Project #1

Complete By: Monday, June 25th @ 11:59pm
Assignment: completion of following programming exercise
Policy: Individual work only, late work *is* accepted (see "Policy" section at end of doc for more details)
Submission: electronic submission of VS project folder to BB

Programming Exercise: Build and populate a BikeHike Database

As discussed in class, the goal in this homework is to bring your **BikeHike** database design to life. There are 3 main steps in this project:

1. Work with Visual Studio and a query window to write the SQL DDL to create your tables.
2. Download and modify the provided C# console app to execute your DDL and build a BikeHike database.
3. Parse the provided data files and insert this data into your database.

When you are done, you should have a complete C# console app that when run, produces a populated BikeHike database of your own design.

Step 1: write SQL DDL

Based on your solution to HW #1, and the subsequent discussion in class, write the SQL DDL to create a BikeHike database. Work with Visual Studio as discussed in class to test your DDL; an empty database file is available on the course web page under Lectures, Day 2. If you missed class, the lecture notes are available on the course web page, and lecture recordings are available on Blackboard.

For simplicity, redefine your **Customer** entity to contain just a customer ID, first name, last name, and email address. The **BikeType** and **Bike** entities remain the same as in HW #1. Your design should support the same scenarios as discussed in HW #1. Modify your DDL so that bike types start with an ID of 1, bikes start with an ID of 1001, and Customers start with an ID of 20001 (yes, that's 20,001).

Save your SQL DDL queries (i.e. save the query window) in the file "BikeHike.sql". Note that since this DDL is going to be executed against an empty database each time, it does not need to start with "Drop Table" queries. In fact, those queries will cause errors when executed against an empty database, so delete or comment out.

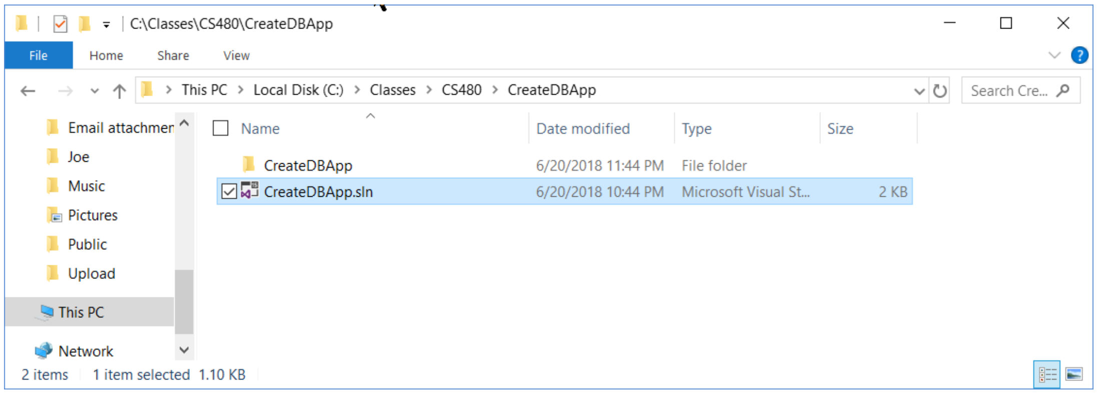
Step 2: programmatically execute SQL DDL using C#

The next step is to create a BikeHike database starting from an empty MDF file and then executing your DDL against this file. To automate this, we're going to use a console app written in C# to programmatically copy the empty database files and execute your DDL. The app has been written for you, so start by downloading the file "CreateDBApp.zip" from the course web page --- use the "download" option in upper-right corner:

https://www.dropbox.com/s/5lz46w5z3nj9qkj/CreateDBApp.zip?dl=0

Double-click to open the .zip, and *EXTRACT* the folder "CreateDBApp" to your desktop. Delete the compressed (.zip) file. [*NOTE: I would leave the "CreateDBApp" folder on your desktop --- when you want to backup, copy this folder to your backup device (e.g. I copy to my dropbox folder). I do *not* recommend working on this folder directly from a cloud storage account (e.g dropbox or onedrive) --- the cloud backup mechanism often interferes with Visual Studio's background compilation, causing file system errors.*]

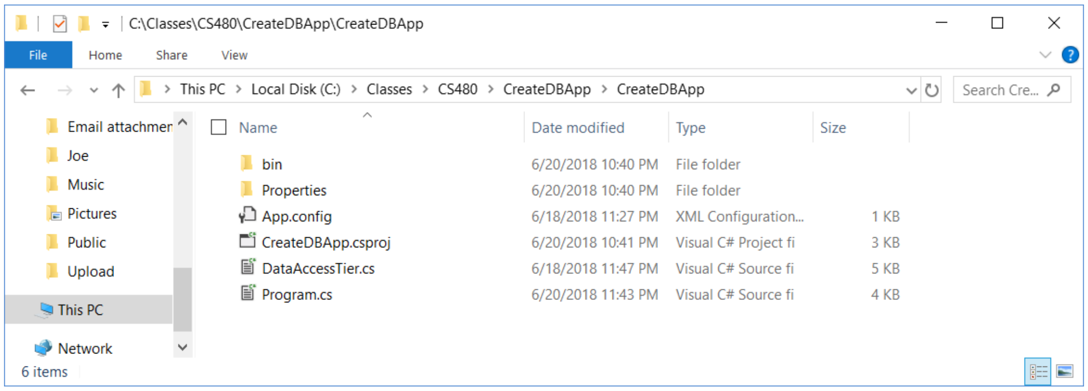
Find the CreateDBApp folder that you extracted. Double-click to open, and you'll see the Visual Studio Solution File (.sln extension if you have file extensions visible):



Name	Date modified	Type	Size
CreateDBApp	6/20/2018 11:44 PM	File folder	
CreateDBApp.sln	6/20/2018 10:44 PM	Microsoft Visual St...	2 KB

2 items 1 item selected 1.10 KB

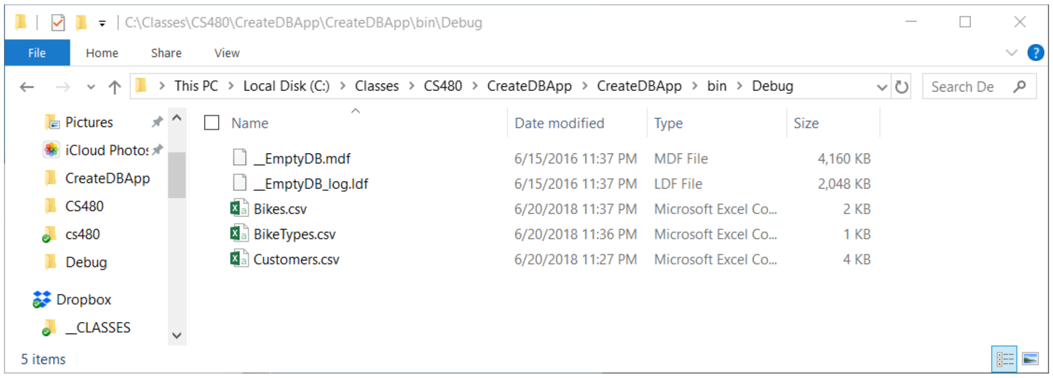
This Solution File (.sln) is what you double-click to open the console app in Visual Studio --- ignore for the moment... Instead drill down another level into the CreateDBApp sub-folder:


C:\Classes\CS480\CreateDBApp\CreateDBApp
File
Home
Share
View
This PC > Local Disk (C:) > Classes > CS480 > CreateDBApp > CreateDBApp
Search Cre...
Email attachmen
Joe
Music
Pictures
Public
Upload
This PC
Network

Name	Date modified	Type	Size
bin	6/20/2018 10:40 PM	File folder	
Properties	6/20/2018 10:40 PM	File folder	
App.config	6/18/2018 11:27 PM	XML Configuration...	1 KB
CreateDBApp.csproj	6/20/2018 10:41 PM	Visual C# Project fi	3 KB
DataAccessTier.cs	6/18/2018 11:47 PM	Visual C# Source fi	5 KB
Program.cs	6/20/2018 11:43 PM	Visual C# Source fi	4 KB

6 items

Here you see the C# source code (.cs) files. Continue to drill-down further into the bin\Debug folder:

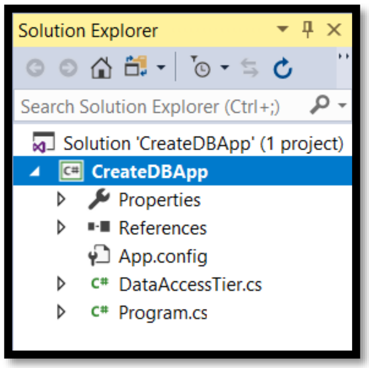

C:\Classes\CS480\CreateDBApp\CreateDBApp\bin\Debug
File
Home
Share
View
This PC > Local Disk (C:) > Classes > CS480 > CreateDBApp > CreateDBApp > bin > Debug
Search De
Pictures
iCloud Photos
CreateDBApp
CS480
cs480
Debug
Dropbox
_CLASSES
5 items

Name	Date modified	Type	Size
_EmptyDB.mdf	6/15/2016 11:37 PM	MDF File	4,160 KB
_EmptyDB_log.ldf	6/15/2016 11:37 PM	LDF File	2,048 KB
Bikes.csv	6/20/2018 11:37 PM	Microsoft Excel Co...	2 KB
BikeTypes.csv	6/20/2018 11:36 PM	Microsoft Excel Co...	1 KB
Customers.csv	6/20/2018 11:27 PM	Microsoft Excel Co...	4 KB

This sub-folder --- **bin\Debug** --- is where Visual Studio looks for files when you run. Notice the empty database files are here, as well as the input data files:

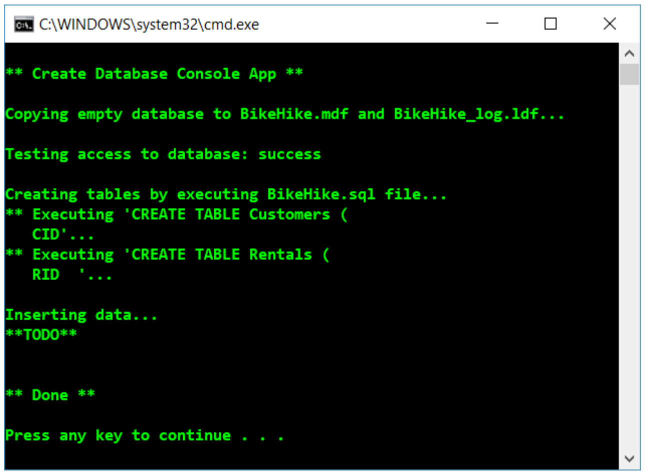
Bikes.csv
BikeTypes.csv
Customers.csv

Place a copy of your DDL file here, which must be called "BikeHike.sql". Ultimately, this is also the folder where you'll find your final "BikeHike.mdf" and "BikeHike_log.ldf" database files.



Go back to the top-level folder that contains the Solution File (.sln), and double-click to open the program in Visual Studio. After a few seconds you should see the **Solution Explorer** window in the top-right corner of VS --- if not, use the View menu to open this window ============================> Double-click to on "Program.cs" to view the main program in the editor. Browse through the source code for Main().

When you are ready, go ahead and run the program: Start menu, run without debugging (Ctrl+F5). This will compile and run the program, leaving the console window open so you can view the results. If your "BikeHike.sql" file works and resides in the bin\Debug sub-folder, you should see something like this:



```
** Create Database Console App **

Copying empty database to BikeHike.mdf and BikeHike_log.ldf...

Testing access to database: success

Creating tables by executing BikeHike.sql file...
** Executing 'CREATE TABLE Customers (
   CID'...
** Executing 'CREATE TABLE Rentals (
   RID  '...

Inserting data...
**TODO**


** Done **

Press any key to continue . . .
```

If your DDL is correct, you should see more "Executing…" statements, one for each DLL command in your .sql file. And if you minimize Visual Studio and look in the bin\Debug sub-folder, you should see the database files for your BikeHike database. Finally, you should be able to use Visual Studio's Server Explorer to connect to your BikeHike database in bin\Debug, and confirm that the tables were created. [***Note***: *if you connect using Server Explorer, don't forget to close the connection before you attempt to run the console app again. Otherwise the database files are locked and the console app will fail.*]

Step 3: programmatically parse and insert data to populate your database

The provide app contains 3 input files in CSV format: Bikes.csv, BikeTypes.csv, and Customers.csv. Here are the first few lines of each file:

Bikes.csv:
1001,1,2016
1002,1,2016
1003,1,2016
1004,1,2017
1005,1,2017

BikeTypes.csv:
1,Tandem For Two,20.00
2,Single Cruiser,10.00
3,Single Electric,30.00
4,Recumbent,50.25
5,Single 21-speed,12.50

Customers.csv:
20001,Booma,Balasubramani,bbalas3@uic.edu
20002,Joe,Hummel,jhummel2@uic.edu
20003,Mohit,Ghia,mghia2@uic.edu
20004,Tejas,Sarma,tsarma2@uic.edu
20005,James,Hwang,jhwang47@uic.edu

Assume the files are in ascending order by their ID field, so that the data matches the IDs auto-generated by the DBMS according to your DDL.

Since many of you are new to C#, here's some help parsing the files… When parsing in C#, the simplest approach is to read the file line by line. For each line, parse the data by splitting the line based on the "," separating each value. Once parsed, build an SQL **insert** query and execute. Let's assume we are parsing the bike types data: type id, description, and price per hour. Here's some code to open, loop through the file, and parse each line:

```
//
// using stmt will close file when scope is exited:
//
using (var file = new System.IO.StreamReader("biketypes.csv"))
{
  while (!file.EndOfStream)
  {
        string line = file.ReadLine();

        string[] values = line.Split(',');

        int    typeid       = Convert.ToInt32(values[0]);
        string description  = values[1];
        double priceperhour = Convert.ToDouble(values[2]);

        << build SQL insert query >>

        << execute SQL query >>

      }//while
    }//using
```

To build the insert query, the best approach is to use **string.Format**, which is much more efficient (and readable) than string concatenation:

```
    string sql = string.Format(@"
    Insert Into
      BikeTypes(TID,Description,PricePerHour)
      Values({0},'{1}',{2});
    ",
    typeid, description, priceperhour);
```

The C# **string.Format** function is like printf, building a formatted string from given arguments. In the format string, the {0} means insert the 1st argument into the result string, {1} means insert the 2nd argument into the result string, and so on. Notice that {1} is surrounded by single quotes ' since the Description attribute in the BikeTypes table is a string value.

To execute the query, use the provided Data Access Tier; this has already been instantiated as **data** by the Main() program. In short, for every SQL insert query you want to execute, call

```
data.ExecuteActionQuery(sql);
```

This will throw an exception if it fails; if you get an exception, the first step towards debugging is to output the SQL query so you can check to see if it's properly formatted:

```
Console.WriteLine(sql);
```

Have a question? Use Piazza, not email

Use Piazza for questions: http://piazza.com/uic/summer2018/cs480/home. As discussed in the syllabus, questions via email will be ignored, so post to our Piazza site. Remember the guidelines for using Piazza:

1. *Look before you post* — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post a question. Posts are categorized to help you search, e.g. "Pre-class" or "HW".
2. Post publicly — only post privately when asked by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means…".
4. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, so don't hesitate to take a screenshot and post; see http://www.take-a-screenshot.org/ .
5. Don't post your entire answer — if you do, you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question --- in that case post only the fragment that denotes your question, and omit whatever details you can. If you must post the entire code, then do so privately --- there's an option to create a private post ("visible to staff only").

Electronic Submission

First, modify the header comment at the top of your C# source code file ("Program.cs") so that it includes your name:

```
//
// Console app to create a database, e.g. BikeHike.
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS480, Summer 2018
// Project #1
//
```

To submit, exit out of Visual Studio, and find your project folder --- this should be the **top-level** folder "CreateDBApp". Create an archive (.zip) of this entire folder: right-click, Send To, and select "Compressed (zipped) folder". Then submit the resulting "CreateDBApp.zip" file on Blackboard: open "Projects" and submit under "Project 01: Create BikeHike DB".

Your program should be readable with proper indentation and naming conventions; commenting is expected where appropriate. You may submit as many times as you want before the due date, but we grade the last version submitted. This implies that if you submit a version before the due date and then another after the due date, we will grade the version submitted after the due date — we will *not* grade both and then give you the better grade. We grade the last one submitted. In general, do not submit after the due date unless you had a non-working program before the due date.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is described here:

https://dos.uic.edu/docs/Student%20Disciplinary%20Policy%2017-18%20(FINAL).pdf

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml.