by the db.

**java.sql.Timestamp;** Used to get the time to be inserted into the db.

**java.sql.Connection;** Used to connect to the db.

**java.sql.ResultSet;** Used to retrieve the results of any query and pass it to local variables.

**java.sql.PreparedStatement;** Used to convert the query in strong form to something the db can process.

**java.sql.DriverManager;** Used to allow a connection to the db.

**java.util.Scanner;** Used to accept input from the user.

**java.util.ArrayList;** Used to store the result of queries that return more than 1 tuple.

**java.util.NoSuchElementException;** Used to catch if the user hits ctrl+d , to handle the program crashing more gracefully.