**ncsu_logo**

**BUS 442: Information Systems Development**

**Project 3: Creating an Application to Track and Manage Customer Data**

**Parts 2 and 3: Design and Build the Interface and Code Module**

Once your team has designed and built the customer database, Parts 2 and 3 of this project are to design and create the interface and VB.NET app code that uses the database.

**Design the App**

Think about all of the requirements needed for this app. Design the app ***before*** you begin coding it. Create a TOE chart as the primary design spec for this project, complete with all tasks, objects, and events. Use pseudo code for detailed logic, if preferred.

**Build the Prototype (optional)**

Your team may decide to first build a prototype for this app using arrays with data stored in memory. This prototype could then serve as the basis for defining the complete user requirements for this project. Alternatively, your team may decide to iterate the interface and database to firm up the functional requirements before building and deploying the app.

**Build the App**

Once your requirements are understood and your design is stable, create a solution with your *CustomerInformation* database as the data source. For grading purposes, you will actually store the database inside your VB project folder. Use the following steps to connect your database to your app and store the database in the project folder:

**Steps to Connect a Database to a VB Project:**

1. Open your new project in VB.
2. Click the Project menu then click Add New Data Source.
3. If necessary, click Database and Next.
4. If necessary, click Dataset and Next.
5. Click the New Connection button then click the Change button.
6. Select Microsoft Access Database File and choose OK.
7. Click the Browse button to find the database file. Select the database and open it.
8. (optional) click the Test Connection button to see if the connection succeeded. Click OK.
9. Choose the OK button back on the Add Connection dialog box. Choose the Next button.
10. Choose **Yes** to the question of whether you would like to copy the file to your project. Choose the Next button.
11. Click the + button to expand the tables.
12. Check the checkbox to the left of the tables you wish to select and click the Finish button.
13. Be sure to click the form tab to make your project form active then choose Show All Files from the Project menu.
14. If necessary, click View, Other Windows, Data Sources to view the Data Sources window.
15. Click the + button to expand the table. Consider changing your table view default. For example, select the **Details** View (not DataGrid) view. To do so, click the arrow to the right of the table name and choose Details.
16. Consider changing any of your table attribute controls to something other than the textbox default. For example, change the icon for country, state, and status to a combo box by clicking on the attribute itself, then clicking the combo box arrow to the right of each attribute name you want to change.
17. Click the icon to the left of the table name and drag it onto your form. (This will “bind” the table attributes to the newly-built controls on your form and will show the Binding Navigator bar at the top of your form.) This will give the user the ability to scroll backward or forward through the data and maintain existing entries, delete entries, or add new ones. By using the Details View, the data will be positioned *vertically* on the screen to allow room for a list box that will display customer sales data with the total and average of all customer sales listed below.
18. Open the Code Editor and type Option Infer ***On*** in the General Declarations section of the code to enable the use of LINQ in your app.
19. Run your project to ensure the data is appearing on your form.
20. Because you are including the actual database table data sets in the project, set the database’s **Copy to Output Directory property** to “**Copy if newer**.”

**Coding the App**

Once the database is connected to the project, you are ready to code the remaining parts of the module. Note that this app will not update data in the SALESREP table, only the CUSTOMER table. Here are some minimal requirements to consider when coding:

**Main Form Load subprocedure:**

* Review the code to load database data onto the interface form. This should already work without any necessary adjustments.
* Preload all combo boxes with valid data. Load the sales rep id for each sales rep from the SALES REP table and display it in the sales rep list (combo) box. You may also choose to display the name and region in a separate control on the form.

**Calculate Button subprocedure:**

When the ***Calculate*** button is clicked, read through the data set and display each customer in the **Average Sales** **list box** with the customer’s respective sales amount. Display the customer name then sales amount. When all records are listed, display the ***total*** sales amount and the ***average*** sales amount for all of the customers.

Here is one way to read through the data and calculate row by row using the For Each Next loop in a table named *CUSTOMER* using an attribute named *Sales*:

*Dim salesTotal as Double*

*For Each record As DataRow In Me.Customer\_DatabaseDataset.CUSTOMER*

*salesTotal += Double.Parse(record.Item(“Sales”))*

*…*

*Next*

Immediately after displaying the individual customer data, display the maximum sales and minimum sales for the whole group of customers using the Max and Min LINQ aggregate functions, respectively. Display these results in a message box or at the bottom of the list. (See the example on page 638.)

**Search Button:**

Provide the ability to search by customer name on the interface and just return those records to the interface (via a ***Search*** button). Use an InputBox to retrieve the customer name. Use the Try Catch method to provide a message if the customer name isn’t found. Use the example on page 635 to assist you, including the use of the Like operator and the wildcard character, “\*”.

**Refresh Button:**

Provide a ***Refresh*** button to return the form’s data back to all the original records after a search. Disable the Calculate button if the Search button is clicked. Enable it after the Refresh button is clicked.

**Menus and Tooltips:**

Add a **Queries menu** with three commands: CAN Customers, USA Customers, and NY Customers. Provide the logic behind these commands to execute and list on the form all customers by country or state, respectively. (See page 635 for examples of selecting data via LINQ).

Add an **Exit menu** to close the project, as well as an Exit button.

Add **tooltips** to further describe all major form controls.

**Navigator Bar and DropDown Button:**

You will use the Navigator Bar to add, delete, and update customer data in the CUSTOMER table. As stated earlier, you will not update data in the SALES REP table; instead, you will use it to *supply* the additional data on the company’s sales reps. However, be sure to display the correct sales rep name and region on your form if the user enters or maintains the sales rep number in the CUSTOMER table.

Add a **Sort By DropDown button** to the Binding Navigator Control to sort the customers on the form by name (ascending order), customer id, or sales (descending order). (See the programming example on pp. 635 for an example of sort data using the Order BY clause.)

**Technical Requirements**

* Use all of the form design and programming techniques we’ve learned this semester that are applicable to this project. This includes (but is not limited to):
  + Windows-standard interface adherence;
  + Error checking and validation messages;
  + Generous use of comments (both at top of module and throughout);
  + Option statements at top of module;
  + Exit Confirmation;
  + Update messages;
  + Ease of use when selecting or entering data;
  + Clearing of appropriate outputs when new inputs are keyed;
  + Correct formatting of data on the interface;
  + Use of functions, independent sub procedures, or arrays, if appropriate;
  + Use of string manipulation, if appropriate.
* Be sure to thoroughly test your solution and have at least one other person from another team test your work and suggest any needed changes. Create test cases beforehand (with all possibilities) and appropriate results. You may even want to designate one person on your team as the official tester.
* Allow either the lecturer or GA to see and approve your interface before you begin the programming.
* Once it is complete, upload it (one per 3-person team) to the Moodle link by the due date.
* Work hard and have fun!