Database Table Editor for Excel

by Brent Larsen

Executive Summary

This project is a database table editor that is geared toward those who use databases heavily, and in particular those who frequently insert, update, or delete from tables. The business purpose grew from needs that I had as an Implementation Consultant for government Enterprise Resource Planning (ERP) systems. A large part of my job was to configure database tables that served as references for how the rest of the system operated. As more groups within the government agencies were brought into the system and the level of maintenance increased, we frequently copied existing configuration and made modifications as necessary to accommodate the new groups. The modifications were frequently done in Excel due to its flexibility in writing formulas to generate the changes that were needed.

Our system database for several years has been SQL Server 2008, and we frequently copied and pasted directly from Excel into SQL Server. I began to run into limits with this method when dealing with several thousand inserts, where copying nearly 13,000 rows took approximately 15 minutes to accomplish. In order to deal with this I would write Excel formulas to generate insert scripts which could be run in SQL Server Management Studio in less than 20 seconds. While this was a huge increase in performance, the downside was the need for me to write a new formula for generating scripts for every table. Based on the bottlenecks described, this project is meant to simplify two processes:

1. Copying data from Excel to the database.
2. Generating insert scripts for running directly in a database GUI.

Documentation

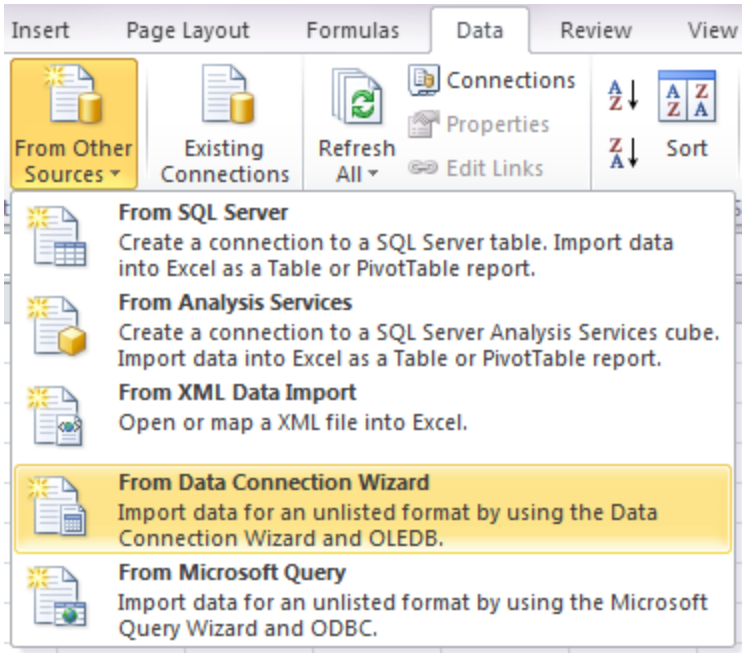


Figure 1

The first step in using the editor is to create a connection to a database table using Excel's built-in Data Connection Wizard (see Figure 1 at left). Alternatively, existing .odc (Office Data Connection) files can be loaded into the workbook as explained in a later section. The original scope of the project included a screen for receiving a connection string from the user in order to connect to the data source. However, this is tedious for users to manually enter, so the scope was changed (with approval from Dr. Allen) to load the .odc files that the Excel Data Connection Wizard creates when it gathers connection data.

Once connections to tables are available, the table editor tools can be used. They are found on the Database Interface tab of the Excel ribbon (see Figure 2 below).

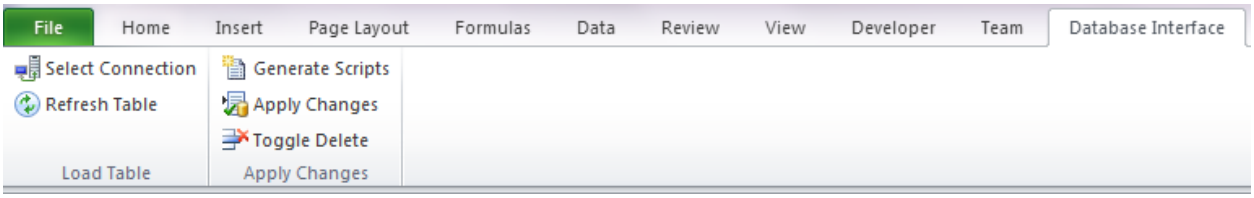


Figure 2

The next step is to use the Select Connection button on the Database Interface tab, which brings up the Connections form (see Figure 3). This form displays the connections that are currently part of the workbook, and it also searches for additional .odc files on the local machine. The default location for these files is C:\...\Documents\My Data Sources. Since different users have different paths to their Documents folder, this initially posed a problem to loading the files. However, after some research I found the SpecialFolders property of the WScript.Shell object for accessing the Documents path for the current user. The connection description shown in the combo box is the workbook connection name assigned by Excel, and represents the server, database, and table names (this also corresponds to the .odc file name, unless manually changed by the user). If there are other .odc files not found in the Documents folder then the user can add them with the command button on the form, which opens a standard file dialog. The text box for the number of records to return will limit how many rows are displayed when the table is loaded. When the user clicks the Load Table button on the Connections form they will be presented with the following prompt:

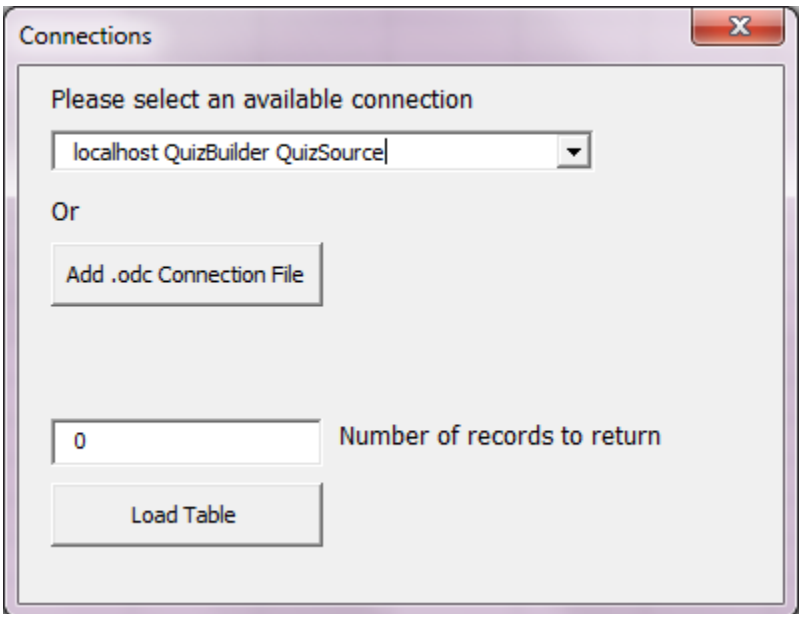


Figure 3

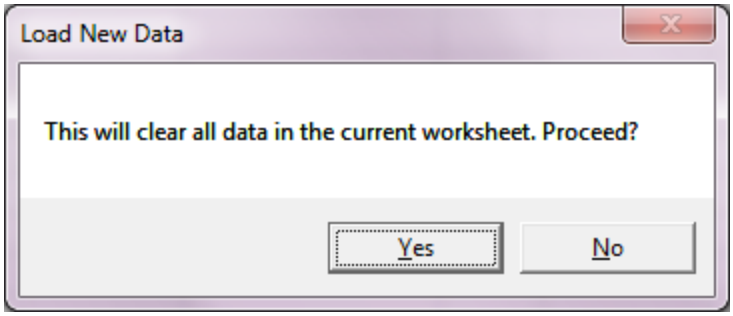


Figure 4

If the user clicks Yes then the Records and RecordVerification worksheets are cleared to prepare for loading the next table. Following this the system checks for a username and password in the connection string for the selected connection. If both are found then the connection is attempted, otherwise a prompt for username and password is displayed. The system has error handling for failed connections. The credential prompt and error handling are shown in Figure 5.

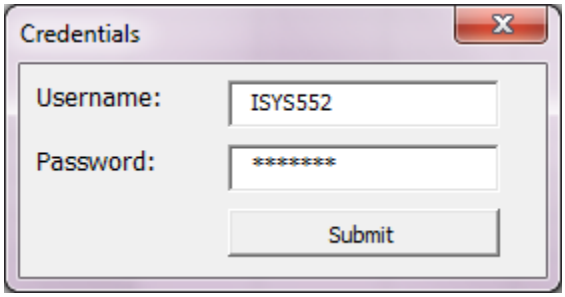
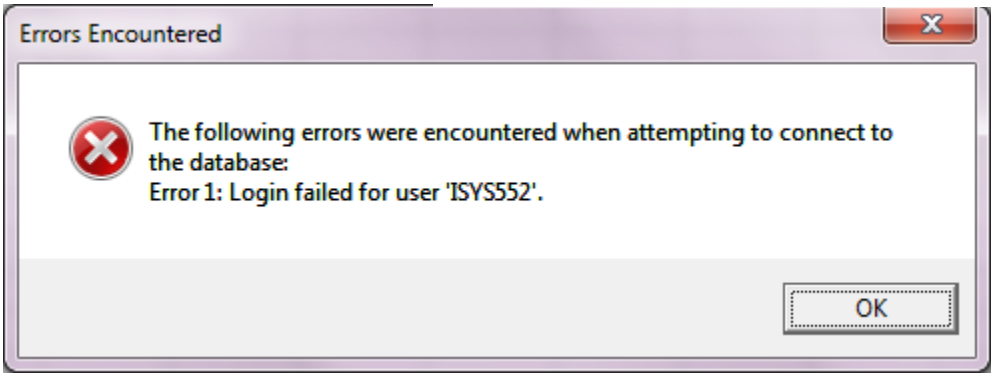


Figure 5

When authentication to the database succeeds, the Records and RecordVerification worksheets are populated with the records from the table, while the table column names appear in the first row of each sheet. The sole purpose of the RecordVerification worksheet is to provide a baseline against which the Records worksheet is compared in order to generate the SQL statements. The RecordVerification worksheet is protected after being populated to prevent users from mistakenly making changes to it.



After the data is loaded the user may change values in fields, add new rows, or mark records for deletion by using the Toggle Delete button on the ribbon (records marked for deletion have the font set to strikethrough as shown below). When the user is finished adding, changing, or marking rows for deletion, there are two options for making the changes to the database. If the Generate Scripts ribbon button is clicked then a SQL Script column to the right of the data set will be populated as shown in Figure 6 below:

	A	B	C	D	E	F	G	H	I	J
1	testInt	testString	testDec	testDate	testBln	testFloat		SQL Script		
2	1	Bob	44.25000	2013-04-15 16:45:32	0	234.2889123				
3	2	Greg	98.24000	2007-10-22 0:00:00	0	7215.5545		UPDATE "QuizBuilder"."dbo"."T		
4	~~3~~	~~Jenny~~	~~511.00000~~	~~2009-02-11 7:19:40~~	~~1~~	~~46.15165~~		~~DELETE FROM "QuizBuilder"."dl~~		
5	4	Nick	82.45000	2012-09-22 0:00:00	0	452.5431		INSERT INTO "QuizBuilder"."db		

Figure 6

The other option for the user is to click the Apply Changes ribbon button. This button will use the connection and the SQL generated to actually apply any changes to the database. Since each SQL statement is representative of 1 record, the expectation is that each executed statement will result in 1 row affected. If this is not the case, or if any other errors occur, the messages are displayed in the column to the right of the SQL statement, and the entire row is highlighted red.

Since update and delete statements include every field value in the WHERE clause, the check for 1 record affected catches instances where the record had been updated or deleted by another process outside of the Excel workbook. In these cases the user has the Refresh Table ribbon button available to reload the worksheets with the latest table values. Examples of the Apply Changes output message and error report are shown below in Figure 7 (the error report is slightly truncated in order to show in this document):

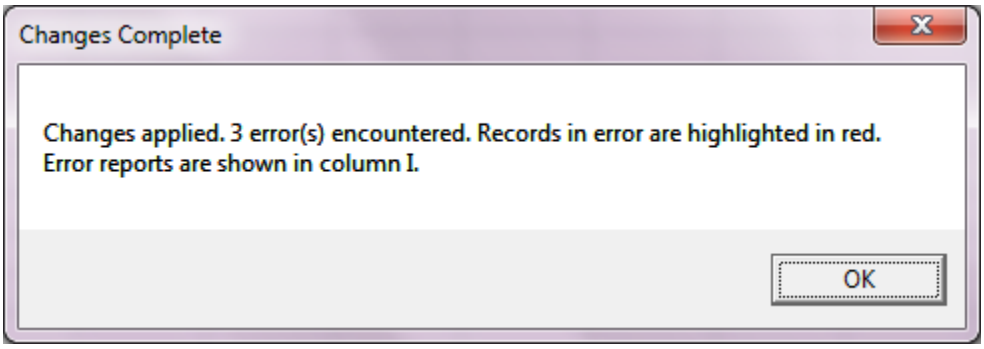
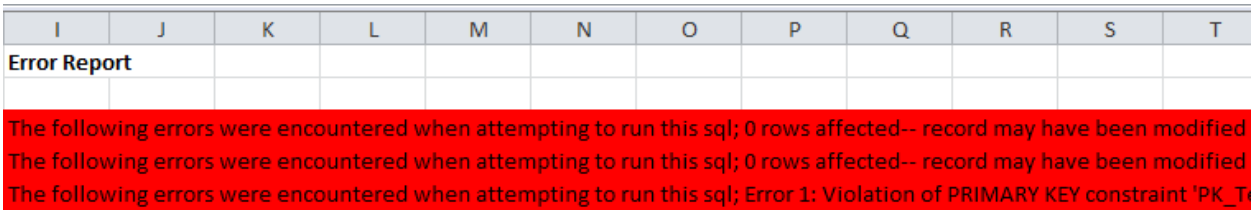


Figure 7



Lessons Learned

I spent a great deal of time in this project studying the ADODB.Connection and ADODB.Recordset objects. One of the critical pieces of this project was figuring out how to determine what the data type was for each field in the recordset. This was vital because generating queries required putting single quotes around strings and dates, and leaving them off of the other types. I found that the data types can be retrieved as enumerated values from the recordset after the query is executed. I was able to match these using a table of ADO data type enumerations to determine which integer values represented which data types. This ultimately allowed me to succeed in building the queries.

Another area that I spent a lot of time studying was the database connection string, and particularly how it is stored in .odc files. I was fortunate to find that .odc files can be easily loaded, but it took me some time to find where they were stored for connections created through the Excel Data Connection Wizard. I was also faced with the difficulty of determining how to get to the Documents folder for a specific user. I found that the WScript.Shell object was particularly helpful in dealing with this issue through the use of the SpecialFolders function. I also explored a few smaller areas of VBA during this project, such as using transactions with the Connection object and utilizing worksheet protection. I also considered using prepared statements when applying the SQL statements directly from Excel, but I felt that it was a trivial precaution since

any user with a valid login with insert/update/delete permissions would be able to do significant damage without having to take advantage of SQL injection techniques. I did, however, remember to escape single quotes in strings when generating the SQL statements.

All of the work on this project was done by me; I did not receive any outside help from another person (besides the answers I found on Google for many different VBA questions). While the workbook itself may not be much to look at on the surface, there are approximately 700 lines of code involved in handling the database connections and generating the SQL statements (not including Module2, which only has test code snippets). And as stated, a good portion of the time spent on this project centered around researching the functionality available in the ADO objects that were at the heart of this workbook.

Potential limitations with the project are that I was only able to test it with a SQL Server database on a Windows 7 machine. I believe that the code is flexible enough to handle Oracle connections, but I don't know if the Oracle connection string would be incompatible with the current code. In addition, I don't know if the workbook would function any differently on a Mac. I also developed this for SQL Server Authentication, and I hadn't attempted connecting with Windows Authentication. Overall, though, I feel that the functionality meets the needs of the business requirements outlined at the beginning. I have worked on a Windows 7 workstation with SQL Server Management Studio for a few years now, and I believe that this workbook will be perfectly suited to the business environment that I designed it for. It is quite likely that I will use this project in my future work with the company.