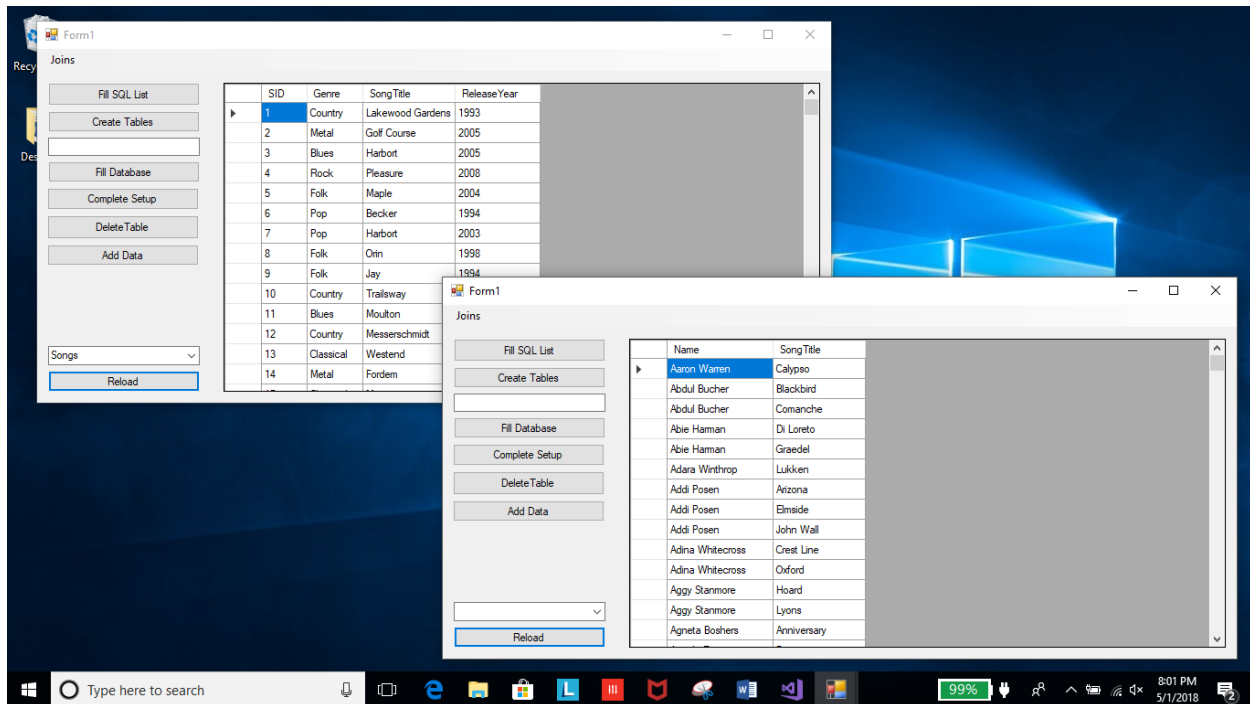


Student: David Kay : dkay5@uic.edu
Professor: Joe Hummel : jhummel2@uic.edu

Report for database project designed and built for CS 398 Professional development class.

Summary:

Built a multi-user database application with transactions and rollback capability.



Report:

DESIGN

After a quick review of the material covered in my previously taken 341 class. I started by using the given crime database to create a empty SQL database template from which to work. Then a rough design of a database was drawn up using music as the theme. The design started with a table for songs and one for artists and a linking table to correlate the two. As well as a table for 1-5 reviews of songs from the songs table.

SONGS			
ID	Genre	Song Title	Release Date
3434	Pop	Gift of Song	1979
4445	blah	xxx	1288
Artists			
ID	Name	YOB	Gender
6666	Earth Wind Fire	1969	N
8976	Rod Stewart	1945	M
7767	Donna Summers	1948	F
Reviews			
ID	Song ID	Review	
1111	3434		5
Artists Songs			
Artist ID	Song ID		
	4545		3434
	4545		4445

After some discussion the design was revised and updated to remove redundancy and add some connections. Three new tables were added to expand the database and clear up some confusion.

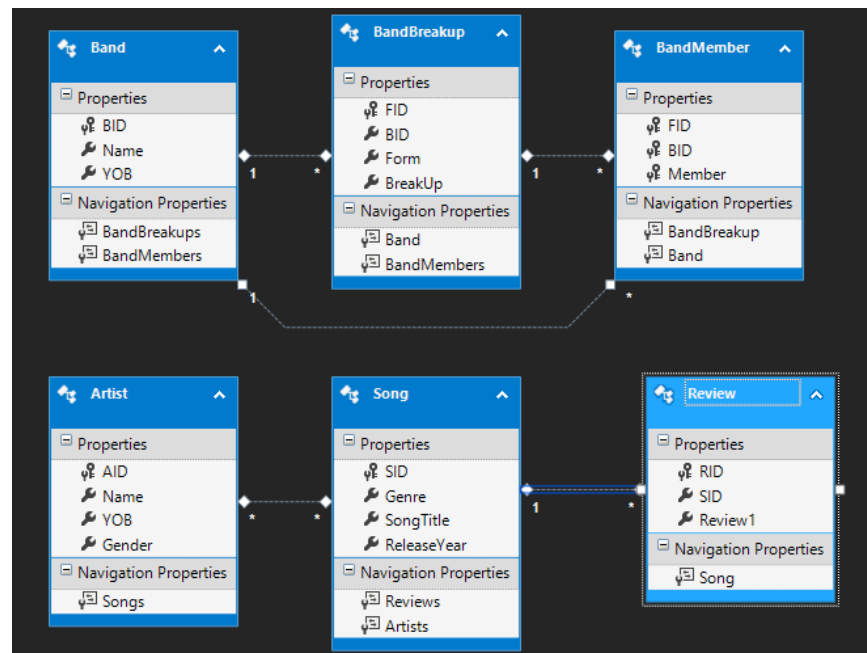
Bands		
ID	Name	YOB
4545	ABBA	1972
6666	Earth Wind Fire	1969

Band Breaks				
Formation ID	Band ID	Begin	End	
2323	4545	1990	1995	
7654	4545	2001	2004	

Band Artists	
formation ID	person
7654	smith
7654	john

DATABASE CREATION

Once there were some designs and goals setup it was much simpler to form the SQL commands to create the tables. The commands for the table creation were created and compiled into a List structure to be looped thru and sent successively to the empty database. Initially all the tables were created to have Identity Primary keys this was later changed for most tables as it created problems with data entry.



When coding the data entry for the application I originally started to make a different entry setup for each table in the database. However after repeating a significant amount of the code for the

second time I decided to try to condense this in to one reusable section of code. This turned out harder to do than I thought.

In order to fill the database the user gives the application the name of a txt file formatted with the name of the table the data should be added to and the names of the columns corresponding to the data below them. For example the Songs data file would look something like this...

```
Songs,SID,Genre,SongTitle,ReleaseYear
1,Country,Red Pickup,1993
2,Metal,Angel of Death,2005
3,Blues,Rainy Day,2009
...
```

The application then reads the file line by line into a string array or returns a error message if the file can not be found. This gives you all the information needed for an INSERT INTO SQL command.

```
INSERT INTO
Table_name(column1,column2,column3,column4)
Values(value1,value2,value3,value4)
```

For the above example the *Table_name* and the columns are given at the beginning of the file and the values are in each following row. Once given a string from the string array it is very simple to split it into its components and format those components. The difficult part comes in when entering different amounts of data from different tables to use the same command. This is done by saving the components of data to otherwise null variables which are then used in a variable SQL command so that any data for any of the tables can be added using this same code.

```
cmd.CommandText =
String.Format(@"
INSERT INTO
{0}({1}{2}{3}{4})
Values({5}{6}{7}{8});
", table, col1, col2, col3, col4, val1, val2, val3, val4);
```

This works because any data lacking would be filled with a null value which is ignored by the database and so any data for any of the tables can be added using the same chunk of code.

This was thoroughly tested thru the use of a significant amount of test data which was generated and formatted into txt files with the use of a data generation website called Mockaroo [<https://www.mockaroo.com/>].

DATABASE USE

Once the application is running and the database setup is done the database can be worked with/ on from the application. All the data in the database can be displayed. Displaying the data in the database was done using a Sql Data Adapter and a Data Grid View.

```
private void GetData(string command)
{
    string version = "MSSQLLocalDB";
    string filename = "MusicDB.mdf";

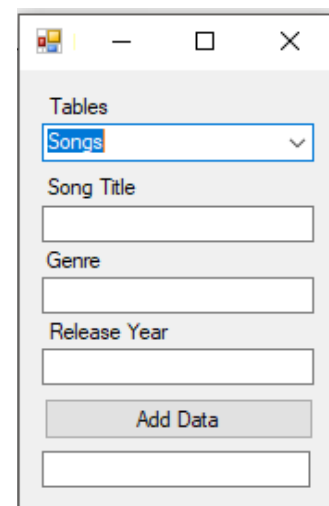
    string connectionInfo = String.Format(@"Data Source=(LocalDB
\{0};AttachDbFilename=|DataDirectory|\{1};Integrated Security=True;", version, filename);

    DA = new SqlDataAdapter(command, connectionInfo);
    SqlCommandBuilder builder = new SqlCommandBuilder(DA);
    DataTable table = new DataTable();
    table.Locale = System.Globalization.CultureInfo.InvariantCulture;
    DA.Fill(table);
    mainbindingSource.DataSource = table;
    mainDataGridView.AutoSizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);
}
```

A variable combo box can be used to display any of the tables while a File menu has several common joins.

```
if (this.comboBox1.SelectedIndex == 0)
{
    mainDataGridView.DataSource = mainbindingSource;
    GetData("select * from Songs");
}
etc...
```

Data can be added as a individual entry by the user to the database from the application using a similar, in terms of code at least, setup to the file-wise data entry. Clicking Add Data button opens a new Form window. Any of the tables can be selected from the combo box and the relevant data can be entered into the variable text boxes below. Underneath the Add Data button in this Form is another text box which is removable but added in order to demonstrate/simulate the multi-user aspect of the application.



The screenshot shows a standard Windows application window with a title bar containing the Windows logo, minimize, maximize, and close buttons. The window's title is "Tables". Inside the window, there is a dropdown menu at the top with the word "Songs" selected and a downward arrow. Below the dropdown are three text input fields, each with a label to its left: "Song Title", "Genre", and "Release Year". At the bottom of the window is a button labeled "Add Data". Below the "Add Data" button is another empty text input field.

In order to add new data like this, the new entry is added to the end of the respective table. However this requires the application to look up the max id in the given table which is an opportunity for conflict in a multi-user setup. Conflict could occur when multiple users request the max id from the database at the same time and so get the same result and then try to add a new entry to the table with the same id. Running multiple instances of the application and utilizing the extra text box to add a time delay simulates the results of two users having this conflict.

This problem was solved by adding a Transaction to enclose the code involved in getting the max id and adding a new entry to the table with an id of max plus one. Thus preventing the system from making multiple requests for the max at the same time.

```
try
{
    db = new SqlConnection(connectionInfo);
    db.Open();
    Tx = db.BeginTransaction(IsolationLevel.Serializable);
    ...
    if (this.comboBox1.SelectedIndex == 0)
    {
        cmd = new SqlCommand();
        cmd.Connection = db;
        cmd.Transaction = Tx;
        cmd.CommandText =
            String.Format(@"
SELECT MAX(SID)
FROM Songs;
");
        object result = cmd.ExecuteScalar();
        int max = (int)result + 1;
        ...
        cmd.CommandText =
            String.Format(@"
INSERT INTO
{0}({1}{2}{3}{4})
Values({5}{6}{7}{8});
", table, col1, col2, col3, col4, val1, val2, val3, val4);
        cmd.ExecuteNonQuery();
        Tx.Commit();
    }
    ...
}
```

Having a try/catch/finally transaction setup prevented multiple users from adding an entry with the same id however this alone caused the application to rollback one of the requests and only add one entry to the database. This problem was resolved by detecting if there was a rollback in the catch code and repeating the request that was rolledback till accepted or the same request had three failed attempts.

```
int count = 0;
while (count <= 3)
{
    try
    {
        ...

    catch (Exception ex)
    {
        if (Tx != null)
            Tx.Rollback();
        count++;
        MessageBox.Show(ex.Message);
    }
    finally
    {
        if (db != null)
            db.Close();
        count = 3;
        this.Close();
    }
}
}
```

Conclusion:

I expanded on my database knowledge from my CS 341 class. Learning most importantly about a multi user design the ways it differs from a single user application and the steps needed to be taken to ensure smooth interaction with a database. I learned about the importance of Transactions and uses in making the interactions smooth and handled correctly. As well as how to rollback a failed interaction so as to keep the system running and the common practices of cleaning up after a rollback so the user notices little to nothing wrong when possible. As a result of this class I also expanded my knowledge and experience working with visual studio and basic database interactions along with a uniquely interesting experience condensing code.