Laboratorium 8 – MS SQL Server 2008

Temat: Środowisko CLR w MS SQL Server - UDP i UDF

Opracowanie: A.Dydejczyk

Ćwiczenia do opracowania w trakcie laboratorium:

1. Zarządzanie i bezpieczeństwo w CLR SQL
2. Ćwiczenie A. Procedury składowane CLR
3. Ćwiczenie B. Funkcje UDF CLR w MS SQL Server
4. Ćwiczenie C. Wyzwalacze CLR w MS SQL Server
5. Zadania.

1. Zarządzanie i bezpieczeństwo CLR SQL

Bezpieczeństwo obiektów CLR tworzonych w bazie MS SQL Server.

- CLR Integration Code Access Security
  https://docs.microsoft.com/en-us/sql/relational-databases/clr-integration/security/clr-integration-code-access-security?view=sql-server-2017

- Host Protection Attributes and CLR Integration Programming
  https://docs.microsoft.com/en-us/sql/relational-databases/clr-integration-security-host-protection-attributes/host-protection-attributes-and-clr-integration-programming?view=sql-server-2017

- SQLCLR Security and Designing for Reuse
  https://www.codemag.com/article/0705051/SQLCLR-Security-and-Designing-for-Reuse

Klauzula WITH PERMISSION_SET w ramach tworzenia „assembly" umożliwia nadanie odpowiednich uprawnień (*Code Access Security CAS*).
Możliwe uprawnienia dla kodu CLR.

- Uprawnienie **SAFE** jest najbardziej restrykcyjne i uniemożliwia „*assembly*" dostęp do zasobów systemowych poza serwerem SQL Server. SAFE jest wartością domyślną.
- Uprawnienie **EXTERNAL_ACCESS** umożliwia „*assembly*" dostęp do niektórych zasobów zewnętrznych, takich jak pliki, sieć, rejestr i zmienne środowiska.
- Uprawnienie **UNSAFE** pozwala „*assembly*" na nielimitowany dostęp do danych zewnętrznych jak również na wykonanie kodu niezarządzanego.

Nadanie uprawnień EXTERNAL_ACCESS i UNSAFE wymaga ustawienia parametru TRUSTWORTHY na ON dla bazy danych w której zostaną dodane „assembly" z wymienionymi uprawnieniami.

Informacje związane z „assembly" składowane obiektach dostępnych w bazie danych.

```
select * from sys.assemblies
select * from sys.Assembly_Files
select * from Sys.Assembly_Modules
select * from Sys.Assembly_References
select * from Sys.Module_Assembly_Usages
```

System View	Stored Information
Sys.Assemblies	Assembly Name, CLR Name, .NET Assembly Version, Permissions, IsVisible Flag, Create_Date, ModifyDate
Sys.Assembly_Files	Original Load Location, Content
Sys.Assembly_Modules	Implemented Class and Method Names
Sys.Assembly_References	Relationship References between Assemblies
Sys.Module_Assembly_Usages	Links from SQL Objects to Assemblies

W trakcie tworzenia „assembly" w IDE Visual Studio metodą auto-deploy realizowane są następujące zadania:

- sprawdzane jest istnienie „assembly" a następnie następuje jego usunięcie,
- tworzone jest „assembly" i wstawianie pliku DLL do metadanych SQL,
- tworzony jest prototyp funkcji T-SQL umożliwiający dostęp do DLL,
- wykonywane są polecenia ALTER dla wszystkich obiektów zawierających informację o „assembly".

Tworzenie „assembly" ręcznie.

```
CREATE ASSEMBLY <assembly name>
[ AUTHORIZATION <owner name> ]
FROM { <client assembly specifier> | <assembly bits> [ ,...n ] }
[ WITH PERMISSION_SET = { SAFE | EXTERNAL_ACCESS | UNSAFE } ]
[ ; ]
```

Tworzenie procedury składowanej CLR na podstawie „assembly".

```
CREATE PROCEDURE|PROC <sproc name>
[<parameter name> [<schema>.]<data type> [VARYING] [= <default value>] [OUT[PUT]]
[,<parameter name> [<schema>.]<data type> [VARYING] [= <default value>][OUT[PUT]]
[,...]]
[WITH RECOMPILE| ENCRYPTION ]
[EXECUTE AS { CALLER|SELF|OWNER|<'user name'>}]
[FOR REPLICATION]
AS
<code> | EXTERNAL NAME <assembly name>.<class name>.<method name>
```

Ćwiczenie A. Procedury składowane CLR

Link:
https://msdn.microsoft.com/pl-pl/library/system.data.sqlclient.sqlcommand.executescalar%28v=vs.110%29.aspx

1. Zadanie można zrealizować z wykorzystaniem Visual Studio (kompilacja i następnie *deploy*) lub ręcznie tworząc „*assembly*" i odpowiednie procedury składowane poleceniem T-SQL. W ramach skryptu są udostępnione cztery procedury składowane. Nazwy procedur są modyfikowane przez parametr „*Name*" w ramach każdej procedury.

```
// Skrypt Lab08.01
using System;
using System.Data;
using Microsoft.SqlServer.Server;
using System.Data.SqlClient;
using System.Data.SqlTypes;

public partial class StoredProcedures
{
  [Microsoft.SqlServer.Server.SqlProcedure(Name = "usp_HelloWorld")]
  public static void usp_Proc1()
  {
     SqlContext.Pipe.Send("HELLO WORLD FROM SQL CLR");
  }
  [Microsoft.SqlServer.Server.SqlProcedure(Name="usp_ProcParams")]
  public static void usp_Proc2(SqlInt32 iCnt, SqlString sName)
  {
     SqlContext.Pipe.Send("FirstParm:" + iCnt.ToString());
     SqlContext.Pipe.Send("SecondParm:" + sName.ToString());
  }
  [Microsoft.SqlServer.Server.SqlProcedure(Name="usp_ProcOutParm")]
  public static void usp_Proc3(SqlInt32 iCnt, out SqlString sName)
  {
     sName = "Tester";
     SqlContext.Pipe.Send("FirstParm:" + iCnt.ToString());
  }
  [Microsoft.SqlServer.Server.SqlProcedure(Name="usp_GetSalesPersonCount")]
  public static int usp_Proc4()
  {
     int iRows;
     SqlCommand sqlCmd = new SqlCommand() ;
     sqlCmd.Connection = new SqlConnection("Context connection=true");
     sqlCmd.Connection.Open();
     sqlCmd.CommandText =  "SELECT COUNT(*) as 'Sales Person Count'"
                    + "FROM AdventureWorks.Sales.SalesPerson" ;
     iRows = (int) sqlCmd.ExecuteScalar();
     return iRows ;
  }
};
```

Sprawdzamy poprawność przygotowanych procedur.

```
exec usp_HelloWorld
```

```
exec usp_ProcParams 10, 'Test'

declare @out nvarchar(40)
exec usp_ProcOutParm 10, @out output
print @out

declare @count int
exec @count = dbo.usp_GetSalesPersonCount
print @count
```

2. W następnym zadaniu opracujemy procedurę składowaną, która odczyta parametry środowiska oraz prześle wykorzystując metodę ***Send(SqlRecord rekord)***. Realizacja zadania wymaga ustawienia parametru **TRUSTWORTHY** w bazie „testCLR" oraz uprawnień „assembly" do wartości „**EXTERNAL_ACCESS**".

```
// Skrypt Lab08.02
using System;
using System.Data;
using System.Collections;
using Microsoft.SqlServer.Server;
using System.Data.SqlClient;
using System.Data.SqlTypes;
public partial class StoredProcedures
{
  [Microsoft.SqlServer.Server.SqlProcedure(Name="usp_GetEnvVars")]
  public static void usp_Proc5()
  {
    try
    {
       SortedList environment_list = new SortedList();
       foreach (DictionaryEntry de in Environment.GetEnvironmentVariables())
       {
         environment_list[de.Key] = de.Value;
       }
       SqlDataRecord record = new SqlDataRecord (
         new SqlMetaData("VarName", SqlDbType.NVarChar, 1024),
         new SqlMetaData("VarValue", SqlDbType.NVarChar, 4000)
       );
       SqlContext.Pipe.SendResultsStart(record);
       foreach (DictionaryEntry de in environment_list)
       {
         record.SetValue(0, de.Key);
         record.SetValue(1, de.Value);
         SqlContext.Pipe.SendResultsRow(record);
       }
       SqlContext.Pipe.SendResultsEnd();
    }
    catch (Exception ex)
    {
       SqlContext.Pipe.Send(ex.Message);
    }
  }
};
```

Sprawdzamy poprawność przygotowanej procedury.

```
exec usp_GetEnvVars
```

Ćwiczenie B. Funkcje UDF CLR w MS SQL Server

Przydatne informacje dotyczące tworzenia i uruchamiania skalarnych i tabelarycznych funkcji UDF z wykorzystaniem technologii CLR znajdują się w poniższym linku.

https://msdn.microsoft.com/en-us/library/ms131077%28v=sql.105%29.aspx

Polecenie umożliwiające utworzenie funkcji T-SQL również na podstawie „assembly".

```
CREATE FUNCTION [<schema name>.]<function name>
([<@parameter name> [AS] [<schema name>.]<data type>
[ = <default value>] [READONLY]
[ ,...n ] ] )
RETURNS {<type>|TABLE [(<table definition>)]}
[ WITH [ENCRYPTION] | [SCHEMABINDING] |
  [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ] |
  [EXECUTE AS {CALLER|SELF|OWNER|<'user name'>} ]
]
[AS]  { EXTERNAL NAME <external method> |
BEGIN
[<function statements>]
{RETURN <type as defined in RETURNS clause>|RETURN (<SELECT statement>)}
END }[;]
```

- CALLED ON NULL INPUT - (wartość domyślnie) specyfikowane, jeżeli funkcja będzie wywołana z pustym parametrem, funkcja zostanie wywołana i wykonana normalnie,
- RETURNS NULL ON NULL INPUT - specyfikowane, jeżeli wywoływana funkcja nie może mieć żadnego parametru wejściowy o wartości NULL, w tym przypadku zwracana jest wartość NULL. Ta opcja nie może być specyfikowana dla TVF UDF.

Opcja EXECUTE AS zarządza bezpieczeństwem wywołującego w UDF. Można określić jedną z opcji wykonania UDF.

- Przy opcji CALLER wywołujący określa, czy UDF powinna być wykonywana z uprawnieniami użytkownika wywołującego funkcję. Jest to wartość domyślna.
- Opcja SELF oznacza, że UDF powinna być wykonywana z uprawnieniami użytkownika, który utworzył (lub zmodyfikował) funkcję.
- Opcja OWNER wskazuje, że UDF powinna być wykonana z uprawnieniami właściciela UDF (lub właściciela schematu zawierającego UDF).
- Wskazuje nazwę użytkownika, z którego uprawnieniami ma być wykonywana UDF.

Funkcje UDF (zarówno T-SQL i CLR) nie mogą zmieniać stanu bazy danych. Poniżej funkcjonalności, które nie mogą być realizowane przez funkcje UDF.

- Modyfikacja danych w bazie danych. Nie można wykonać poleceń INSERT, UPDATE lub DELETE na danych, które nie są „temporary". Można utworzyć tabelę „temporary" i modyfikować dane w tej tabeli.
- Tworzyć nowe obiekty w bazie danych (z wyłączeniem obiektów typu „temporary").
- Wykonywać poleceń DCL.

- Zwracać dane bezpośrednio do klienta.
- Otwierać nową transakcję lub realizować ***commit*** czy ***rollback*** w ramach kontekstu transakcji.

Parametry ustawiane dla funkcji realizowanych w ramach CLR.

Property	Type	Description
DataAccess	Microsoft.SqlServer.Server.DataAccessKind	Indicates whether the function uses the inprocess provider to access SQL Server data. If it does, DataAccess should be set to DataAccessKind.Read; otherwise, it should be set to DataAccessKind.None.
FillRowMethodName	string	For a table-valued UDF, indicates the name of the method that will be called by SQL Server to populate a row in the returned resultset.
IsDeterministic	bool	Specifies whether the function is deterministic or not. *Deterministic* functions always return the same value for a given set of input parameters and can be used within computed columns that are indexed.
IsPrecise	bool	Indicates whether the function returns an exact value, or whether there is a degree of imprecision (e.g., because the function performs floating-point operations).
Name	string	Used by Visual Studio to set the name of the UDF when it is deployed to SQL Server. This property is ignored by SQL Server itself.
SystemDataAccess	Microsoft.SqlServer.Server.SystemDataAccessKind	Indicates whether the function reads data from the system tables using the in-process provider (SystemDataAccessKind.Read) or not (SystemDataAccessKind.None).
TableDefinition	string	Used by Visual Studio to set the return type of a table-valued UDF when it is deployed to SQL Server. This property is ignored by SQL Server itself.

Na początek przykładowe funkcje skalarne SVF (Scalar Valued Functions) UDF CLR.

```
// Skrypt Lab08.03
using System.Data.SqlTypes;
using System.Text.RegularExpressions;
public static class UDFExample
{
  private static readonly Regex email_pattern = new Regex
  (
    // Część przed znakiem @ ("nazwa lokalna").
    "^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*" +
    // Nazwa domeny po znaku @.
    "@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+" +
    // Domena najwyższego poziomu.
```

```
            "(?:[a-z]{2}|com|org|net|gov|mil|biz|info|mobi|name|aero|jobs|museum)\\b$"
          );
          [Microsoft.SqlServer.Server.SqlFunction(Name="udf_EmailMatch",
IsDeterministic=true)]
          public static SqlBoolean udf_Func1(SqlString input)
          {
            SqlBoolean result = new SqlBoolean();
            if (input.IsNull)
              result = SqlBoolean.Null;
            else
              result = (email_pattern.IsMatch(input.Value.ToLower()) == true)? SqlBoolean.True :
SqlBoolean.False;
            return result;
          }
          [Microsoft.SqlServer.Server.SqlFunction(Name="udf_RegExIsMatch",
IsDeterministic=true, IsPrecise=true)]
          public static bool udf_Func2(string pattern, string matchString)
          {
            Regex reg = new Regex(pattern.TrimEnd(null));
            return reg.Match(matchString.TrimEnd(null)).Success;
          }
        }
```

Sprawdzenie poprawności przygotowanych funkcji.

```
SELECT 'tester@google.com' AS Email,
          dbo.udf_EmailMatch (N'tester@google.com') AS Valid
UNION
SELECT '123@456789', dbo.udf_EmailMatch('123@456789')
UNION
SELECT 'user@google.com', dbo.udf_EmailMatch('db2@fis.agh.edu.pl');

DECLARE @Mail1 varchar(100),
      @Mail2 varchar(100),
      @Reg varchar(100) ;
SET @Mail1 = 'db2@fis.agh.edu.pl';
SET @Mail2 = 'misc.text';
SET @Reg = '[a-zA-Z0-9_\-]+@([a-zA-Z0-9_\-]+\.)+(com|org|edu|nz|au)';
SELECT @Mail1, dbo.udf_RegExIsMatch(@Reg,@Mail1) as valid
UNION
SELECT @mail2, dbo.udf_RegExIsMatch(@Reg,@Mail2);
```

Tworzenie funkcji TVF w technologii CLR .NET jest nieco bardziej skomplikowane niż pisanie funkcji SVF UDF. Tworzona funkcja UDF musi implementować interfejs *IEnumerable* oraz dodatkowo tworzona funkcja [SqlFunction] musi posiadać atrybut *FillRowMethodName*, który wskazuje jaka metoda powinna zostać wywołana, aby zapełnić każdy pojedynczy wiersz w zwróconym zestawie wyników.

Kiedy wywoływana jest funkcja UDF tworzona jest nowa instancja implementacji IEnumerable, taka jak ArrayList czy obiekt typu Collection, która wypełniana jest z wykorzystaniem jednego obiektu dla każdego wiersza w zwróconej tabeli. Najczęściej realizujemy to poprzez zdefiniowanie struktury z jednym polem dla każdej kolumny w zestawie wyników. Dodatkowo należy zaimplementować metodę, do której odwołuje się właściwość FillRowMethodName. Metoda ta posiada jeden parametr wejściowy typu obiekt, który SQL Server będzie używał do przekazania obiektu reprezentującego wiersz danych oraz

parametry wyjściowe reprezentujące zwracane dane (dla każdej kolumny w zwróconym zestawie wyników - odpowiedni natywny typ SQL).

W pierwszym przykładzie przedstawiono przykładowe implementacje TVF UDF dla funkcji pobierającej wybrane rekordy z tabeli bazodanowej oraz dla funkcji przekształcającej określony ciąg znaków na tablicę wykorzystując podany ogranicznik.

```
// Skrypt Lab08.04
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
using System.Data;
using System.Data.Sql;
using System.Data.SqlTypes;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;
public static class UDFExample
{
  private class DataResult {
    public SqlInt32 idperson;
    public SqlString lname;
    public SqlString phone;
    public DataResult(SqlInt32 IdPerson, SqlString LName, SqlString Phone) {
      idperson = IdPerson;
      lname = LName;
      phone = Phone;
    }
  }
  [Microsoft.SqlServer.Server.SqlFunction
   (Name="udf_ReadTable",
    DataAccess = DataAccessKind.Read,
    FillRowMethodName = "FillRowTable",
    TableDefinition = "IdPerson int, LName nvarchar(100), Phone nvarchar(50)" )]
  public static IEnumerable table(string cText)
  {
    ArrayList resCollection = new ArrayList();
    using (SqlConnection oConn = new SqlConnection("context connection=true"))
    {
      SqlCommand oCmd = new SqlCommand("SELECT ContactID, LastName, Phone
FROM AdventureWorks.Person.Contact WHERE LastName like @text ", oConn);
      oCmd.Parameters.Add("@text", SqlDbType.NVarChar).Value = cText;
      oConn.Open();
      SqlDataReader oCmdReader = oCmd.ExecuteReader();
      while (oCmdReader.Read())
      {
        resCollection.Add(new DataResult(oCmdReader.GetSqlInt32(0),
oCmdReader.GetSqlString(1), oCmdReader.GetSqlString(2)));
      }
      return resCollection;
    }
  }
  public static void FillRowTable(object dataResultObj, out SqlInt32 idperson,  out
SqlString lname, out SqlString phone)
  {
    DataResult dataResult = (DataResult) dataResultObj;
```

```
        idperson = dataResult.idperson;
        lname = dataResult.lname;
        phone = dataResult.phone;
      }
      [Microsoft.SqlServer.Server.SqlFunction
       (Name = "udf_StringToTable",
        FillRowMethodName = "FillRowTable1",
        TableDefinition = "ID NVARCHAR(50)")]
       public static IEnumerable SqlArray(SqlString str, SqlChars delimiter)
       {
          //return single element array if no delimiter is specified
          if (delimiter.Length == 0)
             return new string[1] { str.Value };
          //split the string and return a string array
          return str.Value.Split(delimiter[0]);
       }
       public static void FillRowTable1(object row, out SqlString str)
       {
          //set the column value
          str = new SqlString((string)row);
       }
}
```

Sprawdzenie poprawności przygotowanych funkcji.

```
SELECT * FROM dbo.udf_ReadTable('Ab%');
SELECT * FROM dbo.udf_StringToTable('test 1|test 2|test 3', '|');
```

W kolejnym przykładzie przedstawiona została funkcja TVF UDF odczytująca dane z tablic systemowych. Realizacja tej funkcji wymaga ustwienia odpowiednich upranień dla „assembly”.

```
// Skrypt Lab08.05
using System.Data.SqlTypes;
using System.Text.RegularExpressions;
using System.Diagnostics;
using System.Collections;
using System;
public static class UDFExample
{
   [Microsoft.SqlServer.Server.SqlFunction
 ( Name="udf_ReadEventLog",
   FillRowMethodName = "FillRow",
   TableDefinition = "logTime datetime,Message nvarchar(4000),Category
nvarchar(4000),InstanceId bigint")]
   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;
    }
}
```

Poprawność funkcji sprawdzimy wykonując poniższe polecenia w SSMS.

```
select top 10 * from dbo.udf_ReadEventLog(N'Application');
select top 10 * from dbo.udf_ReadEventLog(N'Security');
select top 10 * from dbo.udf_ReadEventLog(N'System');
```

Ćwiczenie C. Wyzwalacze CLR w MS SQL Server

W ramach tego punktu zapoznamy się z realizacją wyzwalaczy w bazie danych z wykorzystaniem technologii CLR. Pierwszy skrypt będzie zapisywał wyniki do pliku zewnętrznego, natomiast drugi będzie realizował funkcjonalność „echa". Obydwa wyzwalacze zostaną wyzwolone w chwili dodania rekordu do tabeli „test1". Projekt zrealizujemy w języku C# z wykorzystaniem środowiska Visual Studio i bazy danych „testCLR".

Dodatkowa informacja dotyczą tworzenia i uruchamiania wyzwalaczy z wykorzystaniem technologii CLR znajduje się w poniższych linkach.

https://msdn.microsoft.com/pl-pl/library/ms254959%28v=vs.110%29.aspx
https://msdn.microsoft.com/en-us/library/ms345101%28v=sql.105%29.aspx
https://msdn.microsoft.com/en-us/library/ms186711%28v=sql.105%29.aspx

Pierwszy projekt zapisuje do pliku zewnętrznego datę dodania rekordu do tabeli dla której uruchomiony został wyzwalacz.

```
// Skrypt lab08.06
using System;
using System.Data;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;
using System.IO;
public partial class Triggers
{
    [Microsoft.SqlServer.Server.SqlTrigger
      (Name = "udt_Trigger1", Target = "Table1", Event = "FOR UPDATE")]
    public static void SaveFile()
    {
        SqlContext.Pipe.Send("Trigger FIRED");
        using (StreamWriter outputFile = new StreamWriter(@"c:\lab9\log.txt"))
        {
            outputFile.WriteLine(DateTime.Today);
```

```
        }
    }
}
```

Na początek utworzymy wyzwalacz dla tabeli „test1” i sprawdzamy poprawność działania wyzwacza dodając rekord do tabeli „test1”.

```
create trigger dbo.<trigger-name> on <table> after insert
as external name <assembly>.<class>.<method> ;
```

W ramach kodu wyzwalacza zapisujemy rekordy do pliku znajdującego się na zewnątrz środowiska CLR uruchomionego na serwerze MS SQL. Wykonanie tego typu opercji wymaga dodatkowych uprawnień zarówno dla kodu zarządzalnego „assembly” jak i bazy danych. Na początek zostaną zmodyfikowane uprawnienia właściciela bazy danych umożliwiające zwiększenie uprawnień dla „assembly”. Poźniej zmieniamy poziom bezpieczeństwa „assembly”. Ponownie sprawdzamy działanie wyzwalacza dodając rekord do tabeli „test1”.

```
ALTER DATABASE testCLR SET TRUSTWORTHY ON;
ALTER ASSEMBLY <assembly> WITH permission_set = EXTERNAL_ACCESS;
```

Na koniec usuwamy dołaczony wyzwalacz.

```
drop trigger dbo.<trigger-name> ;
```

Drugi wyzwalacz będzie odczytywał wprowadzone dane do tabeli i wysyłał je na konsolę. Poniżej kod realizujący postawione zadanie.

```
// Skrypt Lab08.07
public partial class Triggers
{
   [Microsoft.SqlServer.Server.SqlTrigger
     (Name = "udt_Trigger2", Target = "Table1", Event = "FOR INSERT")]
   public static void showinserted()
   {
      SqlTriggerContext triggContext = SqlContext.TriggerContext;
      SqlConnection conn = new SqlConnection(" context connection =true ");
      conn.Open();
      SqlCommand sqlComm = conn.CreateCommand();
      SqlPipe sqlP = SqlContext.Pipe;
      SqlDataReader rdr;
      sqlComm.CommandText = "SELECT * from inserted";
      rdr = sqlComm.ExecuteReader();
      while (rdr.Read())
         sqlP.Send("Count= " + rdr.FieldCount.ToString() + " i1= " + rdr[0].ToString());
   }
   [Microsoft.SqlServer.Server.SqlTrigger
    (Name = "udt_Trigger3", Target = "table1", Event = "FOR
INSERT,UPDATE,DELETE")]
    public static void DMLTrigAction()
    {
      SqlTriggerContext oTrigContext = SqlContext.TriggerContext;
```

```
        if (oTrigContext.TriggerAction == TriggerAction.Insert)
        {
          SqlContext.Pipe.Send("INSERT PROCESSING LOGIC GOES HERE");
        }
        else if (oTrigContext.TriggerAction == TriggerAction.Update)
        {
          SqlContext.Pipe.Send("UPDATE PROCESSING LOGIC GOES HERE");
        }
        else if (oTrigContext.TriggerAction == TriggerAction.Delete)
        {
          SqlContext.Pipe.Send("DELETE PROCESSING LOGIC GOES HERE");
        }
        else {
          SqlContext.Pipe.Send("ACTION COULD NOT BE DETECTED");
        }
      }
    }
```

Dodajemy wyzwalacz do tabeli „test1” i sprawdzamy poprawność działania wyzwacza dodając rekord do tabeli „test1”. Po sprawdzeniu poprawności działania wyzwalacza usuwamy wyzwalacz poleceniem „drop”.

```
create trigger dbo.<trigger-name> on <table> after insert
as external name <assembly>.<class>.<method> ;
```

Zadania

Zadanie Z1

Napisać procedurę składowaną mającą za zadanie wyszukanie wszystkich rekordów HumanResources.Employee, dla których podany ciąg znaków (parametr procedury) będzie zawarty w polu Address1 tabeli Person.Address. Zadanie zrealizować nie korzystając z operatora LIKE, procedurę umieścić w bazie danych testCLR.

Zadanie Z2

Opracować funkcję UDF umożliwiającą przedstawienie daty w dowolnym formacie.

Zadanie Z3

Opracować funkcję TVF UDF realizującą funkcjonalność zrealizowaną w zadaniu 1.