International Journal of Allied Practice, Research and Review
Website: www.ijaprr.com (ISSN 2350-1294)

CLR Integration in SQL Server 2008

Iqra jan
M.Tech Student
CSE Department,Kurukshetra University,India

***Abstract*– SQL Server 2008 gives us flexibility to implement programming logic by using different programming constructs like loops, conditional constructs etc. But there are certain situations where we require complex programming which is very difficult to implement in SQL Server .To handle such situations there should be some feature to create such a complex code in any .NET supported language and then embed that code in SQL Server. This particular feature is called CLR Integration. There is no support for CLR Integration in earlier versions of SQL Server. This feature is supported in SQL Server 2005 version onwards.2005 version onwards ,SQL server integrates CLR to allow execution of managed code within the SQL Server environment. This provides flexibility in writing the database code in multiple languages supported by .NET. Managed code also takes advantage of programming logic to implement complex programming logic in database objects, such as stored procedures, triggers, user-defined functions etc .This paper explains how to implement triggers, user defined functions, stored procedures and user defined types by using managed code.**

Keywords*– *CLR Integration; T-SQL; Managed code; SQL Server; Assembly

I. Introduction

CLR (Common Language Runtime) is one of the most essential components of .NET framework. It provides an environment for the application to run. It provides functionalities such as exception handling, security, debugging and versioning support to the applications. Important features provided by CLR are automatic memory management, standard type system, language interoperatibility, platform independence and security management.CLR can host a variety of languages. It offers a common set of tools across these languages, ensuring interoperatibility between the codes. The code developed with a language compiler that targets CLR is called a managed code.CLR is an environment that executes codes written in any .NET programming language.CLR integration allows database developers to create objects in any of the .NET supported languages (e.g., C#, J#, VB etc) and embeds the objects in the database .Such a database object is called a managed database object. In SQL Server 2008, we can create database objects like procedures, triggers, user-defined functions etc to implement programming logic using T-SQL. However in some situations, it is not possible to implement the required functionality by using T-SQL code e.g., if we want to get the domain of user's email id we need to apply various string functions which is very complex process to do using T-SQL code. With CLR integration in SQL Server 2008,we can create programs in any of the .NET supported languages. We can embed these programs in our database so that they can run in the same environment in which the database exists.Microsoft.Sql.Server.Server namespace includes core functionality for CLR programming.

II. Need for Managed Database Objects

We need managed database objects to implement complicated programming logic for which we can reuse the functionality provided by the .NET base class libraries, to access external resources such as calling a web service or accessing the file system, to implement a CPU-intensive functionality that run more efficiently as compared to the managed code. For example, we need to extract data from a database table and store the result set in a file stored in the local file system in the XML format. We will not be able to implement this functionality by using T-SQL. Instead; we can create a managed database object by using a .NET language to store the data in an XML file.

III. Implementing CLR Integration

To implement CLR integration in SQL Server 2008, we have to follow the following steps:

Step 1: We have to first create a managed code in any of the .NET programming languages for creating database objects in SQL Server. Managed code contains classes and methods that provide a desired functionality.

Step 2: After creating a managed code we have to create an assembly.SQL Server cannot directly execute the assemblies. Therefore before using assemblies, we need to import and configure the assemblies in the database engine. Whenever we import an assembly, its details are added to the sys.assemblies.system table. We can import a .NET assembly in SQL Server database engine using

CREATE ASSEMBLY statement. The syntax is

CREATE ASSEMBLY assembly_name

[AUTHORIZATION owner_name]

FROM {<client_assembly_specifier>|<assembly_bits>[,……n]}

[WITH PERMISSION_SET={SAFE|EXTERNAL_ACCESS|UNSAFE}]

Where,

assembly_name is the name of the assembly that we want to create in SQL Server.

AUTHORIZATION owner_name specifies the name of user or role as the owner of the assembly.

Client_assembly_specifier specifies the local or network path of the .NET assembly that is being imported.

PERMISSION_SET specifies the permissions that are granted to the assembly when it is accessed by SQL Server.This can accept any of the following values:

a) SAFE:This permission makes a code unable to access any external resource like files,networks,environment variables, or registry. If no value is specified, the default value is SAFE.

b) EXTERNAL_ACCESS:Enables .NET code to access external resources.

c) UNSAFE:Enables .NET code to access any resource within or outside SQL Server.

Step 3: After importing assemblies, we can create managed database objects that use the managed code provided in the assembly. By default, SQL Server does not allow the running of managed code on the server. So, we need to enable the CLR integration feature in our database before creating a managed database object. We use following statement to enable CLR integration feature

```
Sp_configure  CLR_ENABLED, 1;

GO

RECONFIGURE;

GO
```

While developing managed database objects, we will have to use the System.Data.SqlClient,System.Data.SqlTypes, and Microsoft.SqlServer.Server namespaces found in .NET base class libraries.

Depending on the requirements, the database developer can create the following types of objects:

1. Triggers.

2. User Defined Functions.

3. User-Defined Types.

4. Stored Procedures

Creating Managed Trigger

Managed triggers are used for implementing advanced trigger logic that cannot be done using T-SQL. For creating managed triggers we have to follow the following steps:

(a) Create a .NET class that implements the functionality of trigger. Then compile that class to produce a .NET assembly.

(b) Register that assembly in SQL Server using CREATE ASSEMBLY statement.

(c) Create a trigger and associate it with the actual methods of the assembly. Syntax for creating trigger is:

```
CREATE TRIGGER <Trigger Name>
ON <Table or View><FOR|INSTEAD OF|AFTER>
<INSERT|UPDATE|DELETE>
AS EXTERNAL NAME <Assembly Identifier>.<Type Name>.<Method Name>
```

Where,
<Trigger Name> is the name of a trigger.
<Table or View> is the name of a table or view on which we want to create a trigger.
<FOR|INSTEAD OF|AFTER>is the type of a trigger we want to create.
<Assembly Identifier> is the name of imported assembly.
<Type Name>is the name of a class that contains the method that will be executed through the procedure.
<Method name> is the name of method that will be executed through the procedure.

Creating managed User-Defined Functions

To create managed functions we have to follow the following steps:

(a) Create a .NET class that implements the functionality of the user-defined function. Then, compile that class to produce a .NET assembly.

(b) Register that assembly in SQL Server using CREATE ASSEMBLY statement.

(c) Create user-defined function and associate it with the actual methods of the assembly. Syntax for creating managed user-defined function using an imported assembly is:

```
CREATE FUNCTION <Function Name>
(
<Parameter List>
)
RETURNS <Return Type>
AS EXTERNAL NAME <Assembly Identifier>.<Type Name>.<Method Name>
```

Where,
<Parameter List> is the list of parameters that must be passed to function.
<Return Type> specifies value returned by function.
<Assembly Identifier> is the name of imported assembly.
<Type Name>is the name of class that contains the method that will be executed through procedure.
<Method Name> is the name of method that will be executed through procedure.

Creating User-Defined Types

We can create a data type definition in any of the .NET supported languages and use it as a data type within SQL Server. This data type can be a combination of any other existing data type with more modifications applied on it. We have to perform the following steps to create user-defined data type:

(a) Create a .NET class that implements the functionality of the user-defined type. Then, compile that class to produce a .NET assembly.

(b) Register that assembly in SQL Server using the CREATE ASSEMBLY statement.

(c) Create a user-defined data type that refers to the registered assembly.Syntax for creating User defined types is:

```
CREATE TYPE [schema_name. ] type_name
{
EXTERNAL NAME assembly_name [.class_name]
}
```

Where,
Schema_name is the name of the schema to which the alias data type or user-defined type.
Type_name is the name of the alias data type or user-defined type.
Assembly_name specifies the SQL Server assembly that references the implementation of the user-defined type in CLR.
[.class_name] specifies the class within assembly that implements the user-defined type.

Creating Stored Procedures

With CLR integration, we can use managed code to be executed as a stored procedure. For this purpose, we have to create a procedure that refers to an imported assembly. To create a stored procedure using the managed code, we have to perform the following steps:

(a) Create a .NET class that implements the functionality of the stored procedure. Then, compile that class to produce a .NET assembly.

(b) Register that assembly in SQL Server using the CREATE ASSEMBLY statement.

(c) Create a stored procedure and associate the stored procedure with the actual methods of the assembly. Syntax for creating stored procedure using managed code:

```
CREATE PROCEDURE <Procedure Name>
AS EXTERNAL NAME <Assembly Identifier>.<Type Name>.<Method Name>
```

Where,
<Procedure Name >is the name of the procedure we want to create.
<Assembly Identifier> is the name of the imported assembly.
<Type Name> is the name of the class that contains the method that will be executed through the procedure.
<Method Name> is the name of the method that will be executed through the procedure.

IV. Advantages of using Managed code over T-SQL

Transact-SQL is specifically designed for direct data access and manipulation in the database. While Transact-SQL excels at data access and management, it is not a full-fledged programming language. For example, Transact-SQL does not support arrays, collections, for-each loops, bit shifting, or classes. While some of these constructs can be simulated in Transact-SQL, managed code has integrated support for these constructs. Depending on the scenario, these features can provide a compelling reason to implement certain database functionality in managed code.

.NET supported languages offer object-oriented capabilities such as encapsulation, inheritance, and polymorphism. Using these capabilities related code can now be easily organized into classes and namespaces. When we are working with large amounts of server code, this allows us to more easily organize and maintain our code.

Managed code is better suited than Transact-SQL for calculations and complicated execution logic, and features extensive support for many complex tasks, including string handling and regular expressions. With the functionality found in the .NET Framework Library, we have access to thousands of pre-built classes and routines. These can be easily accessed from any stored procedure, trigger or user defined function. The Base Class Library (BCL) includes classes that provide functionality for string manipulation, advanced math operations, file access, cryptography, and more.
One of the benefits of managed code is type safety, or the assurance that code accesses types only in well-defined, permissible ways. Before managed code is executed, the CLR verifies that the code is safe. For example, the code is checked to ensure that no memory is read that has not previously been written. The CLR can also help ensure that code does not manipulate unmanaged memory.CLR integration offers the potential for improved performance.

Another important factor in our decision about whether to use Transact-SQL or managed code is where we would like our code to reside, the server computer or the client computer. Both Transact-SQL and managed code can be run on the server. This places code and data close together, and allows us to take advantage of the processing power of the server. On the other hand, you may wish to avoid placing processor intensive tasks on our database server. Most client computers today are very powerful, and we may wish to take advantage of this processing power by placing as much code as possible on the client. Managed code can run on a client computer, while Transact-SQL cannot.

V. Conclusion

SQL Server 2008 is a database engine that provides a platform to build and manage data applications; it allows secure and efficient storage and management of data. In addition, it combines data analysis,reporting,integration and notification services that enable organizations to build and deploy efficient Business Intelligence Solutions .This paper explains one of the integration services offered by SQL Server 2008 that makes it more efficient to implement complex programming logic.

ACKNOWLEDGMENT

I would like to acknowledge all the people who have helped me to write this paper. I also would like to acknowledge my teachers especially Mr. Naseer Ganai and my friends who supported me to write this paper and put my ideas on paper as a contribution to coming generation.

REFERENCES

[1] Robert Vieira “Beginning Microsoft SQL Server 2008 Programming”.

[2] Erik Veerman “Microsoft SQL Server 2008 Integration Services: Problem, Design, Solution”.

[3] Brian Knight “Knight’s 24-Hour Trainer: Microsoft SQL Server 2008 Integration Services”.

[4] Paul Nielsen “Microsoft SQL Server 2008 Bible”.

[5] Robert Vieira “Professional Microsoft SQL Server 2008 Programming”.

[6] Paul Turley “Beginning T-SQL with Microsoft SQL Server 2005 and 2008”.

[7] Erik Veerman “Professional Microsoft SQL Server 2008 Integration Services”.

[8] Christian Nagel, Bill Enjen, Jay Glynn, Morgan Skinner, Karli Watson “Professional C# 2008”, Wiley Publications

[9] Micheal Otey “Microsoft SQL server 2005 New features” .