CMPT 354
Database Systems

Simon Fraser University
Summer 2016

Instructor: Oliver Schulte

Assignment 3: Application Development.

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. 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. On your assignment, put down your **name**, the number of the assignment and the number of the course. Spelling and grammar count.

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

Additional instructions for what to submit appear in a separate file on the course website.

There will be no email support for this assignment. 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 .

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. I suggest you get started on this assignment early to check your basic system setup.

- *System Requirements*. Our basic requirement is that **we should be able to run your solution on the CSIL server.** This means that you should use one of the development setups available in CSIL. That's the only constraint. For a list of what's available in CSIL, see http://www.cs.sfu.ca/cc/CSILPC/software.html . General info about CSIL is posted at http://www.cs.sfu.ca/cc/Labs/. You can of course develop your solution on your home system and then test it on the CSIL set-up. Tip: Ask a friend to run your code on CSIL so you know someone else can run it with their settings. A few suggestions for planning your work.

1. The basic architecture required for the assignment is a client-server architecture where the client functions provide an interface for accepting user input, and the server is the SQL server. The program structures required are simple, so the main new challenges for you are a) the presentation layer and b) the interaction with the server. You may want to choose a development set-up that makes this as easy as possible. I list some of the common environments used by students, starting with the easiest first.
    a. *Python.* The CSIL Python installation comes with the database package you need (see lecture slides). You just need to import it. *This is by far the easiest for students.* One student wrote a comment that: "Time to do this assignment in Java – infinite. Time to do it in Python – 15 min".
    b. *Visual Studio* provides a graphical interface for creating forms, buttons etc. We also provide some support by way of example code and a flash video for it. If you are used to Java programming, but not to creating forms for user input or connecting to the database server, your best bet is probably to use Visual C#: writing code is easy to learn and Visual Studio simplifies the client-server tasks.
    c. *Java.* CSIL has posted instructions for database connectivity with Java (JDBC).

2. In addition to examples and general principles covered in the class and the text, there is much documentation available with the common development tools, such as Visual Studio and in on-line discussion. You should not look for the specific solution, but feel free to look for general information (e.g., "what does this menu do?", or go through the Beginner's Development Tutorial in Visual Studio).

3. One of the most finicky and system-dependent parts is *establishing a connection* between your application and your SQL server. We will provide sample

instructions for some systems, but it's up to you to find out how to establish a connection with your system (e.g. VB Express, JDBC.) Establishing a connection may be the part that takes the most time.

4. To connect to the database server cypress.csil.sfu.ca, you must connect from within the CSIL system. You cannot connect just over the internet from home. One option is to connect from a CSIL workstation. Another is to remotely connect to leto.csil.sfu.ca, and then connect to cypress.csil.sfu.ca from leto.csil.sfu.ca. This is the option shown in class.

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

Part I. Database Connection. 10 points.

Create a new application. When the application is run, the following should happen.

a) Calculate how many customers there are in the AdventureWorks database via a SQL query to the database.
b) Write to the screen how many customers there are.

Grading Criteria.

- Code + Connection: 40%.
- Query: 30%.
- Output: 30%.

Requirements

- Machine Environment: Your program should run on leto.csil.sfu.ca. Note: If you developed on your own system or via remote access, you may have to recompile your files on leto.
- Please include a readme file with instructions for running your code (e.g., open Visual Studio, open the project file "myproject", run it using Visual Studio…).
- Your application should connect to your own account on cypress.
- Your username and password (for the CSIL SQL server) should be in your code, so that the user doesn't have to enter them. To connect to the CSIL SQL server, currently Cypress, you will need to find out the exact hostname, your username on Cypress, and your password on Cypress. You can find this out using the CSIL instructions emailed earlier. One useful webpage is http://www.cs.sfu.ca/about/school-facilities/csil/windows/how-to-use-sql.html.
- Include supporting files if needed. (like any dll, jar or class files).

II. Stored Procedures/Functions. 10 points.

1. Write SQL code for a stored procedure (function) *AverageCost* that takes as input parameter a color and returns the average StandardCost of the products in the Product table that have that color. Execute the SQL code to create the stored procedure.
2. Create a new application. When the application is run, the application should write to the screen what the average StandardCost of *red* products in the Product table is. The answer should be computed by calling the stored procedure that you wrote for part II.1.

Grading Criteria.

- SQL code + creating stored procedure: 50%
- Application + output: 50%.

General Grading Criteria.

- 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.

- Your code should run in the CSIL environment so we can run it if necessary. However, if we have to check it out by running it, your documentation is probably insufficient.

Part III. Views. 25 points

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 color 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 <color> from the keyboard. This will be one of the colors.
    iii. Create a view table for that <color>, 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.

- SQL query 1, result: 20%.
- SQL code for view, result: 40%.
- Application with Interface: 40%.

Part IV. Cursors and HTML/XML. 25 points.

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 <color> 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 part II.
   Otherwise the application should 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.).

   You should use a cursor type of object for part 3 (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/XML 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.

- Input Form: 15%.
- Price Error Check: 15%.
- Sorted Output: 35%.
- HTML output: 35%.

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).

- 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 I: Database Connection. Submit the following files.

- Your source code sourceI.*, 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 sourceI.zip.
- A readme file with instructions for how to run your code on leto.csil.sfu.ca . Call this file readmeI.rtf .
- A screenshot of the output from running your application. Call this file outputI.pdf .

Put all these files together into a single archive called solutionI.zip .

Part II: Stored Procedures/Functions. Submit the following files.

- For the SQL code a single sql script. This should execute without error on SQL Server. Call this file sqlII.sql .
- Your source code sourceII.*, where * is the file extension required for your development setup. If your code comprises several files, please combine them into a single file sourceII.zip.
- A readme file with instructions for how to run your code on leto.csil.sfu.ca . Call this file readmeII.rtf .
- A screenshot of the output from running your application. Call this file outputII.pdf .

  Put all these files together into a single archive called solutionII.zip .

Part III: Views. Submit the following files.

For components 1 and 2:

- 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:

- 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.
- A readme file with instructions for how to run your code on leto.csil.sfu.ca . Call this file readmeIII.rtf .
- 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/XML. Submit the following files.

- 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.
- A screenshot of the session, where we can see the displayed colours, the user choosing a colour, and a price, and the html 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 = ‘green’, price is okay). Call this file outputIV.pdf .
- Also include the html file generated by your program, call it outputIV.html.

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