Лабораторная работа № 3

SQL модули

Введение

Язык SQL в своем оригинальном виде не являлся привычным процедурным языком программирования (то есть не предоставлял средств для построения циклов, ветвлений и т. д.). По этой причине разные производители вводили различные процедурные расширения языка SQL. Microsoft SQL Server поддерживает следующие программируемые возможности:

- Stored procedures (Хранимые процедуры),
- Functions (Функции),
- Triggers (Триггеры),
- Assemblies (Сборки),
- Types (Типы данных),
- Rules (Правила),
- Default (Умолчания).

Целью данной лабораторной работы является изучение трех видов программируемых возможностей Microsoft SQL Server: хранимых процедур, функций и триггеров. В дальнейшем эти программируемые возможности будут называться SQL модулями.

Рекомендуемые ресурсы для выполнения лабораторной работы

Функции T-SQL

Определяемые пользователем функции (компонент Database Engine)
http://msdn.microsoft.com/ru-ru/library/ms189593.aspx

CREATE FUNCTION (Transact-SQL)
http://msdn.microsoft.com/ru-ru/library/ms186755.aspx

Хранимые процедуры T-SQL

Хранимые процедуры (компонент Database Engine)
http://msdn.microsoft.com/ru-ru/library/ms190782.aspx

CREATE PROCEDURE (Transact-SQL)
http://msdn.microsoft.com/ru-ru/library/ms187926.aspx

Триггеры DML T-SQL

Триггеры DML
http://msdn.microsoft.com/ru-ru/library/ms191524.aspx

CREATE TRIGGER (Transact-SQL)
http://msdn.microsoft.com/ru-ru/library/ms189799.aspx

Использование таблиц inserted и deleted
http://msdn.microsoft.com/ru-ru/library/ms191300.aspx

Применение триггеров INSTEAD OF
http://msdn.microsoft.com/ru-ru/library/ms179288.aspx

UPDATE() (Transact-SQL)
http://msdn.microsoft.com/ru-ru/library/ms187326.aspx

COLUMNS_UPDATED() (Transact-SQL)
http://msdn.microsoft.com/ru-ru/library/ms186329.aspx

Курсоры T-SQL

Курсоры (компонент Database Engine)
http://msdn.microsoft.com/ru-ru/library/ms191179.aspx

DECLARE CURSOR (Transact-SQL)
http://msdn.microsoft.com/ru-ru/library/ms180169.aspx

Теоретическая часть

Хранимая процедура - это сохраненная совокупность инструкций языка T-SQL или ссылка на метод среды CLR платформы Microsoft .NET Framework, которая может принимать и возвращать предоставленные пользователем параметры.

Функция - это подпрограмма языка T-SQL или ссылка на метод среды CLR платформы Microsoft .NET Framework, которая возвращает значение. Функция не должна (не может) выполнять действия, изменяющие состояние базы данных. Она должна быть (может быть) вызвана из запроса.

Триггер - это особая разновидность хранимой процедуры, выполняемая автоматически при возникновении события на сервере базы данных. Различают триггеры DDL и DML. Событиями DML являются инструкции INSERT, UPDATE или DELETE, применяемые к таблице или представлению. Событиями DDL прежде всего являются инструкциям CREATE, ALTER, DROP и вызовы некоторых системных хранимых процедур, которые выполняют схожие операции.

Подробные методические материалы по созданию и использованию модулей T-SQL представлены в папке «Методические материалы».

Задание

Разработать и тестировать 10 модулей

A. Четыре функции
    1) Скалярную функцию (пример 1)
    2) Подставляемую табличную функцию (пример 3)
    3) Многооператорную табличную функцию (пример 4)
    4) Рекурсивную функцию или функцию с рекурсивным ОТВ (примеры 2, 5)
B. Четыре хранимых процедуры
    1) Хранимую процедуру без параметров или с параметрами (примеры 6, 7, 8, 9)
    2) Рекурсивную хранимую процедуру или хранимую процедур с рекурсивным ОТВ (пример 10)
    3) Хранимую процедуру с курсором (пример 11)
    4) Хранимую процедуру доступа к метаданным (пример 12)
C. Два DML триггера
    1) Триггер AFTER (пример 13)
    2) Триггер INSTEAD OF (пример 14)

Примечания - варианты для составления хранимой процедуры доступа к метаданным находятся в файле «Варианты_для_хранимой_процедуры_метаданных».

Приложение

Примеры создания и выполнения модулей T-SQL

Функции T-SQL
Пример 1. Создание и вызовы скалярной функции

```
CREATE FUNCTION dbo.CalculateCircleArea (@Radius float = 1.0)
RETURNS float
WITH RETURNS NULL ON NULL INPUT
AS
BEGIN
      RETURN PI() * POWER(@Radius, 2);
END;
GO
-- Тестируем функцию
SELECT dbo.CalculateCircleArea(10);
SELECT dbo.CalculateCircleArea(NULL);
SELECT dbo.CalculateCircleArea(2.5);
GO
```

Результаты выполнения функции

```
NULL
NULL
265252859812191058636308480000000
```

Пример 2. Создание и вызовы рекурсивной скалярной функции

```
CREATE FUNCTION dbo.CalculateCircleArea (@Radius float = 1.0)
RETURNS float
WITH RETURNS NULL ON NULL INPUT
AS
BEGIN
      RETURN PI() * POWER(@Radius, 2);
END;
GO
-- Тестируем функцию
SELECT dbo.CalculateCircleArea(10);
SELECT dbo.CalculateCircleArea(NULL);
SELECT dbo.CalculateCircleArea(2.5);
GO
```

Результаты выполнения функции

```
314,159265358979
NULL
19,6349540849362
```

Пример 3. Создание и вызовы подставляемой табличной функции

```
-- Создаем вспомогательную таблицу, состоящую из одного целочисленного столбца
CREATE TABLE dbo.Numbers (Num int NOT NULL PRIMARY KEY);
GO

-- Заполняем эту таблицу числами от 0 до 30000
WITH NumCTE (Num)
```

```
AS
(
      SELECT 0

      UNION ALL

      SELECT Num + 1
      FROM NumCTE
      WHERE Num < 30000
)
INSERT INTO dbo.Numbers (Num)
SELECT Num
FROM NumCTE
OPTION (MAXRECURSION 0);
GO

-- Создаем функцию, которая разбивает строку на лексемы, если в качестве разделителя
лексем  -- используется запятая. Функция возвращает таблицу лексем и их порядковые
номера
CREATE FUNCTION dbo.GetCommaSplit (@String nvarchar(max))
RETURNS table
AS
RETURN
(
      WITH Splitter (Num, String)
      AS
      (
            SELECT Num, SUBSTRING
                  (
                        @String,
                        Num,
                        CASE CHARINDEX(N',', @String, Num)
                              WHEN 0 THEN LEN(@String) - Num + 1
                              ELSE CHARINDEX(N',', @String, Num) - Num
                        END
                  ) AS String
            FROM dbo.Numbers
            WHERE Num <= LEN(@String)
                  AND (SUBSTRING(@String, Num - 1, 1) = N',' OR Num = 0)
      )
      SELECT ROW_NUMBER() OVER (ORDER BY Num) AS Num, RTRIM(LTRIM(String)) AS Element
      FROM Splitter
      WHERE String <> ''
);
GO
-- Тестируем функцию
SELECT Num, Element
FROM dbo.GetCommaSplit ('Michael, Tito, Jermaine, Marlon, Rebbie, Jackie, Janet, La
Toya, Randy');
GO
```

Результаты выполнения функции

```
Num                  Element
-------------------- ---------------
                   1 Michael
                   2 Tito
                   3 Jermaine
                   4 Marlon
```

```
                  5 Rebbie
                  6 Jackie
                  7 Janet
                  8 La Toya
                  9 Randy

(обработано строк: 9)
```

Пример 4. Создание и вызов многооператорной табличной функции

```
USE [Northwind]
GO

/* Строим дерево субординации, возвращая таблицу с ID узлов и уровнями узлов в
дереве */
CREATE FUNCTION [dbo].[ufnSUBORDINATES](@root int)
RETURNS @Subs Table (empid int NOT NULL PRIMARY KEY, lvl int NOT NULL)
AS
BEGIN
      DECLARE @lvl int = 0; -- Инициализируем уровень в 0
      INSERT INTO @Subs(empid, lvl) -- Вставляем в @Subs корень
      SELECT EmployeeID, @lvl FROM dbo.Employees WHERE EmployeeID = @root;
      WHILE @@rowcount > 0 -- пока предыдущий уровень имеет строки
      BEGIN
            SET @lvl = @lvl + 1; -- Увеличиваем счетчик уровня
            INSERT INTO @Subs(empid, lvl) /* Вставляем в @Subs следующий уровень
субординации */
            SELECT C.EmployeeID, @lvl -- ID узла и уровень узла
            FROM @Subs AS P /* P = Родитель */ JOIN dbo.Employees AS C /* C =
Потомок */
         ON P.lvl = @lvl - 1 /* Фильтруем родителей из предыдущего уровня */ AND
C.ReportsTo = P.empid;
      END
      RETURN;
END
GO
-- Тестируем функцию
SELECT * FROM [dbo].[ufnSUBORDINATES](5)
```

Результаты выполнения функции

```
empid       lvl
----------- -----------
          5           0
          6           1
          7           1
          9           1

(обработано строк: 4)
```

Пример 5. Создание и вызовы скалярной функции, использующей рекурсивное ОТВ

```
CREATE FUNCTION dbo.CalculateFactorial1 (@n int = 1)
RETURNS float
WITH RETURNS NULL ON NULL INPUT
AS
BEGIN
      DECLARE @result float;
      SET @result = NULL;
```

```
	IF @n > 0
	BEGIN
		SET @result = 1.0;
		WITH Numbers (num)
		AS
		(
			SELECT 1

			UNION ALL

			SELECT num + 1
			FROM Numbers
			WHERE num < @n
		)
		SELECT @result = @result * num
		FROM Numbers;
	END;
	RETURN @result;
END;
GO

SELECT dbo.CalculateFactorial1 (-10);
SELECT dbo.CalculateFactorial1 (NULL);
SELECT dbo.CalculateFactorial1 (100);
GO
```

Результаты выполнения функции

```
NULL
NULL
9,33262154439441E+157
```

Хранимые процедуры T-SQL

Решение задачи о ханойских башнях на T-SQL

Пример 6. Создание хранимой процедуры без параметров

```
-- Создаем процедуру отображения всех дисков на соответствующей башне
CREATE PROCEDURE dbo.ShowTowers
AS
BEGIN
-- Каждый диск отображается, например, так "===3===", где 3 – номер диска, а === -
радиус
-- диска

-- Для отображения дисков в соответствующем порядке на каждой башне используется ОТВ
-- Рекурсивное ОТВ генерирует таблицу с числами 1...5
	WITH FiveNumbers(Num)
	AS
	(
		SELECT 1

		UNION ALL

		SELECT Num + 1
		FROM FiveNumbers
		WHERE Num < 5
	),
```

```
	GetTowerA (Disc) -- Диски для башни A
	AS
	(
		SELECT COALESCE(a.Disc, -1) AS Disc
		FROM FiveNumbers f
		LEFT JOIN #TowerA a
			ON f.Num = a.Disc
	),
	GetTowerB (Disc) -- Диски для башни B
	AS
	(
		SELECT COALESCE(b.Disc, -1) AS Disc
		FROM FiveNumbers f
		LEFT JOIN #TowerB b
			ON f.Num = b.Disc
	),
	GetTowerC (Disc) -- Диски для башни C
	AS
	(
		SELECT COALESCE(c.Disc, -1) AS Disc
		FROM FiveNumbers f
		LEFT JOIN #TowerC c
			ON f.Num = c.Disc
	)
	-- Данная инструкция SELECT генерирует текстовое представление всех трех башен и
	-- всех пяти дисков. FULL OUTER JOIN используется для представления башен в
формате
	-- “бок-о-бок“.
	SELECT CASE a.Disc
			WHEN 5 THEN ' =====5===== '
			WHEN 4 THEN '  ====4====  '
			WHEN 3 THEN '   ===3===   '
			WHEN 2 THEN '    ==2==    '
			WHEN 1 THEN '     =1=     '
			ELSE '       |       '
			END AS Tower_A,
		CASE b.Disc
			WHEN 5 THEN ' =====5===== '
			WHEN 4 THEN '  ====4====  '
			WHEN 3 THEN '   ===3===   '
			WHEN 2 THEN '    ==2==    '
			WHEN 1 THEN '     =1=     '
			ELSE '       |       '
			END AS Tower_B,
		CASE c.Disc
			WHEN 5 THEN ' =====5===== '
			WHEN 4 THEN '  ====4====  '
			WHEN 3 THEN '   ===3===   '
			WHEN 2 THEN '    ==2==    '
			WHEN 1 THEN '     =1=     '
			ELSE'       |       '
			END AS Tower_C
	FROM
	(
		SELECT ROW_NUMBER() OVER(ORDER BY Disc) AS Num,
		COALESCE(Disc, -1) AS Disc
		FROM GetTowerA
	) a
	FULL OUTER JOIN
```

```
	(
		SELECT ROW_NUMBER() OVER(ORDER BY Disc) AS Num,
		COALESCE(Disc, -1) AS Disc
		FROM GetTowerB
	) b
		ON a.Num = b.Num
	FULL OUTER JOIN
	(
		SELECT ROW_NUMBER() OVER(ORDER BY Disc) AS Num,
		COALESCE(Disc, -1) AS Disc
		FROM GetTowerC
	) c
		ON b.Num = c.Num
	ORDER BY a.Num;
END;
GO
```

Пример 7. Создание хранимой процедуры с входными параметрами

```
-- Создаем хранимую процедуру, которая перемещает один единственный диск с башни
источника -- на башню приемник
CREATE PROCEDURE dbo.MoveOneDisc
(
	@Source nchar(1),
	@Dest nchar(1)
)
AS
BEGIN
	-- @SmallestDisc – наименьший диск на башне источнике
	DECLARE @SmallestDisc int
	SET @SmallestDisc = 0;

	-- IF ... ELSE conditional statement gets the smallest disc from the
	-- correct source tower
	IF @Source = N'A'
	BEGIN
		-- Это дает нам наименьший диск на башне A
		SELECT @SmallestDisc = MIN(Disc)
		FROM #TowerA;

		-- Теперь удаляем этот диск с башни A
		DELETE FROM #TowerA
		WHERE Disc = @SmallestDisc;
	END
	ELSE IF @Source = N'B'
	BEGIN
		-- Это дает нам наименьший диск на башне B
		SELECT @SmallestDisc = MIN(Disc)
		FROM #TowerB;

		-- Теперь удаляем этот диск с башни B
		DELETE FROM #TowerB
		WHERE Disc = @SmallestDisc;
	END
	ELSE IF @Source = N'C'
	BEGIN
		-- Это дает нам наименьший диск на башне C
		SELECT @SmallestDisc = MIN(Disc)
		FROM #TowerC;
```

```
            -- Теперь удаляем этот диск с башни C
            DELETE FROM #TowerC
            WHERE Disc = @SmallestDisc;
      END

      -- Показываем, что происходит перемещение диска
      SELECT N'Moving Disc (' + CAST(COALESCE(@SmallestDisc, 0) AS nchar(1)) +
            N') from Tower ' + @Source + N' to Tower ' + @Dest + ':' AS Description;

      -- Выполняе перемещение - INSERT перекладывает диск с башни источника на башню
      -- приемник
      IF @Dest = N'A'
            INSERT INTO #TowerA (Disc) VALUES (@SmallestDisc);
      ELSE IF @Dest = N'B'
            INSERT INTO #TowerB (Disc) VALUES (@SmallestDisc);
      ELSE IF @Dest = N'C'
            INSERT INTO #TowerC (Disc) VALUES (@SmallestDisc);

      -- Показываем башни в новом состоянии
      EXECUTE dbo.ShowTowers;
END;
GO
```

Пример 8. Создание рекурсивной хранимой процедуры с входными и выходными параметрами

```
-- Создаем хранимую процедуру, которая рекурсивно перемещает все диски
CREATE PROCEDURE dbo.MoveDiscs
(
      @DiscNum int,
      @MoveNum int OUTPUT,
      @Source nchar(1) = N'A',
      @Dest nchar(1) = N'C',
      @Aux nchar(1) = N'B'
)
AS
BEGIN
      -- Если число перемещаемых дисков равно 0, то решение найдено
      IF @DiscNum = 0
            PRINT N'Done';
      ELSE
      BEGIN
            -- If the number of discs to move is 1, go ahead and move it
            IF @DiscNum = 1
            BEGIN
                  -- Increase the move counter by 1
                  SELECT @MoveNum = @MoveNum + 1;

                  -- And finally move one disc from source to destination
                  EXEC dbo.MoveOneDisc @Source, @Dest;
            END
            ELSE
            BEGIN
                  -- Determine number of discs to move from source to auxiliary tower
                  DECLARE @n int;
                  SET @n = @DiscNum - 1;

                  -- Move (@DiscNum - 1) discs from source to auxiliary tower
                  EXEC dbo.MoveDiscs @n, @MoveNum OUTPUT, @Source, @Aux, @Dest;
```

```
                -- Move 1 disc from source to final destination tower
                EXEC dbo.MoveDiscs 1, @MoveNum OUTPUT, @Source, @Dest, @Aux;

                -- Move (@DiscNum - 1) discs from auxiliary to final destination
tower
                EXEC dbo.MoveDiscs @n, @MoveNum OUTPUT, @Aux, @Dest, @Source;
            END;
        END;
END;
GO
```

Пример 9. Создание (основной) хранимой процедуры без параметров

```
-- This SP creates the three towers and populates Tower A with 5 discs
CREATE PROCEDURE dbo.SolveTowers
AS
BEGIN
      -- SET NOCOUNT ON to eliminate system messages that will clutter up
      -- the Message display
      SET NOCOUNT ON;

      -- Create the three towers: Tower A, Tower B, and Tower C
      CREATE TABLE #TowerA (Disc int PRIMARY KEY NOT NULL);
      CREATE TABLE #TowerB (Disc int PRIMARY KEY NOT NULL);
      CREATE TABLE #TowerC (Disc int PRIMARY KEY NOT NULL);

      -- Populate Tower A with all five discs
      INSERT INTO #TowerA (Disc)
      VALUES (1);
      INSERT INTO #TowerA (Disc)
      VALUES (2);
      INSERT INTO #TowerA (Disc)
      VALUES (3);
      INSERT INTO #TowerA (Disc)
      VALUES (4);
      INSERT INTO #TowerA (Disc)
      VALUES (5);

      -- Initialize the move number to 0
      DECLARE @MoveNum int;
      SET @MoveNum = 0;

      -- Show the initial state of the towers
      EXECUTE dbo.ShowTowers;

      -- Solve the puzzle. Notice you don't need to specify the parameters
      -- with defaults
      EXECUTE dbo.MoveDiscs 5, @MoveNum OUTPUT;

      -- How many moves did it take?
      PRINT N'Solved in ' + CAST (@MoveNum AS nvarchar(10)) + N' moves.';

      -- Drop the temp tables to clean up - always a good idea.
      DROP TABLE #TowerC;
      DROP TABLE #TowerB;
      DROP TABLE #TowerA;

      -- SET NOCOUNT OFF before we exit
```

```
      SET NOCOUNT OFF;
END;
GO
-- Тестируем хранимые процедуры для решения задачи о Ханойских башнях
EXECUTE dbo.SolveTowers;
GO
```

Пример 10. Создание и вызов хранимой процедуры с курсором

```
USE AdventureWorks;
GO

-- Создаем хранимую процедуру с курсором
CREATE PROCEDURE uspCursorScope
AS
BEGIN
      DECLARE @Counter int = 1, @OrderID int, @CustomerID int
      DECLARE CursorTest CURSOR
      GLOBAL
      FOR
            SELECT SalesOrderID, CustomerID
            FROM Sales.SalesOrderHeader;
      OPEN CursorTest;
      FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID;
      PRINT 'Row ' + CAST(@Counter AS varchar) + ' has a SalesOrderID of ' +
            CONVERT(varchar,@OrderID) + ' and a CustomerID of ' + CAST(@CustomerID AS
varchar);
      WHILE (@Counter<=5) AND (@@FETCH_STATUS=0)
      BEGIN
            SELECT @Counter = @Counter + 1;
            FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID;
            PRINT 'Row ' + CAST(@Counter AS varchar) + ' has a SalesOrderID of ' +
                  CONVERT(varchar,@OrderID) + ' and a CustomerID of ' +
CAST(@CustomerID AS varchar);
      END
      -- Курсор остается открытым
END;
GO

-- Тестируем хранимую процедуру
EXECUTE uspCursorScope;

-- Продолжаем работать с курсором, т.к. он открыт
DECLARE @Counter int = 6, @OrderID int, @CustomerID int;
WHILE (@Counter<=10) AND (@@FETCH_STATUS=0)
BEGIN
      PRINT 'Row ' + CAST(@Counter AS varchar) + ' has a SalesOrderID of ' +
            CAST(@OrderID AS varchar) + ' and a CustomerID of ' +
            CAST(@CustomerID AS varchar);
      SELECT @Counter = @Counter + 1;
      FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID;
END;
-- Наконец закрываем курсор
CLOSE CursorTest;
DEALLOCATE CursorTest;
```

Результаты выполнения хранимой процедуры и сценария

```
Row 1 has a SalesOrderID of 43659 and a CustomerID of 676
```

```
Row 2 has a SalesOrderID of 43660 and a CustomerID of 117
Row 3 has a SalesOrderID of 43661 and a CustomerID of 442
Row 4 has a SalesOrderID of 43662 and a CustomerID of 227
Row 5 has a SalesOrderID of 43663 and a CustomerID of 510
Row 6 has a SalesOrderID of 43664 and a CustomerID of 397

Row 7 has a SalesOrderID of 43665 and a CustomerID of 146
Row 8 has a SalesOrderID of 43666 and a CustomerID of 511
Row 9 has a SalesOrderID of 43667 and a CustomerID of 646
Row 10 has a SalesOrderID of 43668 and a CustomerID of 514
```

Пример 11. Создание и вызов хранимой процедуры, использующей рекурсивное ОТВ

```
USE Northwind;
GO

CREATE PROCEDURE dbo.uspSUBORDINATES
AS
      WITH OrgChart (EmployeeID, ReportsTo, Title, Level, Node)
      AS (
            SELECT EmployeeID, ReportsTo, Title, 0, CONVERT(VARCHAR(30),'/') AS Node
            FROM Employees
            WHERE ReportsTo IS NULL
            UNION ALL
            SELECT a.EmployeeID, a.ReportsTo, a.Title, b.Level + 1,
CONVERT(VARCHAR(30), b.Node + CONVERT(VARCHAR, a.ReportsTo) + '/')
            FROM Employees AS a INNER JOIN OrgChart AS b ON a.ReportsTo = b.EmployeeID
      )
      SELECT EmployeeID, ReportsTo, SPACE(Level * 3) + Title AS Title, Level, Node
      FROM OrgChart
      ORDER BY Node;
GO
-- Тестирует хранимую процедуру
EXECUTE dbo.uspSUBORDINATES;
GO
```

Результаты выполнения хранимой процедуры

```
EmployeeID  ReportsTo   Title                     Level       Node
----------- ----------- ------------------------- ----------- -----------
          2 NULL        Vice President, Sales                0 /
          1           2 Sales Representative                 1 /2
          3           2 Sales Representative                 1 /2/
          4           2 Sales Representative                 1 /2/
          5           2 Sales Director                       1 /2/
          8           2 Inside Sales Coordinator             1 /2/
          6           5 Sales Representative                 2 /2/5/
          7           5 Sales Representative                 2 /2/5/
          9           5 Sales Representative                 2 /2/5/

(обработано строк: 9)
```

Пример 12. Создание и вызов хранимой процедуры доступа к метаданным

```
-- Шаг 1: Создание таблицы ObjList, используемой в хранимой процедуре:
CREATE TABLE [ObjList] (
      [DBName] [sysname] NOT NULL ,
      [Object_Type] [varchar] (20) NULL ,
      [ObjectOwner] [sysname] NOT NULL ,
```

```
	[ObjectName] [sysname] NOT NULL ,
	[cur_date] [datetime] NOT NULL CONSTRAINT [DF_ObjList_cur_date] DEFAULT
(getdate())
);
GO

-- Шаг 2: Создание хранимой процедуры Object_owned_by_non_dbo:
CREATE PROCEDURE Object_owned_by_non_dbo
AS
BEGIN
	DECLARE @dbname varchar(200), @mSql1 varchar(8000);
	SET NOCOUNT ON;
	DECLARE DBName_Cursor CURSOR FOR
	SELECT name
	FROM master.dbo.sysdatabases
	WHERE name NOT IN ('master', 'model', 'msdb', 'tempdb')
	ORDER BY name;
	OPEN DBName_Cursor;
	FETCH NEXT FROM DBName_Cursor INTO @dbname;
	WHILE @@FETCH_STATUS = 0
	BEGIN
		SET @mSql1 = 'INSERT INTO ObjList (DBName, Object_Type, ObjectOwner,
ObjectName)'+char(13);
		SET @mSql1 = @mSQL1+'SELECT '''+@dbname+''' AS dbName, ObjType =
			CASE xtype
				WHEN ''U'' THEN ''Table''
				WHEN ''V'' THEN ''View''
				WHEN ''P'' THEN ''Procedure''
				WHEN ''FN'' THEN ''UD Function''
				ELSE xtype END, SU.name, SO.name FROM '
			+@dbname+'.dbo.sysobjects SO JOIN '+@dbname+'.dbo.sysusers SU ON
SO.uid = SU.uid AND su.name <> ''dbo''
AND SO.type IN (''U'',''V'',''P'',''FN'')';
		-- PRINT @mSql1
		EXEC ( @mSql1 );
		FETCH NEXT FROM DBName_Cursor INTO @dbname;
	END;
	CLOSE DBName_Cursor;
	DEALLOCATE DBName_Cursor;
END;
GO

-- Шаг 3: Выполнение созданной хранимой процедуры:
EXECUTE Object_owned_by_non_dbo;
GO

-- Шаг 4: Извлечение данных из таблицы ObjList:
SELECT * FROM [ObjList];
GO

-- Шаг 5: Уничтожение таблицы ObjList
DROP TABLE [ObjList];
GO
```

Результаты выполнения хранимой процедуры (для конкретного случая преподавателя)

```
DBName      Object_Type ObjectOwner ObjectName        cur_date
------ ------------ ---------- ------------------ ----------------------
myDB  Table       User3       CustomerInformation     2013-02-18 02:18:25.320
```

```
myDB  Procedure   User2       ViewCustomerNames 2013-02-18 02:18:25.320
```

Триггеры DML T-SQL

Пример 13. Создание и использование триггера AFTER

```
USE Northwind;
GO

-- Создаем таблицу, выполняющую роль журнала аудита изменений в таблицах
CREATE TABLE dbo.DmlActionLog
(
      EntryNum int IDENTITY(1, 1) PRIMARY KEY NOT NULL,
      SchemaName sysname NOT NULL,
      TableName sysname NOT NULL,
      ActionType nvarchar(10) NOT NULL,
      ActionXml xml NOT NULL,
      UserName nvarchar(256) NOT NULL,
      Spid int NOT NULL,
      ActionDateTime datetime NOT NULL DEFAULT (GETDATE())
);
GO

-- Создаем триггер AFTER INSERT, UPDATE, DELETE для таблицы Customers,
-- который регистрирует любые изменения в таблице Customers
CREATE TRIGGER CustomersChangeAudit
ON Customers
AFTER INSERT, UPDATE, DELETE
NOT FOR REPLICATION
AS
BEGIN
      -- Получить число обработанных строк
      DECLARE @Count int;
      SET @Count = @@ROWCOUNT

      -- Убедиться в том, что хотя бы одна строка на самом деле обработана
      IF (@Count > 0)
      BEGIN
            -- Отключить сообщения вида "rows affected"
            SET NOCOUNT ON;
            DECLARE @ActionType nvarchar(10);
            DECLARE @ActionXml xml;

            -- Получить количество вставляемых строк
            DECLARE @inserted_count int;
            SET @inserted_count = (SELECT COUNT(*) FROM inserted);

            -- Получить количество удаляемых строк
            DECLARE @deleted_count int;
            SET @deleted_count = (SELECT COUNT(*) FROM deleted);

            -- Определить тип действия DML, которое привело к срабатыванию триггера
            SELECT @ActionType = CASE
                  WHEN (@inserted_count > 0) AND (@deleted_count = 0)
                        THEN N'insert'
                  WHEN (@inserted_count = 0) AND (@deleted_count > 0)
                        THEN N'delete'
                        ELSE N'update'
```

```
                END;

            -- Использовать FOR XML AUTO для получения снимков до и после изменения
            -- данных в формате XML
            SELECT @ActionXml = COALESCE
                (
                    (
                        SELECT *
                        FROM deleted
                        FOR XML AUTO
                    ), N'<deleted/>'
                ) + COALESCE
                (
                    (
                        SELECT *
                        FROM inserted
                        FOR XML AUTO
                    ), N'<inserted/>'
                );

            -- Вставить строки для протоколирования действий в журнал аудита таблицы
            INSERT INTO dbo.DmlActionLog
            (
                SchemaName,
                TableName,
                ActionType,
                ActionXml,
                UserName,
                Spid,
                ActionDateTime
            )
            SELECT
                OBJECT_SCHEMA_NAME(@@PROCID, DB_ID()),
                OBJECT_NAME(t.parent_id, DB_ID()),
                @ActionType,
                @ActionXml,
                USER_NAME(),
                @@SPID,
                GETDATE()
            FROM sys.triggers t
            WHERE t.object_id = @@PROCID;
        END;
END;
GO
```

Тестирование триггера на инструкциях UPDATE, INSERT и DELETE

```
USE Northwind;
GO

UPDATE Customers
SET CompanyName = N'Lotus Software'
WHERE CustomerID = 'WOLZA';
GO

INSERT INTO Customers
(
    CustomerID,
```

```
      CompanyName
)
VALUES
(
      N'ZXCVB',
      N'Mandriva'
);
GO

DELETE
FROM Customers
WHERE CompanyName = N'Mandriva';
GO

SELECT
      EntryNum,
      SchemaName,
      TableName,
      ActionType,
      ActionXml,
      UserName,
      Spid,
      ActionDateTime
FROM dbo.DmlActionLog;
GO
```

Результаты срабатывания триггера

```
EntryNum SchemaName     TableName ActionType ActionXml UserName Spid ActionDateTime
-------- ---------- --------- ---------- --------- -------- ---- ---------------------
--
1       dbo        Customers update  <deleted  dbo     53   2011-03-15 03:50:17.450
2       dbo        Customers insert  <deleted  dbo     53   2011-03-15 03:50:17.530
3       dbo        Customers delete  <deleted  dbo     53   2011-03-15 03:50:17.560

<deleted CustomerID="WOLZA" CompanyName="Wolski  Zajazd" ContactName="Zbyszek
Piestrzeniewicz" ContactTitle="Owner" Address="ul. Filtrowa 68" City="Warszawa"
PostalCode="01-012" Country="Poland" Phone="(26) 642-7012" Fax="(26) 642-7012" />
<inserted CustomerID="WOLZA" CompanyName="Lotus Software" ContactName="Zbyszek
Piestrzeniewicz" ContactTitle="Owner" Address="ul. Filtrowa 68" City="Warszawa"
PostalCode="01-012" Country="Poland" Phone="(26) 642-7012" Fax="(26) 642-7012" />
<deleted />
<inserted CustomerID="ZXCVB" CompanyName="Mandriva" />
<deleted CustomerID="ZXCVB" CompanyName="Mandriva" />
<inserted />
```

Пример 14. Создание и использование триггера INSTEAD OF

```
-- Создаем тестовую базу данных специально для примера
CREATE DATABASE OurInsteadOfTest;
GO

USE OurInsteadOfTest;
GO

-- Создаем тестовые таблицы
CREATE TABLE dbo.Customers
    (
```

```
    CustomerID varchar(5) NOT NULL PRIMARY KEY ,
    Name varchar(40) NOT NULL
    );

CREATE TABLE dbo.Orders
     (
    OrderID int IDENTITY NOT NULL PRIMARY KEY,
    CustomerID varchar(5) NOT NULL
        REFERENCES Customers(CustomerID),
    OrderDate datetime NOT NULL
    );

CREATE TABLE dbo.Products
     (
    ProductID int IDENTITY NOT NULL PRIMARY KEY,
    Name varchar(40) NOT NULL,
    UnitPrice money NOT NULL
    );

CREATE TABLE dbo.OrderItems
     (
    OrderID int NOT NULL
        REFERENCES dbo.Orders(OrderID),
    ProductID int NOT NULL
        REFERENCES dbo.Products(ProductID),
    UnitPrice money NOT NULL,
    Quantity int NOT NULL
        CONSTRAINT PKOrderItem PRIMARY KEY CLUSTERED
            (OrderID, ProductID)
    );

    select * FROM dbo.OrderItems

-- Наполняем тестовые таблицы небольшим количеством данных
INSERT dbo.Customers
    VALUES ('ABCDE', 'Bob''s Pretty Good Garage');

INSERT dbo.Orders
    VALUES ('ABCDE', CURRENT_TIMESTAMP);

INSERT dbo.Products
    VALUES ('Widget', 5.55),
           ('Thingamajig', 8.88)

INSERT dbo.OrderItems
    VALUES (1, 1, 5.55, 3);

USE OurInsteadOfTest;
GO

-- Создаем представление с параметром WITH SCHEMABINDING
CREATE VIEW CustomerOrders_vw
WITH SCHEMABINDING
AS
SELECT o.OrderID, o.OrderDate, od.ProductID, p.Name, od.Quantity, od.UnitPrice
FROM dbo.Orders AS o JOIN dbo.OrderItems AS od ON o.OrderID = od.OrderID
      JOIN dbo.Products AS p ON od.ProductID = p.ProductID;
GO
```

```
-- Делаем попытку добавить в представление одну запись
INSERT INTO CustomerOrders_vw
    (
     OrderID,
     OrderDate,
     ProductID,
     Quantity,
     UnitPrice
    )
VALUES
    (
     1,
     '1998-04-06',
     2,
     10,
     6.00
    );

-- Попытка завершается неудачей

Msg 4405, Level 16, State 1, Line 1
View or function 'CustomerOrders_vw' is not updatable because the modification affects
multiple base tables.

-- Создаем INSTEAD OF триггер для представления
CREATE TRIGGER trCustomerOrderInsert ON CustomerOrders_vw
INSTEAD OF INSERT
AS
BEGIN
    -- Проверям, на самом ли деле INSERT пытается добавить хотя бы одну строку.
    -- (A WHERE clause might have filtered everything out)
    IF (SELECT COUNT(*) FROM Inserted) > 0
    BEGIN
        INSERT INTO dbo.OrderItems
            SELECT  i.OrderID,
                    i.ProductID,
                    i.UnitPrice,
                    i.Quantity
            FROM Inserted AS i
            JOIN Orders AS o
                ON i.OrderID = o.OrderID;
        -- Если есть записи в псевдотаблице Inserted,
        -- но нет соответствия с таблицей Orders,
        -- то операция вставки в таблицу OrderItems не может быть выполнена.
        IF @@ROWCOUNT = 0
            RAISERROR('No matching Orders. Cannot perform insert',10,1);
    END
END;
GO

-- Делаем еще одну попытку добавить в представление одну запись
INSERT INTO CustomerOrders_vw
    (
     OrderID,
     OrderDate,
     ProductID,
     Quantity,
     UnitPrice
    )
```

```
VALUES
    (
     1,
     '1998-04-06',
     2,
     10,
     6.00
    );

-- Попытка завершается удачей
```