SQL Server extends the functionality of table-valued functions by allowing you to define a table-valued function in any managed language. Data is returned from a table-valued function through an IEnumerable or IEnumerator object.

For table-valued functions, the columns of the return table type cannot include timestamp columns or non-Unicode string data type columns (such as char, varchar, and text). The NOT NULL constraint is not supported.

Implement table-valued functions as methods on a class in a Microsoft .NET Framework assembly. Your table-valued function code must implement the IEnumerable interface. The IEnumerable interface is defined in the .NET Framework. Types representing arrays and collections in the .NET Framework already implement the IEnumerable interface. This makes it easy for writing table-valued functions that convert a collection or an array into a result set.

**Defining a Table-Valued Function in Transact-SQL**

The syntax for defining a CLR table-valued function is similar to that of a Transact-SQL table-valued function, with the addition of the EXTERNAL NAME clause. For example:

```sql
CREATE FUNCTION GetEmpFirstLastNames()
RETURNS TABLE (FirstName NVARCHAR(4000), LastName NVARCHAR(4000))
EXTERNAL NAME MyDotNETAssembly.[MyNamespace.MyClassname].GetEmpFirstLastNames;
```

Table-valued functions are used to represent data in relational form for further processing in queries such as:

```sql
select * from function();
selct * from tbl join function() f on tbl.col = f.col;
selct * from table t cross apply function(t.column);
```

Table-valued functions can return a table when:

- Created from scalar input arguments. For example, a table-valued function that takes a comma-delimited string of numbers and pivots them into a table.
- Generated from external data. For example, a table-valued function that reads the event log and exposes it as a table.

Note: A table-valued function can only perform data access through a Transact-SQL query in the InitMethod method, and not in the FillRow method. The InitMethod should be marked with the SqlFunction.DataAccess.Read attribute property if a Transact-SQL query is performed.
The following table-valued function returns information from the system event log. The function takes a single string argument containing the name of the event log to read.

**Sample Code**

**C#**

```csharp
using System;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;
using System.Collections;
using System.Data.SqlTypes;
using System.Diagnostics;

public class TabularEventLog
{
    [SqlFunction(FillRowMethodName = "FillRow")]
    public static IEnumerable InitMethod(string logname)
    {
        return new EventLog(logname).Entries;
    }

    public static void FillRow(object obj, out SqlDateTime timeWritten,
        out SqlChars message, out SqlChars category, out long instanceId)
    {
        EventLogEntry eventLogEntry = (EventLogEntry)obj;
        timeWritten = new SqlDateTime(eventLogEntry.TimeWritten);
        message = new SqlChars(eventLogEntry.Message);
        category = new SqlChars(eventLogEntry.Category);
        instanceId = eventLogEntry.InstanceId;
    }
}
```

**Declaring and Using the Sample Table-Valued Function**

After the sample table-valued function has been compiled, it can be declared in Transact-SQL like this:

```sql
use master;
-- Replace SQL_Server_logon with your SQL Server user credentials.
GRANT EXTERNAL ACCESS ASSEMBLY TO [SQL_Server_logon];
-- Modify the following line to specify a different database.
ALTER DATABASE master SET TRUSTWORTHY ON;

-- Modify the next line to use the appropriate database.
CREATE ASSEMBLY tvfEventLog
FROM 'D:\assemblies\tvfEventLog\tvfeventlog.dll'
WITH PERMISSION_SET = EXTERNAL_ACCESS;
GO
CREATE FUNCTION ReadEventLog(@logname nvarchar(100))
RETURNS TABLE
(logTime datetime, Message nvarchar(4000), Category nvarchar(4000), InstanceId bigint)
AS
```
Visual C++ database objects compiled with /clr:pure are not supported for execution on SQL Server 2005. For example, such database objects include table-valued functions.

To test the sample, try the following Transact-SQL code:

```sql
-- Select the top 100 events,
SELECT TOP 100 *
FROM dbo.ReadEventLog(N'Security') as T;
go

-- Select the last 10 login events.
SELECT TOP 10 T.logTime, T.Message, T.InstanceId
FROM dbo.ReadEventLog(N'Security') as T
WHERE T.Category = N'Logon/Logoff';
go
```

The following sample shows a table-valued function that queries a SQL Server database. This sample uses the AdventureWorks Light database from SQL Server 2008. See [http://www.codeplex.com/sqlserversamples](http://www.codeplex.com/sqlserversamples) for more information on downloading AdventureWorks.

Name your source code file FindInvalidEmails.cs.

Compile the source code to a DLL and copy the DLL to the root directory of your C drive. Then, execute the following Transact-SQL query.

```csharp
using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class UserDefinedFunctions { 
    private class EmailResult { 
        public SqlInt32 CustomerId;
        public SqlString EmailAdress;

        public EmailResult(SqlInt32 customerId, SqlString emailAdress) { 
            CustomerId = customerId;
            EmailAdress = emailAdress;
        }
    }
```
public static bool ValidateEmail(SqlString emailAddress) {
    if (emailAddress.IsNull)
        return false;
    if (!emailAddress.Value.EndsWith("@adventure-works.com"))
        return false;
    // Validate the address. Put any more rules here.
    return true;
}

[SqlFunction(
    DataAccess = DataAccessKind.Read,
    FillRowMethodName = "FindInvalidEmails_FillRow",
    TableDefinition = "CustomerId int, EmailAddress nvarchar(4000)")]
public static IEnumerable FindInvalidEmails(SqlDateTime modifiedSince) {
    ArrayList resultCollection = new ArrayList();
    using (SqlConnection connection = new SqlConnection("context connection=true"){
        connection.Open();
        using (SqlCommand selectEmails = new SqlCommand(
            "SELECT [CustomerID], [EmailAddress] " +
            "FROM [AdventureWorksLT2008].[SalesLT].[Customer] " +
            "WHERE [ModifiedDate] >= @modifiedSince",
            connection)) {
            SqlParameter modifiedSinceParam =
                selectEmails.Parameters.Add(
                    "@modifiedSince",
                    SqlDbType.DateTime);
            modifiedSinceParam.Value = modifiedSince;
            using (SqlDataReader emailsReader =
                selectEmails.ExecuteReader()) {
                while (emailsReader.Read()) {
                    SqlString emailAddress =
                        emailsReader.GetString(1);
                    if (ValidateEmail(emailAddress)) {
                        resultCollection.Add(new EmailResult(
                            emailsReader.GetInt32(0),
                            emailAddress));
                    }
                }
            }
        }
    }
    return resultCollection;
}

public static void FindInvalidEmails_FillRow(
    object emailResultObj,
out SqlInt32 customerId,
out SqlString emailAdress) {
EmailResult emailResult = (EmailResult)emailResultObj;

customerId = emailResult.CustomerId;
emailAdress = emailResult.EmailAdress;
}

Compile the source code to a DLL and copy the DLL to the root directory of your C drive. Then, execute the following Transact-SQL query.

use AdventureWorksLT2008;
go

IF EXISTS (SELECT name FROM sysobjects WHERE name = 'FindInvalidEmails')
    DROP FUNCTION FindInvalidEmails;
go

IF EXISTS (SELECT name FROM sys.assemblies WHERE name = 'MyClrCode')
    DROP ASSEMBLY MyClrCode;
go

CREATE ASSEMBLY MyClrCode FROM 'C:\FindInvalidEmails.dll'
WITH PERMISSION_SET = SAFE -- EXTERNAL_ACCESS;
go

CREATE FUNCTION FindInvalidEmails(@ModifiedSince datetime)
RETURNS TABLE (  
    CustomerId int,
    EmailAddress nvarchar(4000)
)  
AS EXTERNAL NAME MyClrCode.UserDefinedFunctions.[FindInvalidEmails];
go

SELECT * FROM FindInvalidEmails('2000-01-01');
go

References: