CMPT 354
Database Systems

Simon Fraser University
Spring 2017

Instructor: Oliver Schulte

Assignment 3b: Application Development, Chapters 6 and 7.

Instructions: Check the instructions in the syllabus. The university policy on academic dishonesty and plagiarism (cheating) will be taken very seriously in this course. *Everything submitted should be your own writing or coding.* You must not let other students copy your work. On your assignment, put down your **name**, the number of the assignment and the number of the course. Spelling and grammar count.

Group Work: Discussions of the assignment is okay, for example to understand the concepts involved. If you work in a group, put down the name of all members of your group. There should be no group submissions. Each group member should write up their own solution to show their own understanding.

For the due date please see our course management server https://courses.cs.sfu.ca .

Additional instructions for what to submit appear at the end of this file.

Systems Issues.

Systems Support. The purpose of this assignment is to give you experience with writing programs that interact with a database management system. You will learn hardly anything from typing in someone else's instructions; you will learn a lot from getting the system to work on your own. Therefore we provide minimal support for getting things to work on your own system. As with other assignments, you can post on the discussion forum, but *there is no email support*. You can also come see us during our office hours on. Start early and ask your questions in class, office hours, or team up with classmates. You can use the discussion forum on courses.cs.sfu.ca .

System Requirements. Please review the instructions for Part 3a for information about what setup to use, where to find documentation, tips and tricks.

For all questions, use the AdventureWorksLT database. Your solutions should follow the general design principles shown in the text and the lecture notes.

Part III: Views

1. Write SQL code that finds customers who have bought a product of color red.

2. Write SQL code that creates a view *RedSpending* on the Customer table with the following specifications.
    a. The fields in the view are CustomerID, FirstName, LastName, HighestPrice, in that order (e.g., CustomerID is the first).
    b. The customers in the view should be those who have bought a product of colour red.
    c. HighestPrice should contain the highest unit price that they have paid for some red product (see SalesOrderDetail; ignore UnitPriceDiscount).

3. Write a small application that does the following when run.

    i. Display on the screen all and only colors of products in the Product table (including "Multi" but excluding Null). The colors should be sorted in alphabetical order.
    ii. Accept a single string input <colour> from the keyboard. This will be one of the colors.
    iii. Create a view table for that <colour>, where the view table meets the criteria a,b,c for that colour. The application should write the result to the screen as plain comma separated text. The output should be sorted in descending order by (LastName, First Name). For instance, customer Catherine Abel should appear below customer Christopher Beck.

Grading Criteria:

Total Marks: 100 Marks
1. SQL query 1 (20 Marks)
2. SQL code for view (40 Marks)
3. Application with Interface (40 Marks)

Part IV: Cursors and HTML

Write a small application that does the following.

1. Display on the screen all and only colors of products in the Product table (including "Multi" but excluding Null). They should be sorted in alphabetical order.
2. Accept a single string input <colour> from the keyboard, and a single input number <price>. This will be one of the colours.
3. If the <price> is less than the average StandardCost for the colour chosen, return a message "Price is too low for <color>" (where you display the name of the color, e.g. "red"). The answer should be computed by calling the stored procedure that you wrote for question II in Assignment 3a.
4. Otherwise the application should do the following:

   Write an **HTML file** that contains the information from the view defined in Part III, sorted in descending order by the value of HighestPrice. So the HTML file should, in valid syntax, contain the information from the fields Customer_ID, FirstName, LastName, HighestPrice, where the customers who have the greatest value for HighestPrice appear first. *This information should appear in an HTML table using the <table>….</table> tag.* The HMTL file should display correctly in a standard browser (Firefox, Internet Explorer, Safari, etc.).

   For part 4, you should use a cursor type of object (depending on your system, a cursor may be called something like resultset, iterator, recordset, datareader, reader) and iterate over the rows in the cursor to produce the HTML file. Several examples of this iteration appear in the lecture notes and the book. Even if your system supports automatically outputting a query result to an HTML table, the point is to give you some practice both with cursors and with the HTML format.

Grading Criteria:

Total Marks: 100 Marks
1. Input Form (15 Marks)
2. Price Error Check (15 Marks)
3. Sorted Output (35 Marks)
4. HTML output (35 Marks)

General Grading Criteria.

- Most application development requires you to make many choices of your own. It is normal that there are several valid solutions, some clearly better than others, some involving trade-offs. Ambiguity is not the same as arbitrariness. An opportunity for you to practice making design choices, dealing with ambiguity and exercising your own judgment is a feature of the assignment, not a bug. We are happy to look at drafts and discuss design choices during the office hours. It's also a good idea to study with other students and discuss (not copy; see syllabus and instructions).

- Code design and documentation are part of the criteria. Remember that your TA may not be an expert in the development system you are using. The code required is so short that having an explanatory comment for *each* line is not overdoing it. In fact, it's a good habit to acquire.

- The burden of proof is on you to convince us that your program works by providing legible code, good documentation and illustrative screenshots. Your code should run in the CSIL environment so we can run it if necessary, but if we have to check it out by running it, your documentation is probably insufficient.

What to Submit

Part III: Views.

Submit the following files:

For components 1 and 2:

1. For the SQL code, a single sql script for both components. This should execute without error on SQL Server. Call this file sqlIII.sql .

For component 3:

2. Your source code sourceIII.*, where * is the file extension required for your development setup. Please include supporting files if needed (like any dll, jar or class files). Your code should be self-contained. If your code comprises several files, please combine them into a single file sourceIII.zip.
3. A readme file with instructions for how to run your code on leto.csil.sfu.ca . Call this file readmeIII.rtf .
4. A screenshot of the session, where we can see the displayed colours, the user choosing a colour, and the system outputting the view. Call this file outputIII.pdf .

   Put all files, including the .sql file, together as an archive solutionIII.zip.

Part IV. Cursors and HTML.

Submit the following files:

1. Your source code sourceIV.*, where * is the file extension required for your development setup. Please include supporting files if needed (like any dll, jar or class files). Your code should be self-contained. If your code comprises several files, please combine them into a single file sourceIV.zip.
2. A screenshot of the session, where we can see the displayed colours, the user choosing a colour, and a price, and the html/xml file produced by your program opened in a browser. Show the interaction for two different colours, one for each of the two possible responses. (e.g., color = 'red', price is too low, and color = 'grey', price is okay). Call this file outputIV.pdf .
3. Also include the html file generated by your program, call it outputIVa.html.

   Put all files together as an archive solutionIV.zip . Then you are finished!