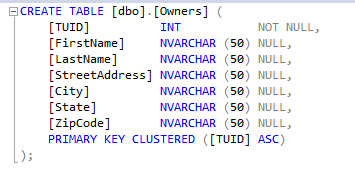
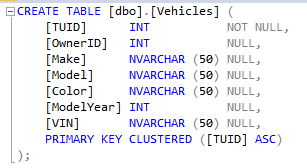
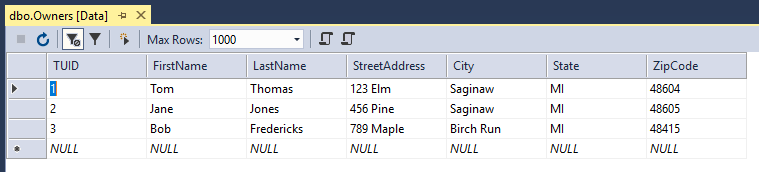
**CIS 311 Assignment 8**

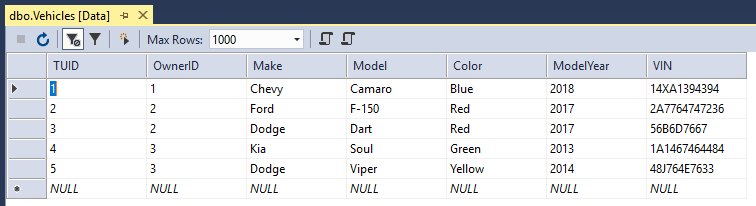
Databases! Begin by constructing a SQL Server database file with two tables in it. You can either do this through the Visual Studio IDE or natively through code. I called the database Autos (and the file was Autos.mdf, which I placed in the same folder as my .exe file – Environment.CurrentDirectory):

Watch the primary key fields – also notice the linkage between Vehicles and the Owners!

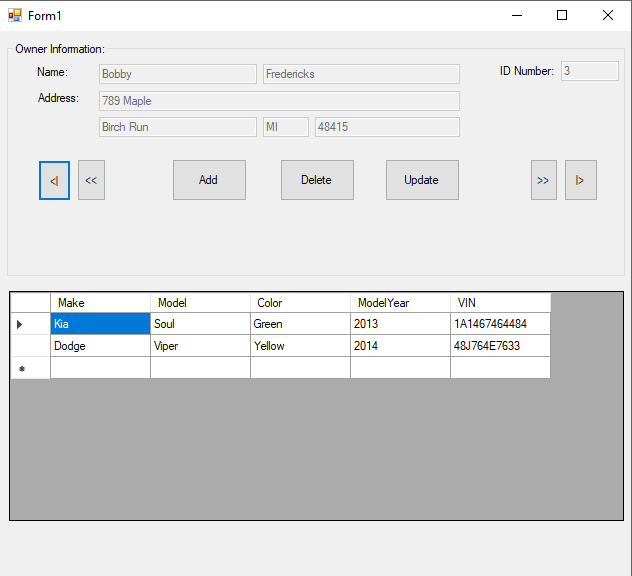
Add some base records there too (again either through code or through Visual Studio):



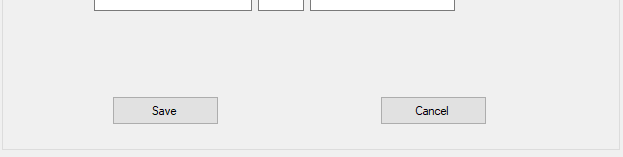


Now that you have a database, we are going to build an application that works with the data. Make sure that you place the database in the location specified above and that all I would need to do is either: (A) run your application and it creates everything or (B) if I need to reattach the database, I only have to browse to your executable’s folder and I find the database there. Let me know which way your application expects to be run at the top of your source code in the comments!!!!

There will be two halves to the application, which can be viewed in the main application form below:



The upper half of the screen shows all the related Owner Information from the database. All fields that are present are shown. There are navigation buttons (first, previous, next and last) that must be built, so a BindingContext makes the most sense to use here, hooked up behind the scenes to the textboxes. Finally, there are Add a record (allows you to enter a new owner record, mind the ID number uniqueness!), Delete a record and Update a record command buttons. Add will unlock the textboxes, take you to a blank record, and, once you are done entering the information, you will press one of the two buttons from the second panel which appears:



Save obviously saves the new record to the database, while Cancel would cancel the operation, throwing away the new record. The Update command works similar to Add in that the textboxes are unlocked, and the user can make whatever changes are desired and then presses the Save or Cancel button to commit or throw the changes away. The Delete command should prompt the user if he/she is sure the current record should be deleted. A “Yes” toasts the record from the database, a “No” leaves it intact.

Finally, as the owner record is navigated, all vehicles that are registered to an owner are shown in the DataGridView at the bottom. Notice that not all fields from that table are displayed. One of the biggest challenges will be to keep the DataGridView in sync with the owner information. Remember an Add operation goes to a new record, so the DataGridView should be blank. Upon a Cancel though, you need to make the DataGridView show the vehicle details for whatever owner record is in context. Ditto for the update and delete operations! Make sure that you thoroughly test your application for all possibilities. You do not have to allow any additions/changes/deletions to the vehicle information in the DataGridView (unless you want to…)!

Build this application up in pieces to avoid extra headaches. Get the BindingContext working for the textboxes and enable navigation. Then pick off the Add, Delete and Update commands one at a time until they work. Once you have reached that point, you can start thinking about the DataGridView and synchronizing its display to the owner information record being shown.

If you have any questions about the SQL (it’s pretty simple for this application), let me know and I’ll walk you through it.

Complete your assignment and place your entire solution in a zip file, which you will upload to Canvas. Turn in a cover sheet, your program source code and screenshots of your program’s execution stapled together in that order in class.