Laboratorium 3 – MS SQL Server 2008

Temat: Język T-SQL

Opracowanie: A.Dydejczyk

Ćwiczenie E. User Define Function

Wprowadzone w SQL2000, a od wersji SQL 2005 wprowadzono UDF CLR.
Mają za zadanie przeprowadzanie obliczeń i przekształceń zwracając wartość lub tabelę.
Mogą być stosowane w pytaniach, więzach i kolumnach wyliczalnych (query, constraint, computed column).
Typ zwracanych danych oraz struktura funkcji są podstawą podziału funkcji użytkownika na trzy typy:

Funkcje skalarne — funkcje tego typu zwracają pojedynczą wartość i przypominają funkcje systemowe.
Proste funkcje tabelaryczne — funkcje zwracające wynik pojedynczej instrukcji SELECT i przypominające widoki, ale różniące się od nich możliwością określania dodatkowych parametrów wywołania.
Złożone funkcje tabelaryczne — funkcje tego typu zwracają wynik dowolnej liczby instrukcji języka Transact-SQL i najbardziej przypominają procedury składowane. Jednak, w przeciwieństwie do procedur składowanych, można się do nich odwoływać w klauzuli FROM instrukcji SELECT.

Domyślnie wyłącznie członkowie ról sysadmin, db_owner oraz db_ddladmin mogą tworzyć własne funkcje. Członkowie dwóch pierwszych ról mogą nadać innym użytkownikom uprawnienie do wykonywania instrukcji CREATE FUNCTION.

Niemożliwe jest tworzenie funkcji użytkownika zawierających w ciele jakąkolwiek niedeterministyczną funkcję systemową, to znaczy funkcję, która zwraca różne wartości, w zależności od okoliczności ich wywołania. Przykładem systemowych funkcji niedeterministycznych są funkcje CURRENT_USER (zwracająca nazwę zalogowanego użytkownika) oraz GETDATE (zwracająca bieżącą datę systemową).

Nie mogą wywoływać efektów ubocznych w stanie bazy na zewnątrz funkcji, np. nie mogą zmieniać zawartości bazy, korzystać z tabel temporary. Mogą tworzyć zmienne typu tabela.

Ograniczenia UDF

UDF nie mogą modyfikować stanu bazy
UDF nie mogą zawierać OUTPUT INTO wskazujące na tabelę
UDF nie mogą zwracać wielokrotnych result sets.
UDF nie mogą stosować TRY...CATCH, @ERROR or RAISERROR.
UDF nie mogą wywoływać procedury składowanej
UDF nie mogą używać dynamicznego SQL i tabeli chwilowych
Instrukcja SET nie jest dozwolona
Konstrukcja FOR XML nie jest dozwolona

1. Funkcja zwracająca wartość skalarną

```
--- Skrypt Lab02.18
SET NOCOUNT ON;
USE AdventureWorks;
GO
IF OBJECT_ID('dbo.fn_ConcatOrders') IS NOT NULL
   DROP FUNCTION dbo.fn_ConcatOrders;
GO

CREATE FUNCTION dbo.fn_ConcatOrders
 (@cid AS NCHAR(5)) RETURNS VARCHAR(8000)
AS
BEGIN
DECLARE @orders AS VARCHAR(8000);
SET @orders = '';
SELECT @orders = @orders + CAST(SalesOrderID AS VARCHAR(10))+';'
FROM Sales.SalesOrderHeader
WHERE CustomerID = @cid;
RETURN @orders;
END
```

Powyższą funkcję wykorzystujemy w zapytaniu przedstawionym poniżej:

```
SELECT CustomerID, dbo.fn_ConcatOrders(CustomerID) AS Orders
FROM Sales.SalesOrderHeader;
```

Po zakończeniu ćwiczenia usuwamy funkcję następującym poleceniem:

```
IF OBJECT_ID('dbo.fn_ConcatOrders') IS NOT NULL
   DROP FUNCTION dbo.fn_ConcatOrders;
```

W serwerach 2005 i nowszych możliwe jest otrzymanie powyższego wyniku wykonując poniższe zapytanie:

```
--- Skrypt Lab02.19
SET NOCOUNT ON;
USE AdventureWorks;
GO

SELECT CustomerID,
  (SELECT CAST(SalesOrderID AS VARCHAR(10)) + ';' AS [text()]
  FROM Sales.SalesOrderHeader AS O
  WHERE O.CustomerID = C.CustomerID
  ORDER BY SalesOrderID
  FOR XML PATH('')) AS Orders
FROM Sales.SalesOrderHeader AS C;
```

2. Użycie UDF w więzach – DEFAULT

Tworzymy tabelę T1 i funkcję fn_T1_getkey w bazie “tempdb”.

```
--- Skrypt Lab02.20
IF OBJECT_ID('dbo.T1') IS NOT NULL
  DROP TABLE dbo.T1;
GO
CREATE TABLE dbo.T1
(
  keycol INT NOT NULL CONSTRAINT PK_T1 PRIMARY KEY CHECK (keycol > 0),
  datacol VARCHAR(10) NOT NULL
);
GO

IF OBJECT_ID('dbo.fn_T1_getkey') IS NOT NULL
  DROP FUNCTION dbo.fn_T1_getkey;
GO

CREATE FUNCTION dbo.fn_T1_getkey() RETURNS INT
AS
BEGIN
 RETURN
   CASE
    WHEN NOT EXISTS(SELECT * FROM dbo.T1 WHERE keycol = 1) THEN 1
    ELSE (SELECT MIN(keycol + 1)
        FROM dbo.T1 AS A
        WHERE NOT EXISTS
         (SELECT *
          FROM dbo.T1 AS B
          WHERE B.keycol = A.keycol + 1))
  END;
END
GO
```

Tworzymy więzy integralności DEFAULT.

```
ALTER TABLE dbo.T1 ADD DEFAULT(dbo.fn_T1_getkey()) FOR keycol;
```

Testowanie opracowanego schematu.

```
INSERT INTO dbo.T1(datacol) VALUES('a');
INSERT INTO dbo.T1(datacol) VALUES('b');
INSERT INTO dbo.T1(datacol) VALUES('c');
DELETE FROM dbo.T1 WHERE keycol = 2;
INSERT INTO dbo.T1(datacol) VALUES('d');

SELECT * FROM dbo.T1;
```

3. UDF który zwraca tabelę

Ma podobne zastosowanie jak perspektywa z dodatkową możliwością parametryzacji (perspektywa nie posiada tej możliwości).

```
--- Skrypt Lab02.21
SET NOCOUNT ON;
USE AdventureWorks;
GO
IF OBJECT_ID('dbo.fn_GetCustOrders') IS NOT NULL
  DROP FUNCTION dbo.fn_GetCustOrders;
GO
CREATE FUNCTION dbo.fn_GetCustOrders
  (@cid AS NCHAR(5)) RETURNS TABLE
AS
RETURN
  SELECT SalesOrderID, CustomerID, SalesPersonID, OrderDate,
  Duedate, ShipDate, Freight
  FROM Sales.SalesOrderHeader
  WHERE CustomerID = @cid;
```

Poniżej test użycia opracowanej funkcji UDF.

```
SELECT O.SalesOrderID, O.CustomerID, OD.ProductID, OD.OrderQty
FROM dbo.fn_GetCustOrders(N'676') AS O
JOIN Sales.SalesOrderDetail AS OD
ON O.SalesOrderID = OD.SalesOrderID;
```

4. Multistatement Table Valued UDFs

Ten rodzaj UDF traktowany jest jako procedura składowana, uczestniczy w poleceniu SELECT w części FROM, nie może być użyty w polecenie np. UPDATE.

a) Przygotowanie struktur do testów w bazie danych „tempdb”.

```
--- Skrypt Lab02.22
SET NOCOUNT ON;
USE tempdb;
GO
IF OBJECT_ID('dbo.Employees') IS NOT NULL
  DROP TABLE dbo.Employees;
GO
CREATE TABLE dbo.Employees
(
  empid   INT         NOT NULL PRIMARY KEY,
  mgrid   INT         NULL     REFERENCES dbo.Employees,
  empname VARCHAR(25) NOT NULL,
  salary MONEY NOT NULL
);
```

```
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(1, NULL, 'David', $10000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(2, 1, 'Eitan', $7000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(3, 1, 'Ina', $7500.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(4, 2, 'Seraph', $5000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(5, 2, 'Jiru', $5500.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(6, 2, 'Steve', $4500.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(7, 3, 'Aaron', $5000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(8, 5, 'Lilach', $3500.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(9, 7, 'Rita', $3000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(10, 5, 'Sean', $3000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(11, 7, 'Gabriel', $3000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(12, 9, 'Emilia' , $2000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(13, 9, 'Michael', $2000.00);
INSERT INTO dbo.Employees(empid, mgrid, empname, salary)
  VALUES(14, 9, 'Didi', $1500.00);

CREATE UNIQUE INDEX idx_unc_mgrid_empid ON dbo.Employees(mgrid, empid);
GO
```

b) Utworzenie fukncji UDFdo realizacji zadania.

```
--- Skrypt Lab02.23
IF OBJECT_ID('dbo.fn_subordinates') IS NOT NULL
  DROP FUNCTION dbo.fn_subordinates;
GO
CREATE FUNCTION dbo.fn_subordinates(@mgrid AS INT) RETURNS @Subs Table
(
  empid   INT NOT NULL PRIMARY KEY NONCLUSTERED,
  mgrid   INT NULL,
  empname VARCHAR(25) NOT NULL,
  salary  MONEY       NOT NULL,
  lvl     INT NOT NULL,
  UNIQUE CLUSTERED(lvl, empid)
)
AS
BEGIN
  DECLARE @lvl AS INT;
  SET @lvl = 0;               -- Init level counter with 0
  -- Insert root node to @Subs
```

```
  INSERT INTO @Subs(empid, mgrid, empname, salary, lvl)
  SELECT empid, mgrid, empname, salary, @lvl
  FROM dbo.Employees WHERE empid = @mgrid;
WHILE @@rowcount > 0         -- while prev level had rows
BEGIN
  SET @lvl = @lvl + 1;       -- Increment level counter
  -- Insert next level of subordinates to @Subs
  INSERT INTO @Subs(empid, mgrid, empname, salary, lvl)
    SELECT C.empid, C.mgrid, C.empname, C.salary, @lvl
      FROM @Subs AS P          -- P = Parent
        JOIN dbo.Employees AS C -- C = Child
          ON P.lvl = @lvl - 1   -- Filter parents from prev level
          AND C.mgrid = P.empid;
  END
  RETURN;
END
GO
```

Wykorzystujemy opracowaną funkcję UDF. Podajemy id managera = 3, lv1 to odległość w hierarchii.

```
SELECT empid, mgrid, empname, salary, lvl
FROM dbo.fn_subordinates(3) AS S;
```

Od wersji 2005 można wykorzystać rekursje i CTE.

```
--- Skrypt Lab02.24
IF OBJECT_ID('dbo.fn_subordinates') IS NOT NULL
  DROP FUNCTION dbo.fn_subordinates;
GO
CREATE FUNCTION dbo.fn_subordinates(@mgrid AS INT) RETURNS TABLE
AS
RETURN
  WITH SubsCTE
  AS
  (
   -- Anchor member returns a row for the input manager
   SELECT empid, mgrid, empname, salary, 0 AS lvl
   FROM dbo.Employees
   WHERE empid = @mgrid
   UNION ALL
   -- Recursive member returns next level of subordinates
   SELECT C.empid, C.mgrid, C.empname, C.salary, P.lvl + 1
   FROM SubsCTE AS P
     JOIN dbo.Employees AS C
       ON C.mgrid = P.empid
  )
  SELECT * FROM SubsCTE;
GO
```

Sprawdzamy ponownie.

```
SELECT empid, mgrid, empname, salary, lvl
FROM dbo.fn_subordinates(3) AS S;
```

Po wykonaniu przykładów usuwamy utworzone obiekty z bazy.

```
USE tempdb;
GO
IF OBJECT_ID('dbo.Employees') IS NOT NULL
  DROP TABLE dbo.Employees;
GO
IF OBJECT_ID('dbo.fn_subordinates') IS NOT NULL
  DROP FUNCTION dbo.fn_subordinates;
```

W przypadku funkcji zdefiniowanych przez użytkowników użytkownik otrzymuje dodatkowe opcje (niektóre będą dostępne też dla , procedur składowanych, procedur wyzwalanych czy widoków) :

ENCRYPTION
SCHEMABINDING
EXECUTE AS
RETURNS NULL ON NULL INPUT
CALLED ON NULL INPUT (the default).

Zalecane: SCHEMABINDING, RETURNS NULL ON NULL INPUT

SCHEMABINDING

Tak jak w przypadku obiektów innych typów utworzenie, za pomocą opcji SCHEMABINDING, funkcji powiązanej ze schematem bazy danych spowoduje, że obiekty źródłowe funkcji nie będą mogły zostać zmodyfikowane lub usunięte. Aby możliwe było utworzenie funkcji użytkownika powiązanej ze schematem bazy danych:

- Wszystkie powiązane z funkcją widoki oraz funkcje użytkownika zostały powiązane ze schematem bazy danych.
- Wszystkie powiązane z funkcją obiekty znajdują się w tej samej bazie danych.
- W definicji funkcji nie występują odwołania do obiektów typu właściciel.nazwa_obiektu,
- Osoba tworząca funkcję ma nadane uprawnienie (co najmniej) DRI do wszystkich powiązanych z funkcją obiektów.

RETURNS NULL ON NULL INPUT

Poprawia wydajność pomijając logikę funkcji, jeżeli jeden z argumentów funkcji ma wartość NULL.

Przykład użycia opcji ENCRYPTION i SCHEMABINDING dla przykładu z widokiem.

```
--- Skrypt Lab02.25
USE  AdventureWorks;
GO
IF OBJECT_ID('dbo.VCustsWithOrders') IS NOT NULL
  DROP VIEW dbo.VCustsWithOrders;
GO
CREATE VIEW dbo.vCustsWithOrders WITH ENCRYPTION, SCHEMABINDING
AS
 SELECT ContactID, FirstName, LastName, EmailAddress, Phone
 FROM Person.Contact AS C
 WHERE EXISTS
  ( SELECT 1 FROM Sales.SalesOrderHeader As O
      WHERE O.ContactID = C.ContactID );
GO
```

Zadania

Zadanie Z1.

Baza danych: AdventureWorks2008
Temat: Funkcja UDF, która zwraca nchar z danymi określonymi przez BusinessEntityID. Każda z kolumn zwracanych jest oddzielona od pozostałych średnikiem.
Dane: nazwisko, imię, email, adres.
Tabele: Person.Person, Person.EmailAddress, Person.BussinessEntity, Person.BussinessEntityAddress, Person.Address.
Kolumny: PersonID, BusinessEntityID.

Zadanie Z2.

Baza danych: AdventureWorks2008
Temat: Funkcja UDF, która zwraca zestaw rekordów: nazwisko, imię, email, adres, uporządkowanych nazwisko, imię, adres. Z pełnego zestawu funkcja zwraca podzbiór określony przez numer rekordu w zakresie ustalonym przez argumenty funkcji.

Zadanie Z3.

Baza danych: AdventureWorks2008
Temat: Funkcja UDF, która zwraca tabelę z danymi zamówień dla zadanego odbiorcy. Odbiorcę zadajemy przez jego nazwę.
Tabele: Sales.Customer, Sales.SalesOrderHeader, Person.Person
Kolumny: CustomerID, PersonID,BusinessEntityID