2SQL

ConvertU2 Technologies
The Data and Software Conversion Experts

2SQL Toolkit

May 2013

Table of Contents

Introduction

In addition to the core 2SQLsoftware ConvertU2 provide software utilities and documented processes which support the process of arriving at the desired result – a fully and completely converted Microsoft Access database to SQL Server

This document explains the features and functionality of the 2SQL Toolkit.

It is expected that a "developer" undertaking a 2SQLConversion Project has a comprehensive knowledge of Microsoft Access and SQL Server programming.

This document should also be read in conjunction with the 2SQL User Guide and 2SQL Technical Reference Guide.

The 2SQL Hosts Utility

The 2SQL Hosts Utility is a suite of software methods in both Access and SQL Server that allows SQL Server to be aware of data values in the Access front end that it otherwise could not.

When a Microsoft Access Database is upsized to SQL Server, some of the properties of the database that JET was able to recognize and use in its processing of SQL Statements become foreign to the corresponding SQL Server Views and Stored Procedures, where JET is no longer used.

The most common example to use that explains this issue are references to Form controls in the SQL Statements. For example:

SELECT * FROM TableName where FieldName = forms!formname!controlname

In this example, the Form control expression is not transferrable to SQL Server, at least not in the same syntax or expression.

Such statements can, of course, be left alone. But to do so means that the statement will be processed on the client side front end using JET, instead of the server. Whenever front end/client side processing of SQL Statements occurs in a Microsoft Access front end/SQL Server back end configuration, performance can be severely compromised, defeating a major purpose of why the database was upsized in the first place. Further, many JET based SQL Statements will be read only when referring to linked SQL Server tables, whereas before the conversion to SQL Server, they were updatable statements.

The solution is to post such expressions to SQL Server as a value that can be recognized by T-SQL, and with the SQL Statement enhanced to refer to this value in the corresponding place that it was used in the original JET Statement.

This section explains how to do this using the 2SQL Hosts Utility. This utility is one of many that ConvertU2 provide in the 2SQL module of every database automatically upsized by 2SQL. Such utilities ensure that every SQL Statement in the entire application can receive server based processing when they are executed, with a minimum of manual coding to achieve it.Importantly,the2SQL Genie implements the 2SQL Hosts Utility automatically where needed, so each conversion solution has plenty of examples which can be referred to when future enhancements are required and to keep the 2SQL Methodology intact.

A comprehensive knowledge of SQL Server T-SQL, Microsoft Access JET, and Visual Basic for Applications (VBA) is assumed in this next section about the framework of the 2SQL Hosts Utility.

Back End Architecture Framework

In the SQL Server back end, the 2SQL Hosts Utility consists of a SQL Server table called tblHost_Values, and a suite of User Defined Functions.

The 2SQL HOSTS Utility Table.

The Hosts table consists of a composite primary key by way of 2 columns called HostName and HostKey. HostName always has a deterministic value which is derived by a JOIN to the SQL Server View called v_CustomLoginID. This view typically returns the value of SQL Server HOST_NAME() function, which is the computer name of the desktop that is using the Microsoft Access front end, and is therefore a reliable means for the other 2SQL Toolkit objects to determine which values posted to the Hosts table, belong to which Host. The HostKey column is used to associate values in the (Hosts) table to a specific purpose from the Access Front End, because there can also be the requirement to not only identify the Host, but also more than one purpose occurring concurrently.

The other columns of the Hosts table are as follows:

VarcharValue, BitValue,IntegerValue, DateTimeValue,FloatValue,MoneyValue and VarbinaryValue

These columns are defined with a SQL Server column type as per their name suggests. For example, the column called VarcharValue is defined as type Varchar, and so on.

When values are passed to this table from the front end, they must be passed to the appropriate and corresponding column type. For example, a value of type string in the front end, must be stored in the column called VarcharValue in the Hosts table, and so on. A strong feature of the 2SQL Hosts Utility is its scalability. There is no need to change the schema if there is a new requirement. Simply introduce a new host key for every new requirement.

The 2SQL HOSTS Utility User Defined Functions.

Each "Value" column in the Hosts table has a corresponding SQL Server User Defined Function. For example, Function Host_VarcharValue for column VarcharValue and so on. These functions provide the mechanism to pull values as and when required, from the Hosts table for use in the SQL Statements.

For example, going back to the sample statement used earlier:

SELECT * FROM TableName where FieldName = forms!formname!controlname

Becomes

SELECT * FROM TableName where FieldName = dbo.Host_VarcharValue('HostKey')

Or

SELECT * FROM TableName CROSS JOIN dbo.fntblHost_Values('HostKey') DerivedTable1 where Tablename.FieldName = DerivedTable1.VarcharValue

The value retrieved is posted to the Hosts table from the Access front end, prior to the execution of this statement.

Front End Architecture Framework

In the Microsoft Access Front End, the 2SQL Hosts Utility consists of a suite of VBA functions that reside in the 2SQL Module. These functions are used to post and manage values in the Hosts Table that would otherwise be foreign to SQL Server.

The SetHostValue VBA Function.

The SetHostValue function is essentially the gateway to posting data values to the Hosts table from the Access front end. It is simple to use, and although how it works is beyond the scope of this section, users are invited to review the source code for a higher level of understanding.

Using our sample SQL statement, if a value of type string requires posting to the Hosts Table, the VBA syntax is as follows:

> SetHostValue "forms_formname_controlname", forms!formname!controlname, vtstring
>
> Where:
>
> "forms_formname_controlname" is the unique HostKey
>
> forms!formname!controlname is the corresponding data value associated to HostKey
>
> vtstring is a component of a VBA enum called enumVarType in the 2SQL Module. In this instance, passing across vtstring tells SetHostValue to post the value to the

VarcharValuecolumn of the HostTable. A type of vtBoolean will instruct SetHostValue to post the value to the BitValue column, and so on.

The ClearHostRows VBA Function.

The ClearHostRows function will remove all rows in the Hosts table that belong to the Host, which is the computer name of the user who is using the Access Front End. It should be part of the Autoexec macro or Startup Form object to ensure that each new user session does not have a residue of values in the Hosts table from earlier sessions. It can also be used as and when required elsewhere in the VBA code if there is a specific circumstance to do so. To date however, there have been no such circumstances as a result of 2SQL exposure to the marketplace.

The ClearHostValues VBA Function.

The ClearHostValues function will remove all rows in the Hosts table that belong to the Host AND Host Key. Note that this function also has a wildcard option. If the wildcard option is used, then this function will remove all rows from the Hosts table that have a HostKey value starting with the value of the HostKey parameter passed in. Otherwise it will remove rows exclusive to the explicit value of the HostKey parameter only.

Guidelines on when to use the 2SQL Hosts Utility

Although the 2SQL Hosts Utility can be used anywhere from the Access front end, there are a suite of conditions in particular where it should be used. These are:

1. Whenever the value is recognized by Jet but not T-SQL, such as a Form control, AND
2. Whenever the SQL Statement was originally the RecordSource property of a Form OR
3. Whenever the SQL Statement was originally the RecordSource property of a Report OR
4. Whenever the SQL Statement was originally the Rowsource property of a Form or Report Control.

2SQL would have converted these statements to SQL Server Views or Stored Procedures, with the 2SQL Hosts Utility implemented automatically as part of the solution inside the View/Procedure. However, there may be other areas in the VBA Code that need calls to SetHostValue. See the 2SQL Conversion Services Guide for more information.

The points above will ensure server side processing of the SQL Statements when Form Control values or other expressions recognized by JET and not T-SQL, reside in these properties of the Forms and Reports. These are guidelines only however. Although they cover most of the circumstances to ensure correct results, only a thorough understanding of the Access application can determine exactly where and when to make a call to SetHostValue.

In addition, a common occurrence in VBA code is the dynamic use of variables (as opposed to Form controls recognized by JET) when generating SQL Statements that belong to the Rowsource property. Although the Hosts Utility can be used here, there is a better way to ensure server side processing of this (read only) property. This is to use the 2SQL SetSQLStatement function instead. See the section of the same name in this user guide for more information.

Restrictions of the 2SQL Hosts Utility

The operational functionality of the 2SQL Hosts Utility can be circumstantially restricted if there is a need to open the Access front end more than once concurrently on the same desktop. This can potentially cause the 2SQL Hosts Utility to malfunction because it identifies everything at the HOST (Computer Name) level and not the session level. However, the same form(s) that use the Hosts table would also need to be referenced by both sessions at the same time to cause a malfunction. This is a very rare circumstance and to date has not been a condition of databases upsized to SQL Server by 2SQL.

If the user interface of the Access Application provides Access to the object viewer (not recommended), extra coding may be required to ensure correct functionality of objects now dependant on the 2SQL Hosts Utility. For example, if a Form is now dependant on the Hosts Utility to return records, it cannot be opened directly from the Form object viewer, because there is no way to intervene and make a call to SetHostValue.

The SQLStatements Utility

This section explains how to implement the 2SQL SQLStatement Utility to optimize properties of forms and reports that can contain SQL Statements, and are dynamically set at run time in the VBA Code.

SQL Statements typically reside in the VBA Code of a Microsoft Access Database Application because the values in the statement can only be determined at run time, not design time. Such statements can be applied to a variety of objects and properties.

Such objects can, of course, be left alone. But to do so means that the SQL Statement will be processed on the client side front end using JET, instead of the server. Whenever front end/client side processing of SQL Statements occurs in a Microsoft Access front end/SQL Server back end configuration, performance can be severely compromised, defeating a major purpose of why the database was upsized in the first place.

For example:-

```
Me.RecordSource = “SELECT * FROM TableName WHERE DateField< #”
&forms!formname!controlname& “#”
```

The next section explains how to implement the 2SQL SQLStatement Utility to optimize such objects. This function is one of many utilities that ConvertU2 provide in the 2SQL module of every database automatically upsized by 2SQL. Such utilities ensure that every SQL Statement in the entire application can receive server based processing when they are executed, with a minimum of manual coding to achieve it.

Back End Architecture/Framework

In the SQL Server Back End database created by 2SQL for the corresponding Access Front End, there will be a table called SQLStatements, and a stored procedure called usp_SQLPassthru.

The SQLStatements Table.

The SQLStatements table consists of a composite primary key by way of 2 columns called HostName and SQLStatementKey, and another data column called SQLStatement. The HostName column is part of the 2SQL Hosts Utility which has already been explained in the previous section. The SQLStatementKey column is used to associate values in this table to a specific SQL Statement stored in the SQLStatement column, and sent from the Access Front End.

The usp_SQLPassThru Stored Procedure.

This procedure represents the mechanism by which an SQL Statement is executed from a pass thru query in the front end. It is dynamic in nature, and completely injection proof. Although it is beyond the scope of this section to explain how it works, users of 2SQL are invited to view the source code of this stored procedure to increase their own understanding.

Front End Architecture Framework

In the Microsoft Access Front End, the SetSQLStatement function resides in the 2SQL Module. In addition, a pass thru query will reside in the front end for each unique call to this function by way of SQLStatementKey.

The SetSQLStatement VBA Function.

The SetSQLStatement function accepts two parameters named SQLStatementKey, and SQLStatement . These parameters represent the data values of what their respective names suggest.

When this function is called in VBA, it will perform two basic operations:

1. It will post the actual SQLStatement to the SQLStatements table in the SQL Server back end, based on the Host Name, and the value of SQLStatementKey.

2. It will create a corresponding pass thru query object with a name the same as SQLStatementKey, if it does not exist from previous calls to SetSQLStatement.

The corresponding Pass Thru Query Object.

For each unique SQLStatement key, there is a corresponding pass thru query object of the same name. The SQL property of this object will make a call to the usp_SQLPassThru Stored Procedure, passing the value of SQLStatementKey as a parameter. This object represents the gateway from the front end to the back end for the execution of a dynamic SQL Statement completely optimized.

Implementation

Using our sample instruction above:

Me.RecordSource = "SELECT * FROM TableName WHERE DateField< #" &forms!formname!controlname& "#"

becomes

SetSQLStatement "FormName_RS","SELECT * FROM TableName WHERE DateDiff(dd,DateField,'" & Format(forms!formname!controlname,"yyyy/mmm/dd") & "') > 0"

Me.RecordSource = "FormName_RS"

This will also create a pass thru query called FormName_RS making the RecordSource property completely functional and optimized. The SQL property of the pass thru query in this example will have a value of EXEC usp_SQLPassThru 'FormName_RS'

Using SetSQLStatement, it is possible to optimize several hundred read only objects that refer to VBA SQL Statements very quickly.

Guidelines on when to use the SQLStatement Utility

SetSQLStatement can typically be used to optimize the processing of SQL Statements in the VBA Code in the following situations:

5. A read only RecordSource property of a Form or Report
6. A read only OLE Based ADO RecordSet
7. Commands such as TransferSpreadSheet that require a query object.

In addition, SetSQLStatement can also be used to optimize VBA references to the RowSource property of Form and Report controls. It is recommended however, to use another 2SQL Function called SetRowSource to do this. SetSQLStatement is an integral part of this function as well. For more information, see the section titled How to Optimize VBA Rowsource Objects using SetRowSource.

Restrictions of the SQLStatement Utility

Because pass thru queries are read only, SETSQLStatement cannot be used to optimize the RecordSource property of an updateable Form, or for instantiating an updateable RecordSet.

Pass thru queries can also not be used on the recordsource property of SubForms and SubReports, even if they are read only. This is a restriction imposed by Microsoft Access.

There are no other restrictions.

See the section on the 2SQL Hosts Utility for optimizing objects in the VBA code that need to remain updateable, or for objects that cannot use pass thru queries..

Optimizing VBA RowSource Objects Using SetRowSource

This procedure is one of many that ConvertU2 provide in the 2SQL module of every database automatically upsized by 2SQL. Such utilities ensure that every SQL Statement in the entire application can receive server based processing when they are executed, with a minimum of manual coding to achieve it.

When a Microsoft Access Database Application is converted to SQL Server by 2SQL all of the SQL Statements that reside in the RowSource Objects of Form and Report Controls (egListBoxes, ComboBoxes, etc), are converted to SQL Server Views or Stored Procedures. This effectively changes the processing of these objects from JET/Client Side based, to Server Side, thereby optimizing them for performance as much as possible.

It is a common practice by Microsoft Access software developers however, to also initialize these objects at run time in VBA. The reason for this is because quite often, the SQL Statement belonging to the Rowsource object depends on values that can only be determined at run time, not design time.

For example:-

Me.Control.RowSource = "SELECT Field1 FROM TableName WHERE DateField< #" &me.txtDate& "#"

Although 2SQL does not convert SQL Statements that reside in the VBA code, a customized VBA procedure called SETRowSource is provided at no charge for developers to implement and optimize all RowSource objects in the VBA Code.

Such statements can, of course, be left alone. But to do so means that the statement will be processed on the client side front end using JET, instead of the server. Whenever front end/client side processing of SQL Statements occurs in a Microsoft Access front end/SQL Server back end configuration, performance can be severely compromised, defeating a major purpose of why the database was upsized in the first place.

Note: It is also recommended to be completely familiar with the functionality of the 2SQL SQLStatement Utility before reading the rest of this section. See the section titled How and When to Implement the 2SQL SQLStatement Utility for more information.

Basic Functionality

The source code to SetRowSource is very simple, although it does depend on other procedures provided by ConvertU2:

```
Public Sub SetRowSource(rsobject As Object, sqlstatement As String, sqlstatementkey As String)

       If sqlstatementkey<> "" Then
SetSQLStatementsqlstatementkey, sqlstatement
rsobject.RowSource = sqlstatementkey
       Else
RowSourceValueListrsobject, sqlstatement
       End If


End Sub
```

The rsobject variable represents the rowsourceobject.

The sqlstatement variable represents as the name suggests.

And the sqlstatementkey variable represents the unique key for the SQLStatements table created by 2SQL in the SQL Server back end. See the section titled How and When to Implement SetSQLStatement to Optimize Objects with VBA SQL Statements for more information.

The logical flow of SetRowSource is controlled by the sqlstatementkey variable.

If sqlstatementkey is not an empty string, control will pass to SetSQLStatement, which will send the SQL Statement to the SQL Server Back End, and create a corresponding pass thru query in the front end, and then set the value of the rowsource object, also to this key.

If sqlstatementkey is an empty string, control will pass to RowSourceValueList, which will change the rowsourcetype property of the rowsource object to Value List, and create a string of values derived from the SQL Statement via an OLE Based ADO RecordSet.

In most instances, passing control to SetSQLStatement will optimize processing completely. Passing control to RowSourceValueList may perform better than a pass thru query in a WAN environment as opposed to LAN, because an ODBC based Pass Thru object can suffer serious performance degradation "Over the Wire". Value Lists however, amongst other limitations, have a 2048 character length limit, so unless it is known at run time that this will not be exceeded, it is best used only when all other options have been exhausted.

Implementation

Using our example above, to make a rowsource use a pass thru query:-

```
Me.Control.RowSource = "SELECT Field1 FROM TableName WHERE DateField< #" &me.txtDate& "#"
```

becomes

```
SetRowSourceme.Control,"SELECT Field1 FROM TableName WHERE DateDiff(dd,DateField,'" & format(me.txtdate,"yyyy/mmm/dd") & "') > 0,"formname_controlname"
```

Using SetRowSource, it is possible to optimize several hundred VBA rowsource objects very quickly. A future version of 2SQL will assist in the automation of this methodology.

The ExecuteScalar Utility

When an Access database is processed by 2SQL in Genie (Migrate and Convert) mode, a module called 2SQL is automatically created in the new front end. This module contains optimized versions of the VBA Domain Aggregate functions such as DLookup (DLookUpDirect), DCount (DCountDirect), etc. These functions use the 2SQL ExecuteScalar Utility as part of their underlying framework to return single values.

This section explains how to use the ExecuteScalar Utility for purposes above and beyond that of the domain aggregate functions. It is a server side processing based utility that can be called whenever there is a need to return a single value. It should only be used when the Server Side Processing Solution Type was chosen to migrate and convert to SQL Server using 2SQL.

Back End Architecture/Framework

In the SQL Server Back End database created by 2SQL for the corresponding Access Front End, there will be a stored procedure called usp_ExecuteScalar.

This stored procedure accepts an SQL Statement to execute, and returns the result of the statement in an output parameter. It uses the generic SQL Server function sp_executesql with injection proof syntax to calculate the result of the SQL Statement.

Front End Architecture/Framework

In the Microsoft Access Front End, the ExecuteScalar function resides in the 2SQL Module.

This function accepts an SQL Statement as a parameter, and uses an OLE based ADO Command to execute the usp_ExecuteScalar stored procedure. It will return a value of type Variant so as to allow the calling routine to receive a null value or a value of any field type, just like the generic domain aggregate functions.

Implementation

The ExecuteScalar utility can typically be used as an extension to the limitation of the domain aggregate functions, which only accept table or query objects as the domain parameter. The only restriction is that the actual SQL Statement to execute must be of a syntax which ensures a single or null return value.

For example:

```
SELECT COUNT(*) FROM Table1 INNER JOIN Table2 ON Table1.Field1 = Table2.Field1 WHERE
Table1.Field3 = ‘xxx’
```

This utility can also often be used to replace existing VBA code that uses DAO or ADOrecordsets to pull back the first column of the first row, offering less overhead and less coding to achieve the same result.

The Connection Strings Utility

When an Access database is processed by 2SQL in Genie (Migrate and Convert) mode, a table called zstblConnectionStrings is automatically created to store the connection properties of the front end that are used to communicate to the corresponding SQL Server back end. A macro called SQLServerConnectionSettings is also created providing user access to this table for maintenance and implementation.

This section explains how to maintain and implement the connection properties in the front end using the 2SQL Connection Settings Utility. Users will find this utility particularly useful when moving their SQL Server database to a different server. For example, when it is time to release the application from a development/testing environment to the production environment.

Maintaining the tblPseudo_Indexes Table

When ODBC links in the front end refer to SQL Server Views in the back end, a unique index is required as part of the ODBC Link if the View needs to be updateable from the front end. This unique index is only used by Microsoft Access and not SQL Server, effectively making it a pseudo index,.because SQL Server does not need it. 2SQL automatically detects this requirement when it processes a database and stores the information required to create these indexes into the SQL Server tblPseudo_Indexes table. When new Views are created or existing ones modified, a corresponding row in this table must be added/maintained as follows:-

1. Enter the name of the SQL Server View in the ObjectName column

2. Enter the column name of the view that represents the unique index, into the column called Columns. If there is more than one column, separate each column with a comma.

Please note also that the column called UniqueTable in this table is not required to create any pseudo indexes. This column is for documentation purposes only and stores as the name suggests, being the name of the table that the View is uniquely bound to.

Maintaining the Connection Properties.

To maintain the connection properties, execute the SQLServerConnectionSettings macro. A form will display enabling entry of the server name, authentication method, etc. When selecting the type of SQL Server driver, use the standard driver if the SQL Server Native Client Driver is not installed (not recommended), otherwise version 9.0 for SQL Server 2005, and version 10.0 for SQL Server 2008. The Native Client driver forms part of the SQL Server 2008/2005 client tools but it is also available as a separate installer from Microsoft.

Refreshing of ODBC Links

To recreate/refresh all of the ODBC Links, click on the command button to Relink Tables and Queries. The following will occur:-

1. An ODBC Table link will be created for each Table and View in the SQL Server back end.

2. The pseudo indexes for each View will be created for the corresponding ODBC Table link.

3. All existing Access Query objects that are ODBC Pass Through links will be refreshed.

The ADODirectConnection Function.

This VBA function returns an OLE Based ADODB Connection and is used extensively in the VBA Code to open ADODB Recordsets or ADODB Commands. Its properties are derived from the zstblConnectionStrings Table. The function will fire up as soon as it is referred to and remain open for the entire session of the application. The ADODirectConnection function represents the optimized equivalent to the JET based CurrentProject.Connection property and should be used wherever possible to maximize performance. When using ADODirectConnection as the connection object for ADODB Recordsets and Commands, the syntax of any SQL Statements must be T-SQL based, not JET based. Also, the ADODirectConnection function does not depend on any of the ODBC links to do its processing.

Other Microsoft Access 2SQL Toolkit Functions

When an Access database is processed by 2SQL in Genie mode, a module called 2SQL Toolkit is automatically created in the new front end. This module contains several functions that form part of the 2SQL Toolkit with their own sections in this document.

This section provides a brief explanation of all the other functions in the 2SQL Toolkit module that do not have their own separate sections. The VBA Functions and Procedures are listed alphabetically.

CACHEDADORECORDSET

This procedure will return a cached ADO Recordset with a defaultcachesize of 30. The cache size was determined by several hours of research and development as the best general fit to maximize performance when communicating with a SQL Server back end. It can be changed dynamically for more specific usage.

DCOUNTDIRECT

This procedure is the optimized equivalent of the generic VBA Function DCOUNT. It enforces Server Side processing whereas the DCOUNT function is client side processing based. The 2SQL Genie replaces DCOUNT with DCOUNTDIRECT automatically everywhere, as well as the other aggregate functions being DSUMDIRECT, DMINDIRECT,DMAXDIRECT.

DELETEQUERY2SQL

This procedure is a wrapper for DoCmd.DeleteObjectacQuery. The 2SQL Genie implements this wrapper automatically. It is needed because some of the original queries in Microsodt Access will be linked Microsoft Access Tables in the front end, and SQL Server Views in the back end.

DELETETABLE2SQL

This procedure is a wrapper for DoCmd.DeleteObjectacTable. The 2SQL Genie implement this wrapper automatically. It is needed because the original tables are now linked tables, and to achieve equivalent functionality after conversion to SQL Server, the SQL Server table itself must also be dropped.

EXECUTESQLCOMMAND

This procedure will do as the name says via an ADODB Command object, using CurrentProject.Connection or the 2SQL ADODirectConnection as the ADODB Connection object. It is recommended to always pass ADODirectConnection as the connection object unless the corresponding SQL Statement refers to local tables. See the section about the 2SQL Connection Settings Utility for more information about ADODirectConnection.

HANDLEQUOTE

This method can be used to encapsulate a string value with singular quotation marks to assist in the creation of dynamic SQL Statements in the VBA Code.

INDEXVIOLATED

This function can be used to replace areas of the code that trap the error number 3022, which occurs on local tables when a primary or unique index has been violated. This error number changes when the tables reside in SQL Server.

INITLINKEDSUBFORMS

This method is called from the form_open event procedure, for Forms that have SubForms. It is one of the more complicated methods of the 2SQL Toolkit and its purpose is to eliminate the drag that would otherwise occur now that the tables and queries for these forms are in SQL Server. Explained by example, if a customer form uses a subform to display all the orders for a customer, and the orders table has tens of thousands or more records in total, but the customer key selected has only 10 orders, Access will download the entire Orders table to the front end first, and filter out the Orders that do NOT belong to the customer selected. This makes the form completely dysfunctional. InitLinkedSubForms is used to store the data needed to apply the filter in SQL Server thereby making the form very fast in performance. The 2SQL Genie implements this method automatically. See also the 2SQL Toolkit method called SynchLinkedSubForms.

OBJECTEXISTS2SQL

This function can be used to determine if the name passed to it, is one of the objects (tables, macros,queries,forms,reports,modules) in Microsoft Access

OPENFORM2SQL

This procedure is a wrapper for DoCmd.OpenForm. It is needed because calls may be needed to the 2SQL Hosts Utility method called SetHostValue, which is done via to call to CheckFormInputParameters. The 2SQL Genie implements this wrapper automatically, and also OpenReport2SQL, and OpenQuery2SQL.

REFRESHLINK

The purpose of this procedure is to refresh ONE linked table to SQL Server, which actually makes all linked tables accessible and operational. It is called from the AutoExec macro which the 2SQL Genie creates automatically.

REFRESHODBCLINK

This procedure will refresh an existing ODBC Link, or create a new Link to a SQL Server Table or View in the back end. When SQL Server tables or views are created or modified, the links in the front end must be refreshed. Using this procedure to perform this task will derive the Connect property from the 2SQL ODBCConnectionString function. See the section about the 2SQL Connection Settings Utility for more information about ODBCConnectionString.

SQLDATEFORMAT

This function will accept a date parameter and return a string of this parameter in yyyy/mmm/dd format. This particular format is reliable for comparing fields of type datetime in SQL Server tables when using VBA code to dynamically generate SQL Statements.

STUFFXML

This function provides a very quick way of concatenating one or more fields of a table together. It can be used to replace the more code intensive RecordSet looping of records, and is also significantly faster. An understanding of the SQL Server STUFF function and FOR XML is required.

SYNCHLINKEDSUBFORMS

This method is called from the form_current and form_afterinsert event procedures, for Forms that have SubForms. It optimizes performance by using the metadata collected by its sister 2SQL Toolkit method called InitLinkedSubForms. The 2SQL Genie implements this method automatically.

VIEWSQL

This function provides a very quick way of retrieving the SQL Statement of a SQL Server View. It can be used typically to replace VBA code that originally retrieved the SQL Statement of an Access Query via the SQL property of a DAO Querydef.

Other SQL Server 2SQL Toolkit Functions

This section provides a brief explanation of all the other SQL Server 2SQL Toolkit User Defined Function do not have their own separate sections in this document.

BETWEENDATE2SQL

This function provides a way to use the BETWEEN clause to avoid convoluted expressions in SQL Statements due to the differences between Microsoft Access and SQL Server for this predicate. See the 2SQL Technical Reference Guide for more information.

COMMANUMBER2SQL

This function inserts commas into Money values. The 2SQL Genies implements this function automatically when converting the equivalent Microsoft Access FORMAT function.

XXX2SQL

There are several other functions that represent their Microsoft Access equivalents. For example the equivalent to the Microsoft Access PARTITION function is PARTITION2SQL. The 2SQL Genie implements these automatically when required by the conversion process.

Disclaimer

The Access to SQL conversion and migration challenge is a very complex subject. Whilst every effort has been made to completely and accurately represent the conversion issues and their corresponding solution, we do not claim perfection in our analysis. For that reason we welcome reader feedback to info@cu2global.com as part of our striving to continue to make 2SQL is the best product of its type in the world market today.