

# Project II: Search And Purchase APP

04/05/2017

**Table of Contents**

Source Program.....	2
Recorded Session of Executing the Program.....	19

**Source Program**

```

import java.util.*;
import java.net.*;
import java.text.*;
import java.lang.*;
import java.io.*;
import java.sql.*;

public class SearchAndPurchase {
    private Connection conDB; // Connection to the database system.
    private String url;      // URL: Which database?

    private Integer cCid;    // Who are we tallying?
    private String cName;   // Name of that customer.
    private String cCity;   // City customer lives in.
    private List<String> cat = new ArrayList<String>(); //List of available categories.
    private List<Integer> book = new ArrayList<Integer>(); //List of books searched by user.
    private String bTitle;  //Title of the book
    private Integer bYear;  //Year book was published
    private String bLanguage; //Language of the book
    private Integer bWeight; //Weight of the book
    private Integer bIndex; //index of the book in the list of books.
    private float minPrice; //the minimum price offered for the customer.
    private String mClub;  //the club that offered the minimum price for the customer.
    private Integer pQuantity; //the number of books the customer purchases.
    private Integer tuplesInserted; //number of new rows added as a result of the purchase.

    int id = -1; //user id input condition
    String category; // user category input
    String title; //user title input
    int index = -1; //user book index input
    int quantity = -1; //user quantity input
    String answer; //user purchase input

    // Constructor
    public SearchAndPurchase (String[] args) {
        // Set up the DB connection.
        try {
            // Register the driver with DriverManager.
            Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
        } catch (ClassNotFoundException e) {

```

```

        e.printStackTrace();
        System.exit(0);
    } catch (InstantiationException e) {
        e.printStackTrace();
        System.exit(0);
    } catch (IllegalAccessException e) {
        e.printStackTrace();
        System.exit(0);
    }
}

// URL: Which database?
url = "jdbc:db2:c3421m";

// Initialize the connection.
try {
    // Connect with a fall-thru id & password
    conDB = DriverManager.getConnection(url);
} catch (SQLException e) {
    System.out.print("\nSQL: database connection error.\n");
    System.out.println(e.toString());
    System.exit(0);
}

//Who are we tallying?
Scanner input = new Scanner(System.in);
PrintStream output = System.out;
boolean exists = false;
while(!exists) {
    while (id <= 0) {
        System.out.print("Enter a cid: ");
        String line = input.nextLine();
        int tokens = new StringTokenizer(line, " ").countTokens();
        if (tokens == 1){
            String tempID = new StringTokenizer(line, " ").nextToken();
            if (isInteger(tempID)) {
                id = Integer.parseInt(tempID);
                if (id > 0) {
                    cCid = id;
                }
            } else {
                System.out.println("Provide an id greater than 0");
            }
        }
        else {
            System.out.println("Provide an INT for the cid");
        }
    }
    else {
        System.out.println("Provide 1 argument");
    }
}

```

```

        }
    }

// Is this cCid for real?
if (customerCheck()) {
    exists = true;
}
else {
    System.out.print("There is no customer #");
    System.out.print(cCid);
    System.out.println(" in the database.");
    id = -1;
}
}

//report the customer info
find_customer();

//show the categories to choose from
System.out.println("Welcome " + cName + ", please choose one of the categories below:\n");
boolean search = false;
while(!search) {
    fetch_categories();
    exists = false;
while(!exists) {
    System.out.print("Enter a category: ");
    String line = input.nextLine();
    int tokens = new StringTokenizer(line, " ").countTokens();
    if (tokens == 1){
        String tempCategory = new StringTokenizer(line, " ").nextToken();
        if (!isInteger(tempCategory)) {
            for (int i = 0; i < cat.size(); i++) {
                if (tempCategory.equals(cat.get(i))) {
                    category = tempCategory;
                    i = cat.size();
                    exists = true;
                }
            }
            else if (i == cat.size() - 1) {
                System.out.println("Invalid category");
            }
        }
    }
    else {
        System.out.println("Invalid category");
    }
}
else {
    System.out.println("Provide 1 argument");
}
}

```

```

    }

    //What is the book title?
    System.out.print("\n" + "Enter the title of the book you are looking for: ");
    title = input.nextLine();

    //Is the book and category real?
    if(bookCheck()) {
        search = true;
        bTitle = title;
    }
    else {
        System.out.println("The book does not exist in this category");
        System.out.println("Please choose another category and title from the list below: \n");
    }
}

//return the list of matching books
find_book();

//choose which book to choose from
exists = false;
while(!exists) {
    while (index <= 0) {
        System.out.print("Choose a book from the list above by entering its index (left number):
");

        String line = input.nextLine();
        int tokens = new StringTokenizer(line, " ").countTokens();
        if (tokens == 1){
            String tempIndex = new StringTokenizer(line, " ").nextToken();
            if (isInteger(tempIndex)) {
                index = Integer.parseInt(tempIndex);
                if (index > 0) {
                    for (int i = 1; i <= book.size(); i++) {
                        if (index == i) {
                            bIndex = index;
                            i = book.size();
                            exists = true;
                        }
                        else if (i == book.size()) {
                            System.out.println("Provide an index that exists");
                            index = -1;
                        }
                    }
                }
            }
            else {
                System.out.println("Provide an index greater than 0");
            }
        }
    }
}

```

```

        else {
            System.out.println("Provide an INT for the index");
        }
    }
    else {
        System.out.println("Provide 1 argument");
    }
}

//get the club that offered this minimum price (for later use)
min_club();

//report the minimum price for the chosen book
min_price();

//how many books to buy?
while (quantity <= 0) {
    System.out.print("\n" + "How many books would you like to buy? ");
    String line = input.nextLine();
    int tokens = new StringTokenizer(line, " ").countTokens();
    if (tokens == 1){
        String tempQuantity = new StringTokenizer(line, " ").nextToken();
        if (isInteger(tempQuantity)) {
            quantity = Integer.parseInt(tempQuantity);
            if (quantity > 0) {
                pQuantity = quantity;
            }
        }
        else {
            System.out.println("Provide an id greater than 0");
            quantity = -1;
        }
    }
    else {
        System.out.println("Provide an INT for the quantity");
    }
}
else {
    System.out.println("Provide 1 argument");
}
}

//Customer checkout
DecimalFormat df = new DecimalFormat("#####0.00");
float totalPrice = pQuantity * minPrice;
System.out.println("\n" + "The total price is: " + df.format(totalPrice));
exists = false;
while(!exists) {
    System.out.print("\n" + "Would you like to purchase the books? ");

```

```

String line = input.nextLine();
int tokens = new StringTokenizer(line, " ").countTokens();
if (tokens == 1){
    String tempAnswer = new StringTokenizer(line, " ").nextToken();
    if (!Integer.parseInt(tempAnswer)) {
        answer = tempAnswer;
        if (answer.equals("Y")) {
            //add the new tuple to the database
            insert_purchase();
            //System.out.print("The database has been updated with the following
purchase: (");

            //System.out.print(cCid + ", ");
            //System.out.print(mClub + ", ");
            //System.out.print(bTitle + ", ");
            //System.out.print(bYear + ", ");
            //Timestamp when = answers.getTimestamp("when");
            //System.out.print(when + ", ");
            //System.out.print(pQuantity + ")");
            System.out.println("The database has been updated with this purchase");
            System.out.println(tuplesInserted + " new row(s) has been added to the
yrb_purchase table.");

            exists = true;
        }
        else if (answer.equals("N")) {
            exists = true;
            System.out.println("The app will now terminate");
        }
        else {
            System.out.println("Please answer Y or N");
        }
    }
    else {
        System.out.println("Please answer Y or N");
    }
}
else {
    System.out.println("Provide 1 argument");
}
}

// Close the connection.
try {
    conDB.close();
} catch (SQLException e) {
    System.out.print("\nFailed trying to close the connection.\n");
    e.printStackTrace();
    System.exit(0);
}
}

```

```

}

//checks if a string is a valid integer
public static boolean isInteger(String s) {
    boolean isValidInteger = false;
    try
    {
        Integer.parseInt(s);

        // s is a valid integer

        isValidInteger = true;
    }
    catch (NumberFormatException ex)
    {
        // s is not an integer
    }

    return isValidInteger;
}

//read input
public void readID() {
    Scanner input = new Scanner(System.in);
    PrintStream output = System.out;
    while (id == -1) {
        System.out.print("Enter a cid: ");
        if (input.hasNextInt()) {
            id = input.nextInt();
            if (id > 0) {
                cCid = id;
            }
        }
        else {
            System.out.println("Provide an id greater than 0");
            input.next();
        }
    }
    else {
        System.out.println("Provide an INT for the cid");
        input.next();
    }
}
}

```

```

//from the example provided, checks if the a customer exists with a given cid
public boolean customerCheck() {
    String      queryText = ""; // The SQL text.
    PreparedStatement querySt = null; // The query handle.
    ResultSet    answers = null; // A cursor.

```



```
boolean    inDB    = false; // Return.
```

```
queryText =  
    "SELECT name    "  
    + "FROM yrb_customer "  
    + "WHERE cid = ?    ";
```

```
// Prepare the query.
```

```
try {  
    querySt = conDB.prepareStatement(queryText);  
} catch(SQLException e) {  
    System.out.println("SQL#1 failed in prepare");  
    System.out.println(e.toString());  
    System.exit(0);  
}
```

```
// Execute the query.
```

```
try {  
    querySt.setInt(1, cCid.intValue());  
    answers = querySt.executeQuery();  
} catch(SQLException e) {  
    System.out.println("SQL#1 failed in execute");  
    System.out.println(e.toString());  
    System.exit(0);  
}
```

```
// Any answer?
```

```
try {  
    if (answers.next()) {  
        inDB = true;  
        cName = answers.getString("name");  
    } else {  
        inDB = false;  
        cName = null;  
    }  
} catch(SQLException e) {  
    System.out.println("SQL#1 failed in cursor.");  
    System.out.println(e.toString());  
    System.exit(0);  
}
```

```
// Close the cursor.
```

```
try {  
    answers.close();  
} catch(SQLException e) {  
    System.out.print("SQL#1 failed closing cursor.\n");  
    System.out.println(e.toString());  
    System.exit(0);  
}
```

```

    }

    // Close the handle.
    try {
        querySt.close();
    } catch(SQLException e) {
        System.out.print("SQL#1 failed closing the handle.\n");
        System.out.println(e.toString());
        System.exit(0);
    }

    return inDB;
}

public void find_customer() {
    String queryText = "";
    PreparedStatement querySt = null;
    ResultSet answers = null;

    queryText = "SELECT cid, name, city FROM yrb_customer WHERE cid = ?";

    //Prepare the query
    try {
        querySt = conDB.prepareStatement(queryText);
    } catch(SQLException e) {
        System.out.println("SQL#2 failed in prepare");
        System.out.println(e.toString());
        System.exit(0);
    }

    //Execute the query
    try {
        querySt.setInt(1, cCid.intValue());
        answers = querySt.executeQuery();
    } catch(SQLException e) {
        System.out.println("SQL#2 failed in execute");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Walk through the results and present them.
    try {
        System.out.print("\n" + cCid + " ");
        System.out.print(cName + " ");
        if (answers.next()) {
            cCity = answers.getString("city");
            System.out.println(cCity);
        }
        System.out.println("");
    }
}

```

```

    } catch(SQLException e) {
        System.out.println("SQL#2 failed in cursor.");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Close the cursor.
    try {
        answers.close();
    } catch(SQLException e) {
        System.out.print("SQL#2 failed closing cursor.\n");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Close the handle.
    try {
        querySt.close();
    } catch(SQLException e) {
        System.out.print("SQL#2 failed closing the handle.\n");
        System.out.println(e.toString());
        System.exit(0);
    }
}

```

```

public void fetch_categories() {
    String queryText = "";
    PreparedStatement querySt = null;
    ResultSet answers = null;

    queryText = "SELECT cat FROM yrb_category";

    //Prepare the query
    try {
        querySt = conDB.prepareStatement(queryText);
    } catch(SQLException e) {
        System.out.println("SQL#3 failed in prepare");
        System.out.println(e.toString());
        System.exit(0);
    }

    //Execute the query
    try {
        answers = querySt.executeQuery();
    } catch(SQLException e) {
        System.out.println("SQL#3 failed in execute");
        System.out.println(e.toString());
        System.exit(0);
    }
}

```

```

// Walk through the results and present them.
try {
    while(answers.next()) {
        cat.add(answers.getString("cat"));
        System.out.println(answers.getString("cat"));
    }
    System.out.println("");
} catch(SQLException e) {
    System.out.println("SQL#3 failed in cursor.");
    System.out.println(e.toString());
    System.exit(0);
}

// Close the cursor.
try {
    answers.close();
} catch(SQLException e) {
    System.out.print("SQL#3 failed closing cursor.\n");
    System.out.println(e.toString());
    System.exit(0);
}

// Close the handle.
try {
    querySt.close();
} catch(SQLException e) {
    System.out.print("SQL#3 failed closing the handle.\n");
    System.out.println(e.toString());
    System.exit(0);
}
}

public boolean bookCheck() {
    String      queryText = ""; // The SQL text.
    PreparedStatement querySt = null; // The query handle.
    ResultSet    answers = null; // A cursor.

    boolean      inDB = false; // Return.

    queryText = "SELECT title, cat FROM yrb_book WHERE title = ? and cat = ?";

    // Prepare the query.
    try {
        querySt = conDB.prepareStatement(queryText);
    } catch(SQLException e) {
        System.out.println("SQL#4 failed in prepare");
        System.out.println(e.toString());
        System.exit(0);
    }
}

```

```

}

// Execute the query.
try {
    querySt.setString(1, title);
    querySt.setString(2, category);
    answers = querySt.executeQuery();
} catch(SQLException e) {
    System.out.println("SQL#4 failed in execute");
    System.out.println(e.toString());
    System.exit(0);
}

// Any answer?
try {
    if (answers.next()) {
        inDB = true;
        bTitle = answers.getString("title");
        category = answers.getString("cat");
    } else {
        inDB = false;
        title = null;
        category = null;
    }
} catch(SQLException e) {
    System.out.println("SQL#4 failed in cursor.");
    System.out.println(e.toString());
    System.exit(0);
}

// Close the cursor.
try {
    answers.close();
} catch(SQLException e) {
    System.out.print("SQL#4 failed closing cursor.\n");
    System.out.println(e.toString());
    System.exit(0);
}

// Close the handle.
try {
    querySt.close();
} catch(SQLException e) {
    System.out.print("SQL#4 failed closing the handle.\n");
    System.out.println(e.toString());
    System.exit(0);
}

return inDB;

```

```

}

public void find_book() {
    String queryText = "";
    PreparedStatement querySt = null;
    ResultSet answers = null;

    queryText = "SELECT title, year, language, weight FROM yrb_book WHERE title = ? and cat
= ?";

    //Prepare the query
    try {
        querySt = conDB.prepareStatement(queryText);
    } catch(SQLException e) {
        System.out.println("SQL#5 failed in prepare");
        System.out.println(e.toString());
        System.exit(0);
    }

    //Execute the query
    try {
        querySt.setString(1, bTitle);
        querySt.setString(2, category);
        answers = querySt.executeQuery();
    } catch(SQLException e) {
        System.out.println("SQL#5 failed in execute");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Walk through the results and present them.
    try {
        while(answers.next()) {
            int i = 1;
            book.add(answers.getInt("year"));
            System.out.println("");
            System.out.print(i + " ");
            System.out.print(bTitle + " ");
            bYear = answers.getInt("year");
            System.out.print(bYear + " ");
            bLanguage = answers.getString("language");
            System.out.print(bLanguage + " ");
            bWeight = answers.getInt("weight");
            System.out.println(bWeight + " ");
            i++;
        }
        System.out.println("");
    } catch(SQLException e) {
        System.out.println("SQL#5 failed in cursor.");
    }
}

```

```

        System.out.println(e.toString());
        System.exit(0);
    }

    // Close the cursor.
    try {
        answers.close();
    } catch(SQLException e) {
        System.out.print("SQL#5 failed closing cursor.\n");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Close the handle.
    try {
        querySt.close();
    } catch(SQLException e) {
        System.out.print("SQL#5 failed closing the handle.\n");
        System.out.println(e.toString());
        System.exit(0);
    }
}

public void min_price() {
    String queryText = "";
    PreparedStatement querySt = null;
    ResultSet answers = null;

    queryText = "select min(price) as minprice from (select club from yrb_member where cid = ?)
a, yrb_offer o, yrb_book b where a.club = o.club and o.year = b.year and o.title = ? and b.cat = ?";

    //Prepare the query
    try {
        querySt = conDB.prepareStatement(queryText);
    } catch(SQLException e) {
        System.out.println("SQL#6 failed in prepare");
        System.out.println(e.toString());
        System.exit(0);
    }

    //Execute the query
    try {
        querySt.setInt(1, cCid);
        querySt.setString(2, bTitle);
        querySt.setString(3, category);
        answers = querySt.executeQuery();
    } catch(SQLException e) {
        System.out.println("SQL#6 failed in execute");
        System.out.println(e.toString());
    }
}

```

```

        System.exit(0);
    }

    //Variable to hold column value(s).
    DecimalFormat df = new DecimalFormat("####0.00");

    // Walk through the results and present them.
    try {
        if (answers.next()) {
            minPrice = answers.getFloat("minprice");
            System.out.print("\n" + "Based on your membership at " + mClub + ", ");
            System.out.print("the minimum price offered is: " + df.format(minPrice));
        }
        else {
            System.out.println("Based on your membership at " + mClub + ", ");
            System.out.print("the minimum price offered is: " + df.format(0));
        }
        System.out.println("");
    } catch (SQLException e) {
        System.out.println("SQL#6 failed in cursor.");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Close the cursor.
    try {
        answers.close();
    } catch (SQLException e) {
        System.out.print("SQL#6 failed closing cursor.\n");
        System.out.println(e.toString());
        System.exit(0);
    }

    // Close the handle.
    try {
        querySt.close();
    } catch (SQLException e) {
        System.out.print("SQL#6 failed closing the handle.\n");
        System.out.println(e.toString());
        System.exit(0);
    }
}

//finds the club that offered the minimum price for a given book, category, and customer
public void min_club() {
    String queryText = "";
    PreparedStatement querySt = null;
    ResultSet answers = null;

```



```

        queryText = "select a1.club as minclub from (select a.club, min(price) as minprice from (select
club from yrb_member where cid = ?) a, yrb_offer o, yrb_book b where a.club = o.club and o.year =
b.year and o.title = ? and b.cat = ? group by a.club) a1 where a1.minprice <= all(select minprice from
(select a.club, min(price) as minprice from (select club from yrb_member where cid = ?) a, yrb_offer o,
yrb_book b where a.club = o.club and o.year = b.year and o.title = ? and b.cat = ? group by a.club)
a2)";

```

```

        //Prepare the query
        try {
            querySt = conDB.prepareStatement(queryText);
        } catch(SQLException e) {
            System.out.println("SQL#7 failed in prepare");
            System.out.println(e.toString());
            System.exit(0);
        }

        //Execute the query
        try {
            querySt.setInt(1, cCid);
            querySt.setString(2, bTitle);
            querySt.setString(3, category);
            querySt.setInt(4, cCid);
            querySt.setString(5, bTitle);
            querySt.setString(6, category);
            answers = querySt.executeQuery();
        } catch(SQLException e) {
            System.out.println("SQL#7 failed in execute");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Walk through the results.
        try {
            if (answers.next()) {
                mClub = answers.getString("minclub");
            }
        } catch(SQLException e) {
            System.out.println("SQL#7 failed in cursor.");
            System.out.println(e.toString());
            System.exit(0);
        }

        // Close the cursor.
        try {
            answers.close();
        } catch(SQLException e) {
            System.out.print("SQL#7 failed closing cursor.\n");
            System.out.println(e.toString());
            System.exit(0);
        }

```

```

    }

    // Close the handle.
    try {
        querySt.close();
    } catch(SQLException e) {
        System.out.print("SQL#7 failed closing the handle.\n");
        System.out.println(e.toString());
        System.exit(0);
    }
}

public void insert_purchase() {
    String queryText = "";
    PreparedStatement querySt = null;

    queryText = "INSERT into yrb_purchase values(?, ?, ?, ?, current timestamp, ?)";

    //Prepare the query
    try {
        querySt = conDB.prepareStatement(queryText);
    } catch(SQLException e) {
        System.out.println("SQL#8 failed in prepare");
        System.out.println(e.toString());
        System.exit(0);
    }

    //Update the query
    try {
        querySt.setInt(1, cCid);
        querySt.setString(2, mClub);
        querySt.setString(3, bTitle);
        querySt.setInt(4, bYear);
        querySt.setInt(5, pQuantity);
        tuplesInserted = querySt.executeUpdate();
    } catch(SQLException e) {
        System.out.println("SQL#8 failed in update");
        System.out.println(e.toString());
        System.exit(0);
    }

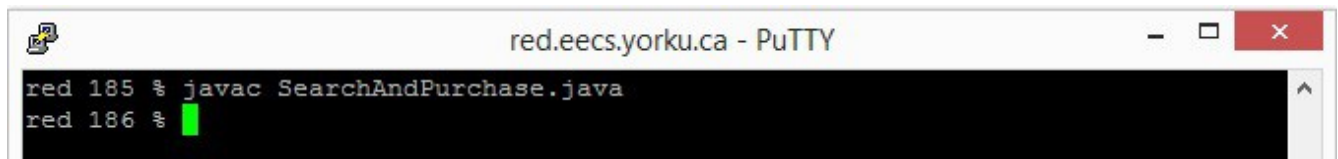
    // Close the handle.
    try {
        querySt.close();
    } catch(SQLException e) {
        System.out.print("SQL#8 failed closing the handle.\n");
        System.out.println(e.toString());
        System.exit(0);
    }
}

```

```
}  
  
public static void main(String[] args) {  
    SearchAndPurchase ct = new SearchAndPurchase(args);  
}  
  
}
```

### Recorded Session of Executing the Program

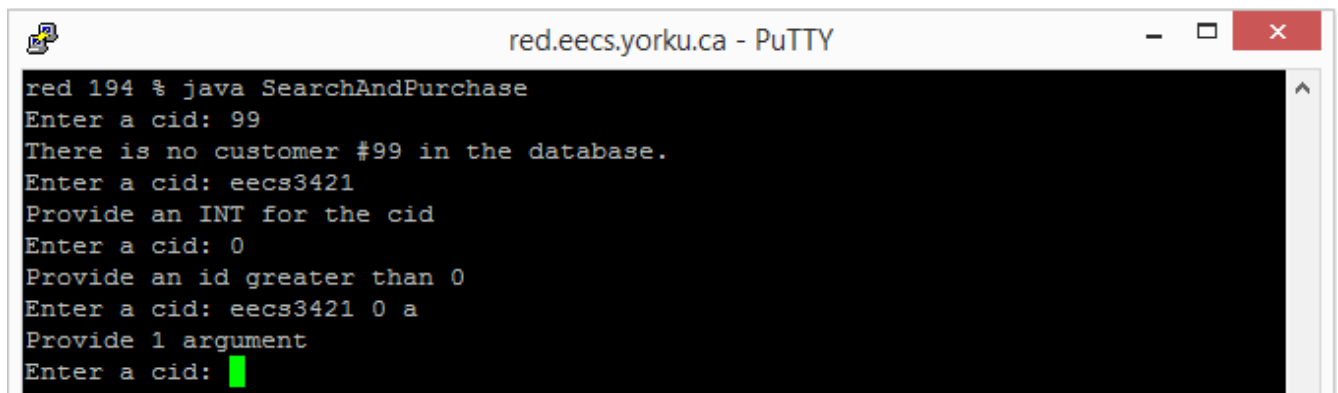
1) Program Compilation: The program (using the source code above) successfully compiled shown below.



```
red.eecs.yorku.ca - PuTTY  
red 185 % javac SearchAndPurchase.java  
red 186 %
```

2) Required: The program starts by finding a customer, that is, it looks for a customer to see if the customer with a given id exists. If the customer does not exist, the program displays an error message and requests the customer id again. If the customer exists, the query returns and displays the customer information (the customer id, name, and city).

Scenarios: The program will accept any integers  $> 0$  as input and reject all other inputs. Such situations include if the user enters a non-integer, an integer  $\leq 0$ , or more than one token, and it will reprompt the user to try again. If the customer does not exist, the program will notify the user and reprompt. The results when the customer exists is shown in the next step.



```
red.eecs.yorku.ca - PuTTY  
red 194 % java SearchAndPurchase  
Enter a cid: 99  
There is no customer #99 in the database.  
Enter a cid: eecs3421  
Provide an INT for the cid  
Enter a cid: 0  
Provide an id greater than 0  
Enter a cid: eecs3421 0 a  
Provide 1 argument  
Enter a cid:
```

3) Required: If the customer exists, write a query in your program to return all the categories (cat) in your database. Then, display all the categories (in a drop-down list), so the customer can choose a category from the list.



```
red.eecs.yorku.ca - PuTTY
Enter a cid: 0
Provide an id greater than 0
Enter a cid: eecs3421 0 a
Provide 1 argument
Enter a cid: 8

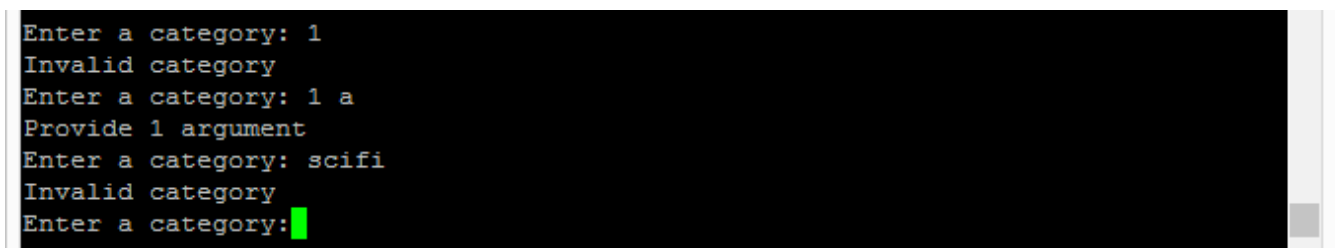
8 Jack Daniels Blacksburg

Welcome Jack Daniels, please choose one of the categories below:

children
cooking
drama
guide
history
horror
humor
mystery
phil
romance
science
travel

Enter a category: █
```

Scenarios: The program will accept any input that matches one of the categories listed and reject all other inputs. Such situations include if the user enters a an integer, more than one token, or a string that doesn't match the category, and it will reprompt the user to try again.

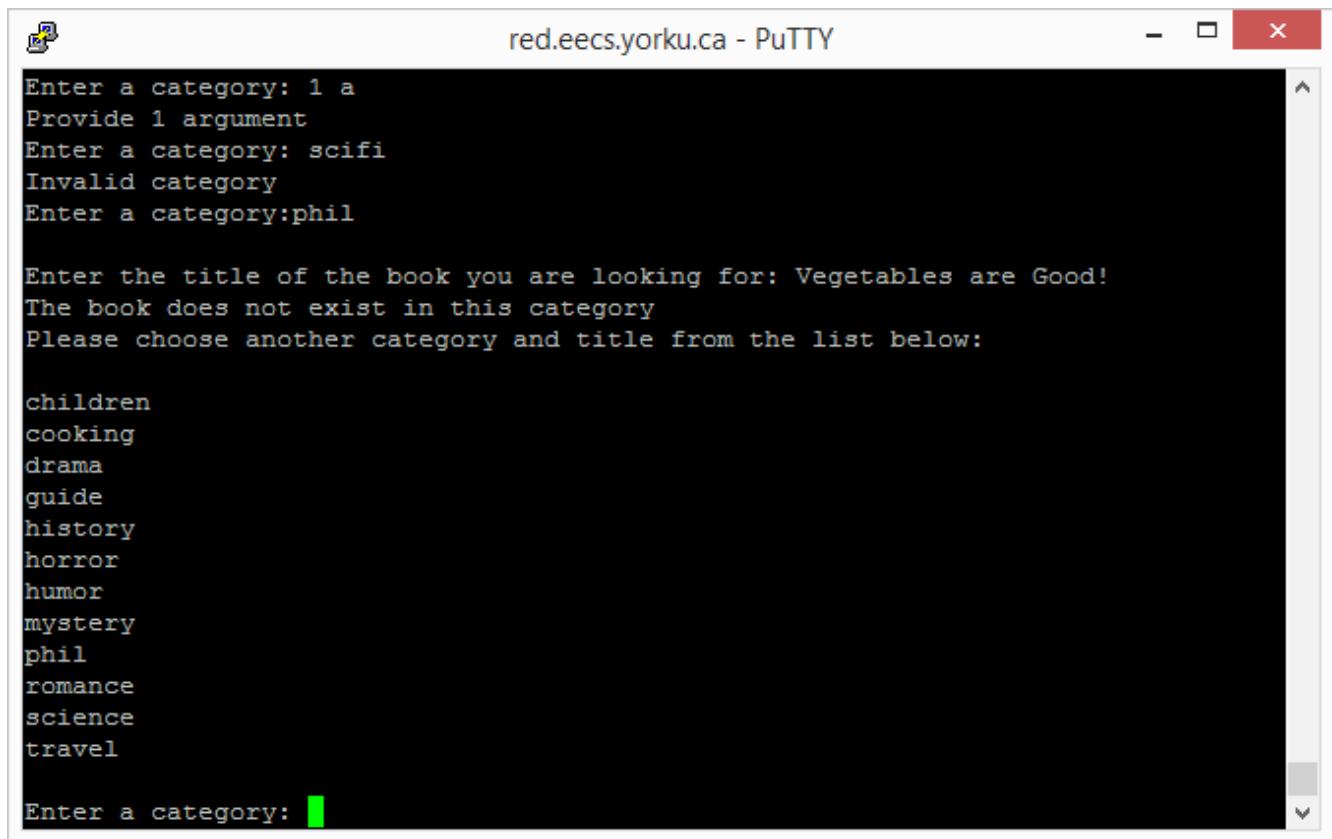


```
Enter a category: 1
Invalid category
Enter a category: 1 a
Provide 1 argument
Enter a category: scifi
Invalid category
Enter a category: █
```

4) Required: After choosing a category by the customer, customer can enter the title of the book. You need to write a query that looks for the book with the given title and the selected category. If the given title with the selected category exists, return the book information (title, year, language, weight). The query for this part may return more than one book. So, display all the books in a list and let the user choose a book from the list. If the book with the given title and the category does not exist, the program lets the user choose another category and enter another title.

Scenarios: The program will accept any input from the customer. It will then check if that title exists in

the given category. If not, the program will prompt the user to choose a new category again before reprompting the user to enter another title.



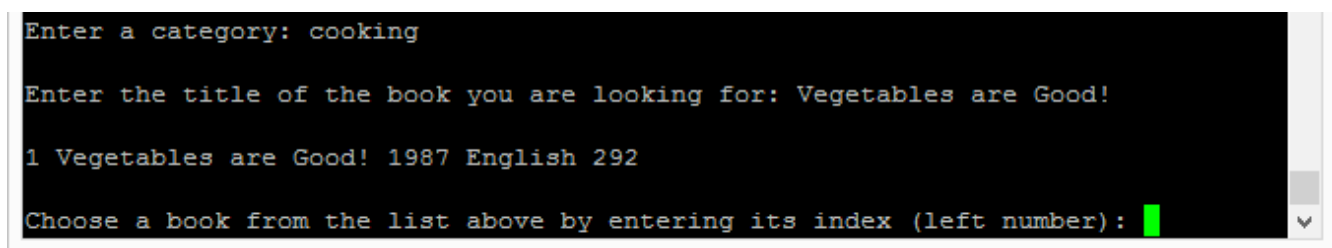
```
red.eecs.yorku.ca - PuTTY
Enter a category: 1 a
Provide 1 argument
Enter a category: scifi
Invalid category
Enter a category: phil

Enter the title of the book you are looking for: Vegetables are Good!
The book does not exist in this category
Please choose another category and title from the list below:

children
cooking
drama
guide
history
horror
humor
mystery
phil
romance
science
travel

Enter a category: █
```

Entering a new category, then entering the title again:



```
Enter a category: cooking

Enter the title of the book you are looking for: Vegetables are Good!

1 Vegetables are Good! 1987 English 292

Choose a book from the list above by entering its index (left number): █
```

5) Required: If the book exists:

Then, the user selects a book from the result of the previous query (or stored procedure) to buy.

After, the user selects a book to buy, the minimum price for that book will be retrieved from the database (yrb\_offer). You need to write a query that returns the minimum price for the book has been offered and display the price to the user. Remember, that each customer is a member of a club that offers books at particular prices.

The minimum price will be displayed to the user.

Ask the user to enter the number of books (the quantity) to buy. After, the user enters the quantity, the total price is calculated and displayed to the user (quantity \* minimum price)

Scenarios: The program will accept any integer matching the indexes of the list it returns and rejects all other inputs. Such situations include if the user enters an integer that does not match any index on the list, any integer  $\leq 0$ , a non-integer, or more than one token (the program will reprompt the user). Once the user properly selects a book from the list, the program will return the minimum price of the book offered to the customer based on their membership of the club that offers that price.

```
Enter the title of the book you are looking for: Vegetables are Good!

1 Vegetables are Good! 1987 English 292

Choose a book from the list above by entering its index (left number): 2
Provide an index that exists
Choose a book from the list above by entering its index (left number): 0
Provide an index greater than 0
Choose a book from the list above by entering its index (left number): a
Provide an INT for the index
Choose a book from the list above by entering its index (left number): 1 a
Provide 1 argument
Choose a book from the list above by entering its index (left number): 1

Based on your membership at Oprah, the minimum price offered is: 25.95

How many books would you like to buy? █
```

When the customer is asked how many books they want to buy, the program will accept an integer and reject all other inputs, reprompting the user. Such situations include if the user enters a number  $\leq 0$ , a non-integer, or more than one token. Once the user enters an integer, the program will return the total price.

```
How many books would you like to buy? 0
Provide a quantity greater than 0

How many books would you like to buy? a
Provide an INT for the quantity

How many books would you like to buy? 2 a
Provide 1 argument

How many books would you like to buy? 2

The total price is: 51.90

Would you like to purchase the books? █
```

6) Required: If the user approves (ask if the user wants to purchase the book/books?), the purchase information will be stored in the purchase table with current date and time. You need to write a query to insert the purchase into the database.

Scenarios: The program will accept “Y” or “N” as input and reject all other input, reprompting the user. Such situations include a string that does not match “Y” or “N”, an integer, or more than one token. If

the user enters “Y”, then the program will update the database, insert the purchase tuple into the table yrb\_purchase, and then terminate. If the user enters “N”, the program does nothing and terminates.

```
How many books would you like to buy? 2

The total price is: 51.90

Would you like to purchase the books? Y
The database has been updated with this purchase
1 new row(s) has been added to the yrb_purchase table.
red 207 %
```

The inserted tuple in yrb\_purchase:

```
red.eecs.yorku.ca - PuTTY

For more detailed help, refer to the Online Reference Manual.

db2 => connect to c3421m

Database Connection Information

Database server      = DB2/LINUX8664 11.1.1
SQL authorization ID = ABAI2
Local database alias = C3421M

db2 => select * from yrb_purchase where cid = 8 and title = 'Vegetables are Good
!'

CID    CLUB          TITLE                YEAR  WHEN
QNTY
-----
--
      8 Oprah          Vegetables are Good!    1987  2017-04-03-02.58.28.4455
71      2

1 record(s) selected.

db2 =>
```

If the user enters “N”:

```
How many books would you like to buy? 2

The total price is: 51.90

Would you like to purchase the books? N
The app will now terminate
red 210 %
```