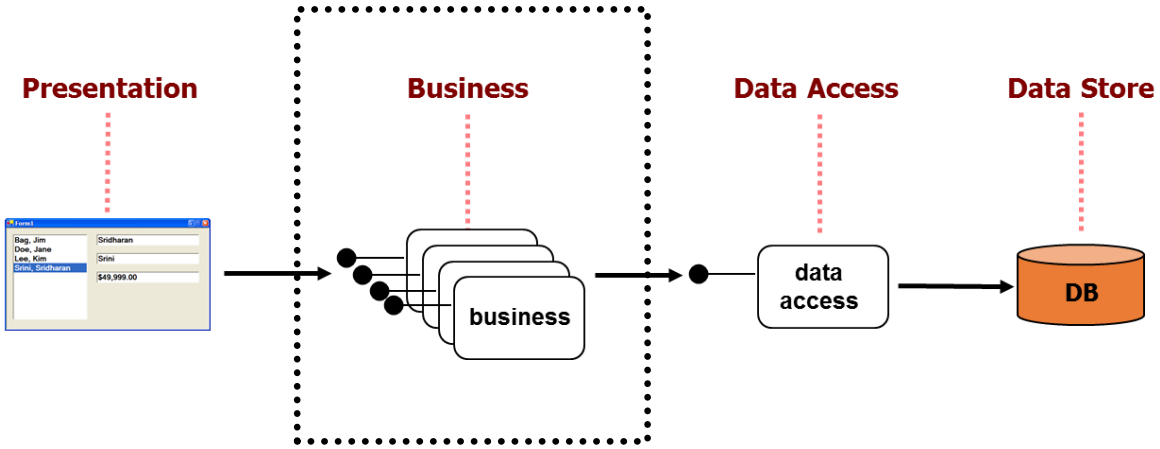
The Final Project: Netflix Database App, Part 2

Complete By: Sunday, May 6th @ 11:59pm
Assignment: completion of N-tier Netflix Database App
Policy: Individual work only, late work *is* accepted
Submission: electronic submission via Blackboard

Assignment

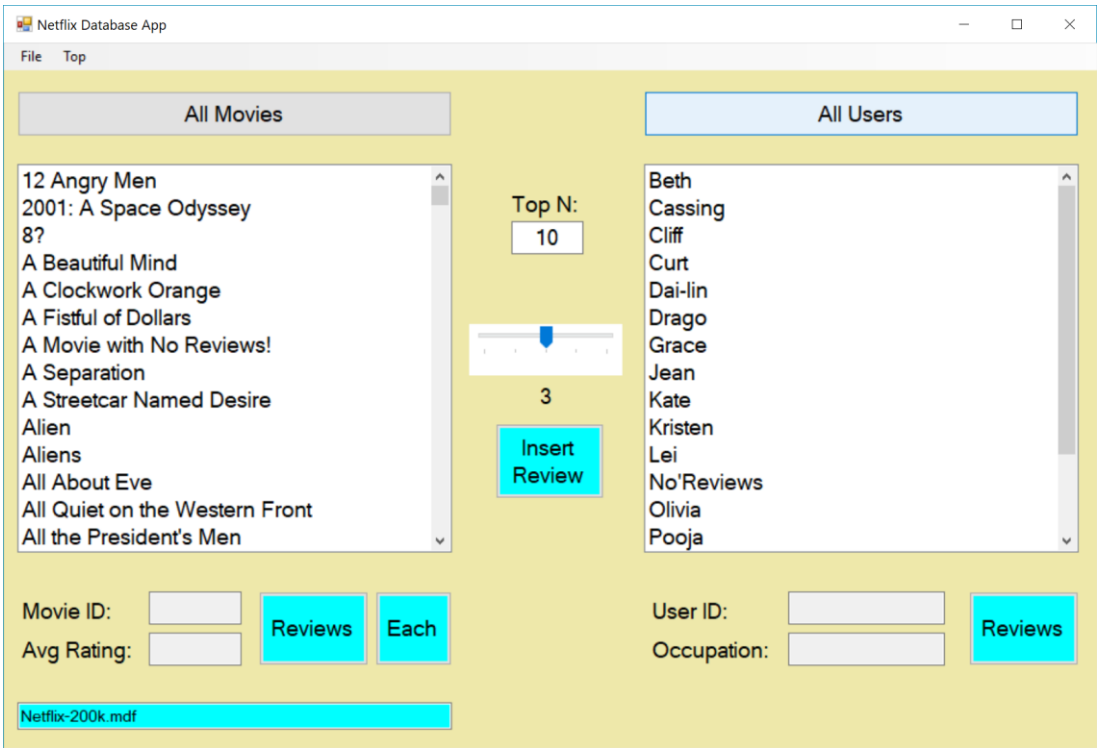
In the previous project you built the GUI for an N-tier database app focused on Netflix movie data. In this assignment you're going to focus on the Business tier, implementing the functions you relied upon in the previous project.



Step 1: Your solution or ours?

The first step is to decide whether to keep your GUI, or use our provided GUI. You are encouraged to use your own GUI, since (a) you know it better, and (b) the final program will end up being completely your own work. But feel free to use our GUI if you were unable to complete the previous project, or you ended up with a GUI that you aren't happy with. Once the deadline for project #08 has passed, you can download a copy of our GUI from the **FinalProject** folder on the course web page. In particular, download and unzip **NetflixApp.zip**.

A screenshot of our GUI is shown on the next page.



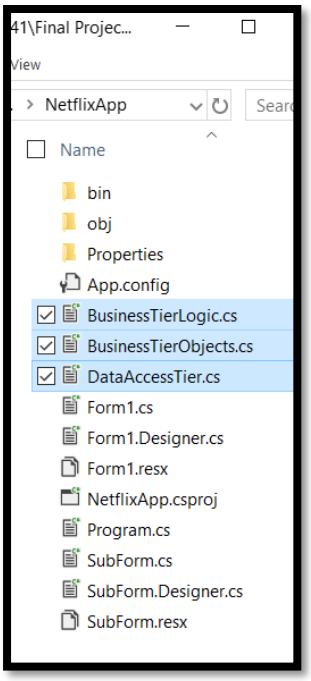
Step 2: Getting Setup

At this point you should have a working solution to the previous project #08 --- if you have decided to use our GUI, make sure you have installed copies of the Netflix database files in the **bin\Debug** project sub-folder.

1. Download the C# implementation files for the Business and Data Access tier. These can be found in the **FinalProject** folder. Best to use the "download" button in dropbox's upper-right corner:

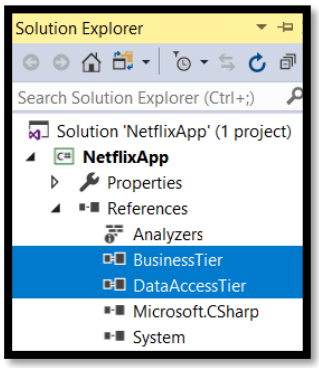
   a. BusinessTierLogic.cs
   b. BusinessTierObjects.cs
   c. DataAccessTier.cs

2. Move these files into your same folder as your other C# (.cs) files, such as Form1.cs. This should be the sub-folder **NetflixApp\NetflixApp**.



3. Open your Netflix app in Visual Studio, and run --- confirm it works properly.

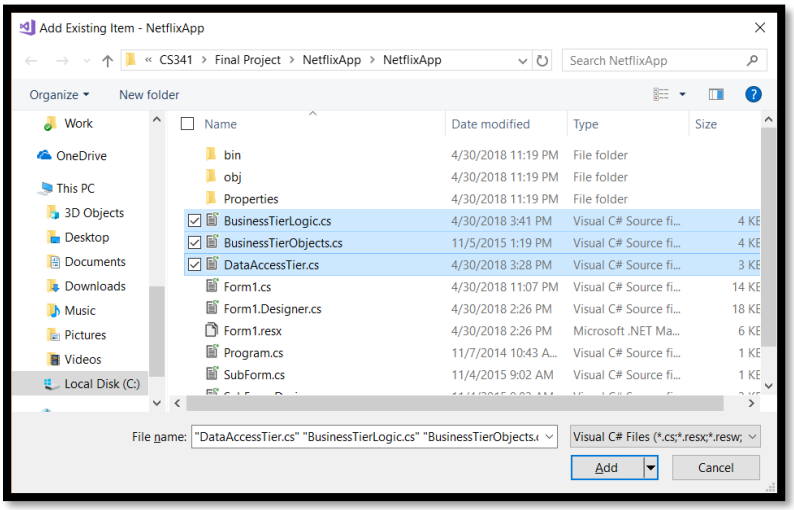
4. Delete the references to the DLLs. View menu, Solution Explorer, expand References, select BusinessTier and DataAccessTier, and **delete** the references (Delete key works).



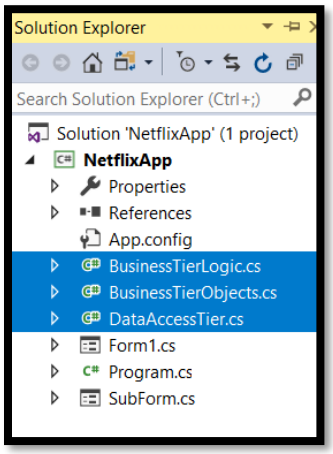
5. Minimize Visual Studio, and delete the DLLs from the bin\Debug and bin\Release sub-folders.

6. Back in Visual Studio, Rebuild the program via the Build menu --- it should fail because the DLLs are now gone.

7. Add the C# files we downloaded earlier --- Project menu, Add Existing Item, and select the 3 files you downloaded earlier. This will add the source code to the Business and Data Access Tiers to your project:



8. You should now see the Business and Data Access Tier source code files listed in your Solution Explorer window.



9. Rebuild your program via the Build menu, and the program should compile successfully. If so, skip ahead to step 10…

   If your program failed to compile, then you are probably using one or more of the Business Tier functions that we deleted because they technically weren't required for the assignment. For example, perhaps you were calling the GetUser( ) function instead of the GetNamedUser () function? If so, you'll need to add back any missing functions to the BusinessTierLogic.cs source file. Here's the function definition for GetUser( ) if you need that functionality back. For other functions, please post to Piazza and we'll provide function definitions if you need them:

```
//
// GetUser:
//
// Retrieves User object based on USER ID; returns null if user is not
// found.
//
// NOTE: if the user exists in the Users table, then a meaningful name and
// occupation are returned in the User object.  If the user does not exist
// in the Users table, then the user id has to be looked up in the Reviews
// table to see if he/she has submitted 1 or more reviews as an "anonymous"
// user.  If the id is found in the Reviews table, then the user is an
// "anonymous" user, so a User object with name = "<UserID>" and no occupation
// ("") is returned.  In other words, name = the user's id surrounded by < >.
//
public User GetUser(int UserID)
{
  //
  // TODO!
  //

  return null;
}
```

10. **Run your program.** You should be able to display all the movies (since that is done using SQL), but nothing else should work because all the Business Tier functions are empty. If you handled getting null back, your program should fail to display any additional data --- if you didn't handle getting null back from the Business Tier, then your program may crash (which is fine).

11. **The next step?** To implement the functions in the Business Tier, one by one…

Step 3: Implementing the Business Tier

Earlier you downloaded and installed three C# source files:

1. BusinessTierLogic.cs
2. BusinessTierObjects.cs
3. DataAccessTier.cs

Files #2 and #3 are already implemented for you --- your job is to implement each of the functions in file #1, **BusinessTierLogic.cs**. These are the functions you were calling in the previous project to retrieve the data. Now you have to implement each of these functions to create the appropriate SQL, execute it using the Data Access Tier, and then return back the data in one or more objects. In short, you are now performing the **Object-Relational Mapping** (ORM).

Open the BusinessTierLogic.cs file. Start with the easiest Business Tier function: **GetMovie()**. There are two versions, one that takes a MovieID and one that takes a MovieName. Here are the function definitions from the C# source file:

```
//
// GetMovie:
//
// Retrieves Movie object based on MOVIE ID; returns null if movie is not
// found.
//
public Movie GetMovie(int MovieID)
{
  //
  // TODO!
  //

  return null;
}

//
// GetMovie:
//
// Retrieves Movie object based on MOVIE NAME; returns null if movie is not
// found.
//
public Movie GetMovie(string MovieName)
{
  //
  // TODO!
  //

  return null;
}
```

To implement these functions, do the following:

1. Notice the Business class contains an instance of the Data Access Tier named **dataTier**. This is created in the constructor, so it's available for use by all the Business Tier methods. Use this object to access the Data Access Tier.

2. Write an SQL string to retrieve the data for the given movie; search based on MovieID or MovieName. Keep in mind it is the responsibility of the Business Tier to escape string-based values that include '.
3. Execute the SQL string using the Data Access Tier. Pick the correct function based on whether you are retrieving a single value, or multiple values.
4. If the SQL fails to find the movie, return null.
5. Otherwise create a new **Movie** object, passing the MovieID and MovieName to the constructor. Return this object as the result of the function call:

```
Movie m = new Movie( MovieID, MovieName );
return m;
```

6. Run and test --- be sure to try various movie ids / names, including one that contains a '.

This is the basic strategy of how all the Business Tier functions work. The more complicated functions may require the execution of multiple SQL queries, and may return a list of objects. A good example is **GetMovieDetail( )**, which returns a **MovieDetail** object that contains the average rating and a list of the reviews. To create the list of reviews, create a **List<Review>** object, retrieve the reviews using SQL, and then create and add individual **Review** objects to the list.

You will run into a complication when you implement the **AddReview** function. This function is supposed to return a Review object that contains the review id — what's the proper way to retrieve this id since it's automatically generated by the database? In SQL Server, the proper technique is to execute a scalar query (*not* an action query) containing two SQL commands in one string: one to do the insert, and the second to select and return the new ID based on a call to the internal function SCOPE_IDENTITY() --- which returns the primary key of the last row inserted / updated / deleted. For example, suppose you are inserting a new movie name:

```
string sql = string.Format(@"
INSERT INTO Movies(MovieName) VALUES('{0}');
SELECT MovieID FROM Movies WHERE MovieID = SCOPE_IDENTITY();
", moviename);
```

Execute this two-step query using a call to **ExecuteScalarQuery**, and the result returned will be the new movie id (or null if the insert failed). Use a similar approach to insert a new review.

Step 4: Extend the Business Tier…

The goal is to remove all SQL form your GUI code. For sure, this means adding a **GetAllMovies( )** function to the Business Tier, which returns a read-only list of Movie objects. Add this function to BusinessTierLogic.cs, and then rewrite the GUI to call this function instead of using SQL. Run and test --- your program should behave exactly as before.

Electronic Submission

First, add a header comment to the top of your "BusinessTierLogic.cs" C# source code file, along the lines of:

```
//
// BusinessTier:  business logic, acting as interface between UI and data store.
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS341, Spring 2018
// Final Project
//
```

Exit out of Visual Studio, and find your **NetflixApp** project folder. Drill down to the "bin\Debug" sub-folder, and delete the database files to save space. Then create an archive (.zip) of this entire folder for submission: right-click, Send To, and select "Compressed (zipped) folder". Submit the resulting .zip file on Blackboard (http://uic.blackboard.com/) under the assignment "FinalProject: Netflix Database App, Part 2".

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:

http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.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 .