بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

# آموزش مجتمع سازی CLR در SQL Server 2012 قسمت سوم

مدرس : مهندس افشین رفوآ

رمز فایل :tahlildadeh.com



اگر فرض کنیم که متد GetStrings() تحت عنوان یک TVF با همان نام ثبت شده است، آنگاه مثلا زیر، قطعه ای از T-SQL از یک stored procedure است که از این TVF برای بیرون کشیدن line items (extract) ، در شکل جدولی استفاده می کند.

```
CREATE PROCEDURE Insert_Order @cust_id int, @lineitems nvarchar(8000)
AS
BEGIN
    ...
    INSERT LineItems
    SELECT * FROM dbo.GetStrings(@lineitems)
    ...
END
```

## کدام را استفاده کنیم؟

تصمیم در مورد اینکه یک stored procedure را در ترکیب با SqlPipe- چه غیر مستقیم در T-SQL و چه مستقیم د ر یک CLR routine- یا یک table-valued function استفاده کنیم، بستگی به فاکتورهای زیادی دارد که باید مد نظر قرار گیرند، که شامل composability requirements ، منبع داده ها، نیاز به اثرات جانبی، و typing requirements برای خروجی است. به ترتیب هر کدام از آنها را مورد بحث قرار می دهیم.

## Composability Requirements

شاید استفاده دوباره یا تغییر (manipulation) خروجی بدست آمده درون یک TVF یا یک stored procedure، مطلوب باشد. Table-valued functions از نقطه نظر composability ، پرکاربردتر هستند، مثلاً نوع بازگشت (return type) یک TVF ، یک rowset نسبی است که در هر جاییکه چنین ساختاری مجاز است، قابل استفاده است. مخصوصاً، دربخش FROM در عبارت SELECT ، قابل استفاده است، و چنین خروجیی می توانند از composability of SELECT در sub-query ها، عبارات INSERT…SELECT ، جدولهای مشتق شده، و جدول عبارات استفاده کنند.

از سوی دیگر، ، می توان stored procedure ها را فقط به عنوان بخشی از ترکیب INSERT…EXEC ، از درون T-SQL language ، compose کرد، که به خروجی بدست آمده اجازه ذخیره شدن در یک جدول دایم یا موقت را می دهد. عملیات INSERT ، یک کپی واقعی از داده ها را ارائه می کند که احتمالاً روی performance تاثیر گذار خواهد بود.

اگر composability و استفاده مجدد از خروجی درون سرور یک نیاز باشند، TVF ها جایگزینهای بهتری هستند. اگر خروجی بدست آمده فقط نیاز به stream شدن به client یا middle-tier داشته باشند، هر دو روش منطقی است.

## منبع داده ها(Source of the Data)

منبع داده هایی که بازگردانده می شود، فاکتور مهم دیگری در انتخاب بین پیاده سازی بر اساس T-SQL یا CLR است. خروجی را میتوان هم از طریق خواندن source در local instance با استفاده از ADO.NET provider یا از طریق یک source خارج از SQL Server ، بدست آورد. در منابع خارجی، پیاده سازی بر پایه CLR، انتخاب بهتری از T-SQL است، به علت داشتن سهولتی که logic accessing the external data ، توسط آن انجام می شود.

در صورت بدست آوردن خروجی بر اساس یک query که با استفاده از ADO.NET provider روی local instance اجرا شده، یک stored procedure ، معمولاً یک query را اجرا می کند، از طریق خروجی تکرار می شود، و بعضی از عملیات ها را روی row ها، قبل از بازگرداندن از طریق یک SqlPipe ، انجام می دهد.

با یک TVF ، برنامه نویسان می توانند یک شی data reader را خوانده و در یک کلکسیون در حافظه بارگذاری کنند. اما، SQL Server 2005، به request ها اجازه نمی دهد تا زمانی که یک table-valued function باز می گردد، pending بمانند؛ هر query که از طریق ADO.NET provider اجرا شده است، باید کاملاً اجرا شوند و خروجی باید به طور کامل قبل از اینکه function body بتواند بازگردد، consume شوند. اگر عبارت بازگشت (return statement)، هنگامی که SqlDataReader operation ها در ADO.NET provider ، معلق (pending) است، اجرا شود، یک error ممکن است روی دهد. این بدین معناست که در اکثر مواردی که داده ها از local database instance بازگردانده می شوند، نمی توان داده ها را از طریق یک CLR TVF ، stream کرد. اگر فاکتورهای دیگر از قبیل composability ، مستلزم نوشته شدن به عنوان یک TVF باشد، شاید نوشتن آن در T-SQL تنها آپشن باشد. در غیر این صورت، استفاده از یک stored procedure مدیریت شده که از SqlPipe استفاده می کند، ممکن است.

در مواردی که در آنها خروجی قرار است از درون یک stored procedure بر اساس داده ها از local instance ، بدست آید، استفاده از API های SendResults فقط در مواردی معنا دارد که خروجی نیاز به مقداری اصلاح یا تغییر رویه ای (procedural modification or manipulation) دارد.

## عملیاتهایی (operation) با اثرات جانبی(Side-Effect)

بطور کلی، operationهایی که دارای side-effect هستندoperation -هایی که حالتdatabase ، از قبیل عباراتDML ، یاtransaction operation ها، را تغییر می دهند– ازfunction هایی (مثلاًfunction های-table valued)که تعریف کاربر است، منع می‌شوند؛ اما شاید اینoperation ها مطلوب باشند. مثلاً ممکن است کسی بخواهد یک transaction SAVEPOINT را set کند، یا یک operation اجرا کند، و در صورت روی دادنerror ، آن را بهSAVEPOINT ، rollbackکند.

با فرض اینکهfunction هایی که کاربر تعریف می کند، منع می شوند، چنین سناریویی فقط از طریق یک stored procedureقابل پیاده سازی است، و خروجی باید از طریق SqlPipe بازگردانده شوند. اما به یاد داشته باشید، بهoperation هایی باside-effect ، اجازه اجرا شدن از طریقADO.NET provider ، زمانیکه SqlPipe مشغول فرستادن خروجی است، داده نمی شود. اینoperation ها فقط قبل از اینکه خروجی شروع یا بعد از اینکه تکمیل شوند، مجاز هستند.

## Typingو تعداد خروجی

توصیف خروجی بدست آمده از طریق یک CLR stored procedure توسطSqlPipe ، با خروجی بدست آمده از یک CLR TVF متفاوت، و با همتایانشان در T-SQL یکسان هستند.

از سوی دیگر، یکstored procedure declaration ، هیچ عبارتی در مورد خروجی بدست آمده – یا حتی آیا آنها را بدست می آورد– نمی سازد. ممکن است آسان بنظر آید، و گرچه مسلماً انعطاف (flexibility) بیشتری ایجاد می کند، ولی باید در نوشتنapplication هایی کهstored procedure ها را اجرا می کنند دقت بیشتری کرد، زیرا ممکن است است به طور دینامیکی شکل خروجی را دوباره تعریف کند. اما، اگر لازم باشد که schema برای خروجی در میان فراخوانی ها invocation() متغیر باشند، باید از یک stored procedure استفاده کرد، زیرا فقط SqlPipe این انعطاف را در اختیار می گذارد.

در حقیقت، نوع داده های ضعیف خروجی ها تولید شده از طریق sqlpipe درونStored Procedure ها، فراتر ازSchema ی یک خروجی واحد است که احتمال بازگشت چندین خروجی متغیر را دربر می گیرد. همtype ها و هم تعداد خروجی را می توان توسط stored procedure به طور دینامیکی تعیین کرد.

## خلاصه

جدول زیر خلاصه ای از راهنماییهایی در مورد چگونگی تعیین اینکه آیا یک application ویژه باید در T-SQL یا CLR نوشته شود، و یا اینکه آیا یک stored procedure یا یک table-valued function باید استفاده شود، است.

| موقعیت | بله | خیر |
|---|---|---|
| آیا Composability نیاز است؟ | TVF | Procedure یا TVF |
| منبع خارجی داه ها (در مقابل accessing only local data) | CLR TVF یا CLR procedure | T-SQL TVF یا SQL procedure (accessing only local data) |
| آیا side-effect نیاز است؟ | Procedure | Procedure یا TVF |
| ثابت؟ results schema | Procedure یا TVF | Procedure |
| بیش از یک دسته خروجی | Procedure | Procedure یا TVF |
| آیا قادر به stream کردن خروجی است؟ | CLR TVF | T-SQL TVF |

در اکثرجاهای این بخش، ارسال خروجی از طریق SqlPipe ، به شدت با procedure ها پیوند خورده است. گرچه SqlPipe و امکان بازگرداندن خروجی در بدنه CLR trigger ها در دسترس هستند، اما این عمل به شدت نهی می شود، زیرا ممکن است به خروجی غیر منتظره ای برای issuing Data Manipulation Language یا عبارات Data Definition Language statements یا trigger هایی که روی اشیاء هدف تعریف شده، منجر شود.

## اجرای Aggregation های سفارشی روی داده ها

سناریوهایی وجود دارند که ممکن است aggregation نیاز به اجرا شدن روی data داشته باشد، که شامل اجرای محاسبات آماری از قبیل یافتن میانگین ها، انحراف از معیار (standard deviation) و غیره است. اگر aggregation function مورد نظر، در داخل ساخته نشود، راههای زیادی برای اضافه کردن (add) functionality در SQL Server 2005 وجود دارد:

- نوشتن aggregation به عنوان یک aggregation تعریف شده توسط کاربر. (UDA)
- نوشتن aggregation با استفاده از یک CLR stored procedure.
- استفاده از یک cursor جانبی سرور (server-side) در T-SQL.

بیایید این سه راه را در بافت (context) یک aggregation function ساده که محصول یک دسته از value های معین را محاسبه می کند، امتحان کنیم.

مثال: یک Product که به عنوان یک aggregate function تعریف شده توسط کاربر، پیاده سازی شده

در اینجا کدی برای این task آورده شده که به عنوان یک aggregate تعریف شده توسط کاربر نوشته شده . Logic محاسبه این محصول، وقتی که وارد می‌شود، در متد Accumulate() وجود دارد. متد Merge() ، تعریف می کند اگر این دو aggregate ادغام شوند، چه اتفاقی می افتد.

در:Visual Basic .NET

```
Imports System
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server


 _
 _
Public Structure ProductAgg

    Private product As SqlInt32

    Public Sub Init()
        product = 1
    End Sub

    Public Sub Accumulate(ByVal value As SqlInt32)
        product = product * value
    End Sub

    Public Sub Merge(ByVal group As ProductAgg)
        product = product * group.product
    End Sub

    Public Function Terminate() As SqlInt32
        Return product
    End Function

End Structure
```

در:C#

```
using System;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

[Serializable]
[SqlUserDefinedAggregate(Format.Native)]
public struct ProductAgg
{
    private SqlInt32 product;

    public void Init()
    {
        product = 1;
    }

    public void Accumulate(SqlInt32 value)
    {
        product = product * value;
```

```
    }

    public void Merge(ProductAgg group)
    {
        product = product * group.product;
    }

    public SqlInt32 Terminate()
    {
        return product;
    }
}
```

بعد از اینکه این type ، با SQL Server ساخته و ثبت شد، می توان آن را فقط به عنوان یک built-in aggregate از T-SQL استفاده کرد:

```
SELECT dbo.ProductAgg(intcol)
FROM tbl
GROUP BY col
```

## مثال Product :، به عنوان یک Stored Procedure مدیریت شده

یک stored procedure را می توان ایجاد کرد که روی data تکرار می شود تا محاسبه (computation) را اجرا کند. این تکرار با استفاده از کلاس SqlDataReader بدست می آید، همانطور که در پایین نشان داده شده.

### در Visual Basic .NET:

```
Imports Microsoft.SqlServer.Server
Imports System.Data.SqlTypes
Imports System.Data.SqlClient

Partial Public Class StoredProcedures

    _
    Public Shared Sub VBProductProc(ByRef value As SqlInt32)

        ' The empty product is 1
        value = 1

        Using conn As New SqlConnection("context connection = true")
            conn.Open()
            Dim cmd As SqlCommand = New SqlCommand()
            cmd.Connection = conn
            cmd.CommandText = "SELECT intcolumn FROM tbl"
            Dim r As SqlDataReader = cmd.ExecuteReader()
            Using r
                Do While r.Read()
```

```
                    value = value * r.GetSqlInt32(0)
                Loop
            End Using
            conn.Close()
        End Using
    End Sub

End Class
```

در:C#

```
using Microsoft.SqlServer.Server;
using System.Data.SqlTypes;
using System.Data.SqlClient;
public partial class StoredProcedures
{
    [SqlProcedure]
    public static void ProductProc(out SqlInt32 value)
    {
        // Ensure that we write to value.
        // Empty product is 1.
        value = 1;

        using (SqlConnection conn =
            new SqlConnection("context connection = true"))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = "SELECT intcolumn FROM tbl";
            SqlDataReader r = cmd.ExecuteReader();
            using (r)
            {
                while (r.Read()) //skip to the next row
                {
                    value *= r.GetSqlInt32(0);
                }
            }
        }
    }
}
```

می توان این را با استفاده از عبارت T-SQL EXEC فرا خواند:

```
EXEC Product @p OUTPU
```

مثال Product : به عنوان یک T-SQL Stored Procedure که از یک Cursor استفاده می کند

نهایتاً، می توان یک T-SQL stored procedure ایجاد کرد که یک query را اجرا و محاسبه را با استفاده از یک T-SQL cursor انجام می دهد تا data را تکرار کند.

T-SQL:

```
create procedure TSQL_ProductProc (@product int output)
as
begin
    declare @sales int
    declare c insensitive cursor for select intcolumn from tbl

    set @product = 1

    open c
    fetch next from c into @sales

    while @@FETCH_STATUS = 0
    begin
        set @product = @product * @sales
        fetch next from c into @sales
    end

   close c
   deallocate c
end
go
```

## خلاصه

تصمیم گیری در مورد استفاده از یک UDA یا یکی از راه حلهای دیگر برای بدست آوردن خروجی به چند عامل بستگی دارد، که شامل composability requirements ، مشخصه های aggregation algorithm ، و نیاز به side-effect می شود.

در واقع، یک UDA ، یک شی مستقل است که از هر T-SQL query ، قابل استفاده است، عموماً در همان مکانی که یک system aggregate استفاده می شود. هیچ تصوری در مورد یک query که روی آن عمل می کند وجود ندارد. مثلاً، می توان آن را در view definition ها (اما نه در indexed view ها) و در sub-query های ترازویی (scalar) به شمار آورد.

می توان UDA ها را قبل از کلمه ORDER BY در یک query ارزیابی کرد، پس هیچ تضمینی برای ترتیبی که value ها به aggregation function ارائه می شوند وجود ندارد. بنابراین، اگر aggregation algorithm باید value ها را به ترتیب خاصی consume کند، نمی توان از یک UDA استفاده کرد. یک UDA نیز دقیقاً value های کل یک گروه را consume می کند و یک value واحد باز می گرداند. اگر مشکل حل نشد، آنگاه باید از یک تکنیک دیگر استفاده کرد.

همچنین یک UDA ، هیچ data access را اجرا نمی کند و هیچ side-effect ندارد؛ اگر هر یک از اینها لازم باشند، آنگاه باید از یک stored procedure استفاده کرد.

گرچهUDA ها دارای محدودیتهایی هستند، اما احتمال دارد بهترین performance را از آپشنهای ارائه شده انجام دهند، پس aggregation عموماً باید از طریق یک UDA اجرا شود، مگر اینکه الزامات (requirement) دیگر مانع آن شوند.

## Typeهای تعریف شده توسط کاربر

حالا به یکی از قدرتمندترین ویژگیهای SQL Server 2005 می رسیم که اغلب مورد سوء تفاهم قرار می گیرد. باType های تعریف شده توسط کاربرUDT) ها)، می توان scalar type system را درdatabase ، گسترش داد. و این، فراتر از تنها تعریف کردن alias برای یک system type است که در نسخه قبلی SQL Server دردسترس بود. تعریف یکUDT ، به سادگی نوشتن یک کلاس در کد مدیریت شده، ایجاد یک اسمبلی، و سپس (register)ثبت کردن type در SQL Server با استفاده از عبارت CREATE TYPE است. مثال زیر، اسکلت یک کد است که شکل ساختاری یک UDT را نشان می دهد.

در:Visual Basic .NET

```
Public Structure SimpleType
    Implements INullable

    Public Overrides Function ToString() As String
        ...
    End Function

    Public ReadOnly Property IsNull() As Boolean Implements _
        INullable.IsNull
        ...
    End Property

    Public Shared ReadOnly Property Null() As SimpleType
        ...
    End Property

    Public Shared Function Parse(ByVal s As SqlString) As SimpleType
        ...
    End Function

End Structure
```

در:C#

```
 [Serializable]
[Microsoft.SqlServer.Server.SqlUserDefinedType(Format.Native)]
public struct SimpleUdt : INullable
{
    public override string ToString() { ...  }
    public bool IsNull { ...  }
    public static SimpleUdt Null
    {
        get { ...  }
    }
    public static SimpleUdt Parse(SqlString s) { ...  }
```

```
}
```

می توان این را در T-SQL بارگذاری (load) و استفاده کرد:

```
CREATE TYPE simpleudt FROM [myassembly].[SimpleUdt]
CREATE TABLE t (mycolumn simpleudt)
```

## کی یک UDT ایجاد کنیم؟

UDTها در SQL Server 2005 ، یک مکانیزم توسعه پذیر object-relational نیستند؛ بلکه راهی برای گسترش scalar type system در database هستند scalar type system. شامل typeهای ستونی (columnar) است که با SQL Server (int، nvarchar، uniqueidentifier، و غیره) منتقل و استفاده می شوند. با UDT ها، می توان یک type جدید تعریف کرد که میتوان به جای یک built-in scalar type استفاده کرد. اگر type، یک value اتمی است که برای مدل شدن به عنوان یک ستون مناسب است، یک UDT ایجاد کنید.

کاندیدهای خوب جهت پیاده سازی یک UDT، شامل تاریخ custom و type های time data در تقویم های مختلف و type های currency data است. یک UDT، یک کلاس واحد است که در معرض همه رفتارهای (behavior) دردسترس روی type است و داده های بنیادی ذخیره شده توسط type را خلاصه می کند: کل data access از programmatic interface در UDT استفاده می کنند. اغلب می توان از functionality موجود در .NET framework — از قبیل بین المللی کردن (internationalization) یا — calendar functionality استفاده کرد تا functionality را که به سختی فراهم می شود، فراهم کرد.

## کی یک UDT ایجاد نکنیم؟

از یک UDT نباید برای مدل سازی اشیای business از قبیل کارمندان، قراردادها، یا مشتری ها استفاده کرد. SQL Server با یک UDT به عنوان یک واحد (unit) رفتار می کند که به آن opaque است. بعضی از مسائل با UDT های پیچیده، شامل محدودیت ۸ کیلو بایتی برای type ها، محدودیتهای indexing ، و این واقعیت است که کل value باید هنگامی که هر value در UDT ، update می شود، update شود.

## فاکتورهایی که هنگام طراحی یک UDT باید در نظر گرفت

از آنجاییکه UDT ها ستونی هستند، می توان index ها را، روی کل value های UDT تعریف کرد؛ همانطور که referential integrity constraint، از قبیل unique بودن، را می توان. همچنین می توان از UDT ها در مقایسه و مرتب کردن سناریوها استفاده کرد.

مقایسه Value های UDT ، بوسیله مقایسه binary representation of the type انجام می شود. اگر از Format.Native به عنوان مکانیزم پایداری (persistence mechanism) استفاده شود، آنگاه یک

persisted form با استفاده از همان ترتیب field ، به عنوان آنچه که در type تعریف می شود، ایجاد می شود؛ پس باید دقت کرد که آنها در جای صحیح قرار می گیرند.

غیر از مقایسه، هر عملیاتی روی یک UDT ، نیازمند این است که UDT value ، de-serialize شود و یک متد، فراخوانده شود. این الگو (pattern) موجب تحمیل بار اضافی روی server می شود، که باید هنگام تصمیم گیری در مورد اینکه آیا آن را به عنوان یک UDT ، مدلسازی کنیم یا نکنیم، مد نظر قرار بگیرد UDT . ها زمانی که یک type،که نیاز به model شدن دارد، رفتارهای پیچیده ای دارد، مفیدتر هستند. اگر type نسبتاً ساده باشد، آنگاه شاید بهتر باشد از ساختار UDT پرهیز کنیم.

نهایتاً، می توان از متدهای static یک UDT به عنوان یک مکانیزم بسته بندی (packaging) راحت جهت ذخیره سازی library تابعهای مربوط استفاده کرد. متدهای static را می توان از T-SQL با استفاده از syntaz زیر فراخوانی کرد:

```
select Type::Function(@arg1)
```

### مثال: تاریخ های غیر غربی

می خواهیم value های زمان و تاریخ را با استفاده از تقویم "ام القرا" که با تقویم گریگوری که SQL Server از آن استفاده می کند، ذخیره کنیم. می خواهیم این data type همان رفتارهای پایه ای datetime data type را داشته باشد، بویژه string conversion ، قابلیت بازیابی اجزای data ، و محاسبات تاریخ و غیره.

مثال زیر از یک type تعریف شده توسط کاربر، یک پیاده سازی ساده از این data type است، و از UmAlQuraCalendar type استفاده می کند، که در نسخه ۲/۰ .NET Framework جدید است. متوجه شدن این مثال کمک زیادی به فراهم کردن متدهای لازم می کند.

UDT ام القرا در C#

```
using System;
using System.Data;
using System.Data.Sql;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.Globalization;

[Serializable]
[SqlUserDefinedType(Format.Native, IsByteOrdered = true)]
public struct UmAlQuraDateTime : INullable
{

    /*
     * Private state.
     */

    private long dtTicks;
    private bool isNull;
```

```
// Calendar object used for all calendar-specific operations
private static readonly UmAlQuraCalendar s_calendar =
    new UmAlQuraCalendar();

// For correct formatting we need to provie a culture code for
// a country that uses the Um Al Qura calendar: Saudi Arabia.
private static readonly CultureInfo ci =
    new CultureInfo("ar-SA", false);


/*
 * Null-Handling
 */

// get a null instance
public static UmAlQuraDateTime Null
{
    get
    {
        UmAlQuraDateTime dt = new UmAlQuraDateTime();
        dt.isNull = true;
        return dt;
    }
}

public bool IsNull
{
    get
    {
        return this.isNull;
    }
}

/*
 * Constructors
 */

public UmAlQuraDateTime(long ticks)
{
    isNull = false;
    dtTicks = ticks;
}

public UmAlQuraDateTime(DateTime time)
    : this(time.Ticks)
{
}

/*
 * Factory routines.
 */

public static UmAlQuraDateTime Parse(SqlString s)
{
    if (s.IsNull) return Null;
    DateTime t = DateTime.Parse(s.Value);
```

```
        return new UmAlQuraDateTime(t);
    }

    public static UmAlQuraDateTime ParseArabic(SqlString s)
    {
        if (s.IsNull) return Null;
        DateTime t = DateTime.Parse(s.Value, ci);
        return new UmAlQuraDateTime(t);
    }

    public static UmAlQuraDateTime FromSqlDateTime(SqlDateTime d)
    {
        if (d.IsNull) return Null;
        return new UmAlQuraDateTime(d.Value);
    }

    public static UmAlQuraDateTime Now
    {
        get
        {
            return new UmAlQuraDateTime(DateTime.Now);
        }
    }

    /*
     * Conversion Routines
     */

    public DateTime DateTime
    {
        get { return new DateTime(this.dtTicks); }
    }

    public SqlDateTime ToSqlDateTime()
    {
        return new SqlDateTime(this.DateTime);
    }

    public override String ToString()
    {
        return this.DateTime.ToString(ci);
    }

    public String ToStringUsingFormat(String format)
    {
        return this.DateTime.ToString(format, ci);
    }

    /*
     * Methods for getting date parts.
     */

    public int Year
    {
        get
        {
```

```
            return s_calendar.GetYear(this.DateTime);
        }
    }

    public int Month
    {
        get
        {
            return s_calendar.GetMonth(this.DateTime);
        }
    }

    public int Day
    {
        get
        {
            return s_calendar.GetDayOfMonth(this.DateTime);
        }
    }

    /*
     * Date arithmetic methods.
     */

    public UmAlQuraDateTime AddYears(int years)
    {
        return new
            UmAlQuraDateTime(s_calendar.AddYears(this.DateTime, years));
    }

    public UmAlQuraDateTime AddDays(int days)
    {
        return new
            UmAlQuraDateTime(s_calendar.AddDays(this.DateTime, days));
    }

    public double DiffDays(UmAlQuraDateTime other)
    {
        TimeSpan diff = DateTime.Subtract(other.DateTime);
        return diff.Days;
    }
}
```

وقتی این type در SQL Server بارگذاری می شود، می توان این type از طریق T-SQL استفاده کرد. در زیر چند مثال T-SQL آورده شده اند که از این UDT و خروجیی که بدست می آورند، استفاده می کنند.

ابتدا یک تاریخ ام القرا را parse و سپس آن را در دو فرمت با معادلهای غربی چاپ می کنیم:

```
DECLARE @d UmAlQuraDateTime
SET @d = UmAlQuraDateTime::ParseArabic('01/02/1400')
```

```
PRINT @d.ToString()
PRINT @d.ToStringUsingFormat('F')
PRINT @d.ToSqlDateTime()
```

نتیجه زیر حاصل می آید:

```
01/02/00 12:00:00 ص

01/رمضان/1400 12:00:00 ص

Dec 20 1979 12:00AM
```

می توانیم تاریخ های غربی را نیز به ام القرا تبدیل کنیم:

```
DECLARE @n DateTime
SET @n = 'March 20, 2005'
DECLARE @d UmAlQuraDateTime
SET @d = UmAlQuraDateTime::FromSqlDateTime(@n)
PRINT @n
PRINT @d.ToString()
```

**نتیجه:**

```
Mar 20 2005 12:00AM

10/02/26 12:00:00 ص
```

نهایتاً، بااین type می توانیم جداولی را با ستون، ایجاد و اصلاح کنیم:

```
CREATE TABLE dates (
   western DateTime,
   umalqura UmAlQuraDateTime
)

INSERT INTO dates(western) VALUES ('June 1, 2005')
INSERT INTO dates(western) VALUES ('July 1, 2005')

UPDATE dates
SET umalqura = UmAlQuraDateTime::FromSqlDateTime(dates.western)

SELECT western, umalqura.ToString() as umalqura FROM dates
```

این جدول، نتیجه نهایی است:

| western | umalqura |
| --- | --- |
| 2005-06-01 00:00:00.000 | 24/04/26 12:00:00 ص |
| 2005-07-01 00:00:00.000 | 25/05/26 12:00:00 ص |

## نتیجه گیری

این مقاله، راهنماییها، سناریوهای استفاده واقعی، و نمونه هایی را با استفاده از ویژگیهای CLR integration در SQL Server 2005 ، ارائه کرده است. برنامه نویسان و معماران (Database application developers and architects) باید از این مقاله جهت ترکیب با documentation در ویژگیهای دیگر SQL Server 2005 ، از قبیل Transact-SQL ، XML، و Service Broker استفاده کنند.