Laboratorium 10 – MS SQL Server 2008

Temat: Środowisko CLR w MS SQL Server - wyzwalacze i transakcje

Opracowanie: A.Dydejczyk

Ćwiczenia do opracowania w trakcie laboratorium:

1. Ćwiczenie A. Wyzwalacze CLR w MS SQL Server
2. Ćwiczenie B. Transakcje w ramach kodu CLR
3. Zadania.

Ćwiczenie A. Wyzwalacze CLR w MS SQL Server

W ramach technologii CLR mamy do dyspozycji wyzwalacze działające na poleceniach DDL i DML.

Struktura wyzwalacza działającego na poleceniu DDL.

```
CREATE TRIGGER trigger_name ON { ALL SERVER | DATABASE }
        [WITH ENCRYPTION]
{ FOR | AFTER } { event_type | event_group } [,...n]
[WITH APPEND]
[NOT FOR REPLICATION]
{ AS
{ sql_statement [...n] | EXTERNAL NAME <method_specifier> }
}
        <method_specifier> ::= assembly_name.class_name.method_name
```

W ramach klauzuli ON definiujemy obszar działania wyzwalacza:

- **ALL SERVER** - wskazuje, że obszarem działania wyzwalacza jest serwer bazodanowy, gdzie FOR | AFTER wskazują na odpowiednie zdarzenie na serwerze;
- **DATABASE** - wskazuje na obszar działanie wyzwalacza tylko do aktualnej bazy danych, wyzwalacz będzie uruchamiany FOR | AFTER dla odpowiednich zdarzeń w bazie danych.

Przykładowe zdarzenia dla wyzwalacza typu DDL:

- **DDL_DATABASE_LEVEL_EVENTS** – wyzwalacz jest uruchamiany dla każdego zdarzenia DDL w ramach bazy danych
- Możliwe jest wyspecyfikowanie szczegółowych akcji w ramach bazy danych tj. CREATE_TABLE, ALTER_TABLE, DROP_TABLE, CREATE_VIEW, ALTER_VIEW, DROP_VIEW i tak dalej.
- **DDL_SERVER_LEVEL_EVENTS** – wyzwalacz jest uruchamiany dla zdarzeń na poziomie serwera bazodanowego tj.:
    - CREATE | ALTER | DROP LOGIN
    - CREATE | DROP HTTP ENDPOINT *(MS SQL Server 2005)*
    - GRANT | DENY | REVOKE SERVER ACCESS
    - CREATE | ALTER | DROP CERT

Opis wszystkich zdarzeń monitorowanych przez wyzwalacze dostępny jest pod poniższym adresem:
https://docs.microsoft.com/pl-pl/sql/relational-databases/triggers/ddl-event-groups?view=sql-server-2014
Zestaw zdarzeń można również otrzymać realizując poniższe polecenie SQL.

```
WITH DirectReports(name, parent_type, type, level, sort) AS
(
   SELECT CONVERT(varchar(255),type_name), parent_type, type, 1,
CONVERT(varchar(255),type_name)
   FROM sys.trigger_event_types
   WHERE parent_type IS NULL
   UNION ALL
   SELECT  CONVERT(varchar(255), REPLICATE ('|   ' , level) + e.type_name),
      e.parent_type, e.type, level + 1,
   CONVERT (varchar(255), RTRIM(sort) + '|   ' + e.type_name)
   FROM sys.trigger_event_types AS e
      INNER JOIN DirectReports AS d
      ON e.parent_type = d.type
)
SELECT parent_type, type, name
FROM DirectReports
ORDER BY sort;
```

W ramach prezentacji wyzwalaczy działających na poleceniach DDL zaprezentowane zostaną dwa wyzwalacze. Pierwszy wyzwalacz DDL będzie zapisywał w tablicy wszystkie zdarzenia dotyczące poleceń DDL w formacie XML, natomiast drugi wyzwalacz będzie zapisywał wybrane zdarzenia do przygotowanej odpowiednio tablicy logu.

Na początek tworzymy tablice logu w której zapisywane będą informacje o zdarzeniach.

```
CREATE TABLE EventDataLog(
TriggerName varchar(50) NOT NULL,
InsertTime datetime NOT NULL DEFAULT (getdate()),
XMLData xml NOT NULL)
```

```
// Skrypt lab10.01
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Xml;
using System.IO;
using Microsoft.SqlServer.Server;
public partial class Triggers
{
   [Microsoft.SqlServer.Server.SqlTrigger(Target = "DATABASE", Event = "FOR
DDL_DATABASE_LEVEL_EVENTS")]
   public static void EventDataXMLDump()
   {
      SqlTriggerContext sqlTrg = SqlContext.TriggerContext;
      string evData = sqlTrg.EventData.Value;
      XmlDocument xmlDoc = new XmlDocument();
      xmlDoc.LoadXml(evData);
```

```
        XmlNode xmlNd =
xmlDoc.SelectSingleNode("//EVENT_INSTANCE/EventType/text()");
        string eventType = xmlNd.Value;
        using (SqlConnection cn = new SqlConnection("context connection=true"))
        {
           cn.Open();
           string sql = "INSERT INTO testCLR.dbo.EventDataLog " +
           "(TriggerName, XMLData) VALUES('" + eventType + "',' " +
           evData + "')";
           SqlCommand sqlComm = new SqlCommand(sql, cn);
           sqlComm.ExecuteNonQuery();
           sqlComm.Dispose();
        }
     }
}
```

Następnie tworzymy assembly i odpowiedni wyzwalacz i sprawdzamy poprawność opracowanego wyzwalacza.

```
CREATE ASSEMBLY  [nazwa_assembly]  FROM  '--plik z dll --'

CREATE TRIGGER nazwa_wyzwalacza ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
as external name [assembly].[klasa].[metoda]
```

Do realizacji drugiego wyzwalacza wymagane są dwie tabele. W pierwszej zapisywane są zdarzenia, które będą monitorowane, natomiast w drugiej log informacji o zdarzeniu.

```
CREATE TABLE AuditEvent(
   AuditId BIGINT IDENTITY(1,1),
   TriggerName VARCHAR(100),
   StartTime CHAR(5),
   EndTime CHAR(5))
GO
SET QUOTED_IDENTIFIER OFF
INSERT INTO AuditEvent (TriggerName, StartTime, EndTime)
  VALUES ("CreateTable","07:00","18:00")
INSERT INTO AuditEvent (TriggerName, StartTime, EndTime)
  VALUES ("AlterTable","07:00","18:00")
GO
CREATE TABLE  EventLog (
   Log_Ident bigint NOT NULL IDENTITY (1, 1),
   ObjectName varchar(100) NOT NULL,
   ObjectType varchar(20) NOT NULL,
   LoginName varchar(80) NOT NULL,
   DatabaseName varchar(30) NOT NULL,
   PostTime datetime NOT NULL,
   ObjectDetails text)
GO

// Skrypt lab10.02
```

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


   public partial class Triggers
   {
      [Microsoft.SqlServer.Server.SqlTrigger(Name = "TriggerDDL", Target =
"DATABASE", Event = "FOR DDL_DATABASE_LEVEL_EVENTS")]
      public static void logEvent()
      {
         XmlDocument xmlEventData;
         ArrayList auditActions = ReadMonitoringActions();
         String[] arrActions = (String[])auditActions.ToArray(typeof(string));
         SqlTriggerContext sqlTrg = SqlContext.TriggerContext;
         for (int i = 0; i < arrActions.Length - 1; i += 3)
         {
            if (arrActions[i] == sqlTrg.TriggerAction.ToString())
            {
               xmlEventData = new XmlDocument();
               xmlEventData.LoadXml(sqlTrg.EventData.Value);
               WriteOldVersion(xmlEventData);
            }
         }
      }

      private static ArrayList ReadMonitoringActions()
      {
         ArrayList actions = new ArrayList();
         using (SqlConnection cn = new SqlConnection("context connection=true"))
         {
            cn.Open();
            string sql = "SELECT TriggerName,  StartTime,  EndTime " +
                     "FROM AuditEvent ";
            SqlCommand sqlComm = new SqlCommand(sql, cn);
            SqlDataReader sqlTriggers = sqlComm.ExecuteReader();
            while (sqlTriggers.Read())
            {
               actions.Add(sqlTriggers.GetString(0));
               actions.Add(sqlTriggers.GetString(1));
               actions.Add(sqlTriggers.GetString(2));
            }
            sqlTriggers.Close();
            return actions;
         }
      }

      private static void WriteOldVersion(XmlNode objectXML)
      {
         string objectName = "";
         string objectType = "";
         string objectDetails = "";
         string loginName = "";
         string databaseName = "";
         string postTime = "";
         XmlNodeReader xmlDetails = new XmlNodeReader(objectXML);
```

```
            while (xmlDetails.Read())
            {
               if (xmlDetails.NodeType == XmlNodeType.Element &&
                     xmlDetails.Name == "ObjectName")
               {
                  xmlDetails.Read();
                  objectName = xmlDetails.Value;
               }
               if (xmlDetails.NodeType == XmlNodeType.Element &&
                     xmlDetails.Name == "ObjectType")
               {
                  xmlDetails.Read();
                  objectType = xmlDetails.Value;
               }
               if (xmlDetails.NodeType == XmlNodeType.Element && xmlDetails.Name ==
"LoginName")
               {
                  xmlDetails.Read();
                  loginName = xmlDetails.Value;
               }
               if (xmlDetails.NodeType == XmlNodeType.Element &&
                     xmlDetails.Name == "DatabaseName")
               {
                  xmlDetails.Read();
                  databaseName = xmlDetails.Value;
               }
               if (xmlDetails.NodeType == XmlNodeType.Element &&
                     xmlDetails.Name == "PostTime")
               {
                  xmlDetails.Read();
                  postTime = xmlDetails.Value;
               }
               if (xmlDetails.NodeType == XmlNodeType.Element &&
                     xmlDetails.Name == "CommandText")
               {
                  xmlDetails.Read();
                  objectDetails = xmlDetails.Value;
               }
            }
            xmlDetails.Close();

            SqlPipe pipeSql = SqlContext.Pipe;
            using (SqlConnection cn = new SqlConnection("context connection=true"))
            {
               cn.Open();
               string sql = "INSERT INTO EventLog  " +
                           "(ObjectName,ObjectType,LoginName,DatabaseName," +
                           " PostTime,ObjectDetails)  " +
                           "VALUES('" + objectName.Trim() + "','" + objectType.Trim() +
                           "','" + loginName.Trim() + "','" + databaseName.Trim() +
                           "','" + postTime.Trim() + "','" + objectDetails.Trim() + "')";
               SqlCommand sqlComm = new SqlCommand(sql, cn);
               pipeSql.Send(sqlComm.CommandText);
               pipeSql.ExecuteAndSend(sqlComm);
            }
         }
      }
```

W ramach kolejnych zadań zapoznamy się z realizacją wyzwalaczy w bazie danych dla poleceń DML. 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 lab10.03
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",true))
      {
         outputFile.WriteLine(DateTime.Now);
      }
   }
}
```

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 SQL Server. 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 Lab10.04
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> ;
```

Ćwiczenie B. Transakcje w kodzie CLR

W ostatnim ćwiczeniu zostanie zaprezentowana realizacja transakcji w kodzie procedury CLR.

W ramach skryptu wykorzystamy „Transakcje implicite”, które są wykorzystywane również do tworzenia transakcji rozproszonych (ang. distributed transaction) . Transakcje tego typu są najszerzej wykorzystywane. Cecha charakterystyczną jest zaznaczenie bloku kodu przez utworzenie obiektu TransactionScope. W ramach tej transakcji nie zaznaczamy początku transakcji BEGIN TRANSACTION i nie stosujemy COMMIT i ROLLBACK. Blok kodu kończymy metody Complete() co może oznaczać zarówno commit jak i rollback w zależności od wyniku realizacji bloku kodu. Transakcje tego typu są „lekkie”, jeżeli jednak w transakcji otworzymy połączenie do zdalnego serwera to taka transakcji zmienia się (jest promowana) w transakcję rozproszoną (distributed).

W celu realizacji zadania dodamy dodatkową referencję obsługującą transakcje, której domyślnie nie ma w Solution Explorer. Referencję dodajemy w węźle References w Solution Explorer, w zakładce SQL dodajemy System.Transactions.

Poniżej kod do wykonania ćwiczenia.

```
// Skrypt Lab10.05
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.Transactions;


public partial class StoredProcedures
{
   [Microsoft.SqlServer.Server.SqlProcedure]
   public static void SimpleTransactionScope()
   {
      //tworzymy obiekt klasy  TransactionScope
      TransactionScope oTran = new TransactionScope();
      using (SqlConnection oConn = new SqlConnection ("context connection=true;"))
      {
         try
         {
            //otwieramy connection
            oConn.Open();
            //pierwszy krok transakcji
```

```
            SqlCommand oCmd =
               new SqlCommand("INSERT AdventureWorks.Production.ProductCategory "
+
                  "([Name]) SELECT 'Mobile Devices'", oConn);
            oCmd.ExecuteNonQuery();
            //drugi krok transakcji
            oCmd.CommandText = "INSERT " +
                "AdventureWorks.Production.ProductSubcategory SELECT " +
                "ProductCategoryID,'PocketPCs',NewID(),GetDate() " +
                "FROM AdventureWorks.Production.ProductCategory " +
                "WHERE [Name] = 'Mobile Devices'" ;
            oCmd.ExecuteNonQuery();
         }
         catch (SqlException ex)
         {
            SqlContext.Pipe.Send(ex.Message.ToString());
         }
         finally
         {
            //commit lub rollback
            oTran.Complete();
            oConn.Close();
         }
      }
      //Uwaga: usuwamy obiekt transakcji
      oTran.Dispose();
   }
};
```

Procedurę uruchamiamy dwukrotnie. Powtórne uruchomienie procedury wyświetli poniższy komunikat.

```
Cannot insert duplicate key row in object 'Production.ProductCategory' with unique index
'AK_ProductCategory_Name'.
The statement has been terminated.
```

Zadania

Zadanie Z1

Opracować wyzwalacz, który po dodaniu rekordu do tablicy „test1” wprowadzi rekord do utworzonej na potrzeby zadania tablicy logu zawierający datę wprowadzenia rekordu, wprowadzone dane oraz nazwę użytkownika.

Zadanie Z2

Opracować kod SQL zawierający dwie tabele połączone zależnością n-m z tablicą asocjacyjną wraz z procedurą CLR wstawiająca dane do tych trzech tabel w ramach transakcji.