Пользовательские функции

В SQL Server обычно имеется два типа подпрограмм:

- хранимые процедуры;
- определяемые пользователем функции (UDF).

Рассмотрим создание и использование определяемых пользователем функций (User Defined Functions - UDF).

Хранимые процедуры состоят из нескольких инструкций и имеют от нуля до нескольких входных параметров, но обычно не возвращают никаких параметров. В отличие от хранимых процедур, функции всегда возвращают одно значение.

С SQL Server вы можете создавать ваши собственные функции, добавляющие и расширяющие функции, предоставляемые системой. Функции могут получать 0 или более параметров и возвращать скалярное значение или таблицу. Входные параметры могут быть любого типа, исключая timestamp, cursor, table.

Сервер SQL поддерживает три типа функций определенных пользователем:

– Скалярные функции – похожи на встроенные функции;

– Функция, возвращающая таблицу - возвращает результат единичного оператора SELECT. Он похож на объект просмотра, но имеет большую эластичность благодаря использованию параметров, и расширяет возможности индексированного объекта просмотра;

– Многооператорная функция - возвращает таблицу созданную одним или несколькими операторами Transact-SQL, чем напоминает хранимые процедуры. В отличие от процедур, на такие функции можно ссылаться в WHERE как на объект просмотра.

Создание и выполнение определяемых пользователем функций

Определяемые пользователем функции создаются посредством инструкции CREATE FUNCTION, которая имеет следующий синтаксис:

```
CREATE FUNCTION [schema_name.]function_name
   [( {@param } type [= default]) {,...}
      RETURNS {scalar_type | [@variable] TABLE}
   [WITH {ENCRYPTION | SCHEMABINDING}
   [AS] {block | RETURN (select_statement)}
```

Параметр schema_name определяет имя схемы, которая назначается владельцем создаваемой UDF, а параметр function_name определяет имя этой функции. Параметр @param является входным параметром функции (формальным аргументом), чей тип данных определяется параметром type.

Параметры функции - это значения, которые передаются вызывающим объектом определяемой пользователем функции для использования в ней. Параметр default определяет значение по умолчанию для соответствующего параметра функции. (Значением по умолчанию также может быть NULL.)

Предложение RETURNS определяет тип данных значения, возвращаемого UDF. Это может быть почти любой стандартный тип данных, поддерживаемый системой баз данных, включая тип данных TABLE. Единственным типом данных, который нельзя указывать, является тип данных timestamp.

Определяемые пользователем функции могут быть либо скалярными, либо табличными. Скалярные функции возвращают атомарное (скалярное) значение. Это означает, что в предложении RETURNS скалярной функции указывается один из стандартных типов данных. Функция является табличной, если предложение RETURNS возвращает набор строк.

Параметр WITH ENCRYPTION в системном каталоге кодирует информацию, содержащую текст инструкции CREATE FUNCTION. Таким образом, предотвращается несанкционированный просмотр текста, который был использован для создания функции. Данная опция позволяет повысить безопасность системы баз данных.

Альтернативное предложение WITH SCHEMABINDING привязывает UDF к объектам базы данных, к которым эта функция обращается. После этого любая попытка модифицировать объект базы данных, к которому обращается функция, претерпевает неудачу. (Привязка функции к объектам базы данных, к которым она обращается, удаляется только при изменении функции, после чего параметр SCHEMABINDING больше не задан.)

Для того чтобы во время создания функции использовать предложение SCHEMABINDING, объекты базы данных, к которым обращается функция, должны удовлетворять следующим условиям:

– все представления и другие UDF, к которым обращается определяемая функция, должны быть привязаны к схеме;

– все объекты базы данных (таблицы, представления и UDF) должны быть в той же самой базе данных, что и определяемая функция.

Параметр block определяет блок BEGIN/END, содержащий реализацию функции. Последней инструкцией блока должна быть инструкция RETURN с аргументом. (Значением аргумента является возвращаемое функцией значение.) Внутри блока BEGIN/END разрешаются только следующие инструкции:

– инструкции присвоения, такие как SET;

– инструкции для управления ходом выполнения, такие как WHILE и IF;

– инструкции DECLARE, объявляющие локальные переменные;

– инструкции SELECT, содержащие списки столбцов выборки с выражениями, значения которых присваиваются переменным, являющимися локальными для данной функции;

– инструкции INSERT, UPDATE и DELETE, которые изменяют переменные с типом данных TABLE, являющиеся локальными для данной функции.

По умолчанию инструкцию CREATE FUNCTION могут использовать только члены предопределенной роли сервера sysadmin и предопределенной роли базы данных db_owner или db_ddladmin. Но члены этих ролей могут присвоить это право другим пользователям с помощью инструкции GRANT CREATE FUNCTION.

В примере ниже показано создание функции ComputeCosts:

```
USE SampleDb;

-- Эта функция вычисляет возникающие дополнительные общие затраты,
-- при увеличении бюджетов проектов
GO
CREATE FUNCTION ComputeCosts (@percent INT = 10)
   RETURNS DECIMAL(16, 2)
   BEGIN
      DECLARE @addCosts DEC (14,2), @sumBudget DEC(16,2)
      SELECT @sumBudget = SUM (Budget) FROM Project
      SET @addCosts = @sumBudget * @percent/100
      RETURN @addCosts
   END;
```

Функция ComputeCosts вычисляет дополнительные расходы, возникающие при увеличении бюджетов проектов. Единственный входной параметр, @percent, определяет процентное значение увеличения бюджетов. В блоке BEGIN/END сначала объявляются две локальные переменные: @addCosts и @sumBudget, а затем с помощью инструкции SELECT переменной @sumBudget присваивается общая сумма всех бюджетов. После этого функция вычисляет общие дополнительные расходы и посредством инструкции RETURN возвращает это значение.

Вызов определяемой пользователем функции

Определенную пользователем функцию можно вызывать с помощью инструкций Transact-SQL, таких как SELECT, INSERT, UPDATE или DELETE. Вызов функции осуществляется, указывая ее имя с парой круглых скобок в конце, в которых можно задать один или несколько аргументов.

Аргументы - это значения или выражения, которые передаются входным параметрам, определяемым сразу же после имени функции. При

вызове функции, когда для ее параметров не определены значения по умолчанию, для всех этих параметров необходимо предоставить аргументы в том же самом порядке, в каком эти параметры определены в инструкции CREATE FUNCTION.

В примере ниже показан вызов функции ComputeCosts в инструкции SELECT:

```
USE SampleDb;

-- Вернет проект "p2 - Gemini"
SELECT Number, ProjectName
   FROM Project
   WHERE Budget < dbo.ComputeCosts(25);
```

Инструкция SELECT в примере отображает названия и номера всех проектов, бюджеты которых меньше, чем общие дополнительные расходы по всем проектам при заданном значении процентного увеличения.

В инструкциях Transact-SQL имена функций необходимо задавать, используя имена, состоящие из двух частей: schema name и function name, поэтому в примере мы использовали префикс схемы dbo.

Возвращающие табличное значение функции

Как уже упоминалось ранее, функция является возвращающей табличное значение, если ее предложение RETURNS возвращает набор строк. В зависимости от того, каким образом определено тело функции, возвращающие табличное значение функции классифицируются как встраиваемые (inline) и многоинструкционные (multistatement). Если в предложении RETURNS ключевое слово TABLE указывается без сопровождающего списка столбцов, такая функция является встроенной. Инструкция SELECT встраиваемой функции возвращает результирующий набор в виде переменной с типом данных TABLE.

Многоинструкционная возвращающая табличное значение функция содержит имя, определяющее внутреннюю переменную с типом данных TABLE. Этот тип данных указывается ключевым словом TABLE, которое следует за именем переменной. В эту переменную вставляются выбранные строки, и она служит возвращаемым значением функции.

Создание возвращающей табличное значение функции показано в примере ниже:

```
USE SampleDb;

GO
CREATE FUNCTION EmployeesInProject (@projectNumber CHAR(4))
   RETURNS TABLE
   AS RETURN (SELECT FirstName, LastName
```

```
    FROM Works_on, Employee
    WHERE Employee.Id = Works_on.EmpId
       AND ProjectNumber = @projectNumber)
```

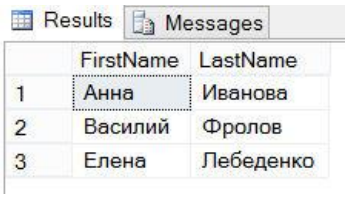
Функция EmployeesInProject отображает имена всех сотрудников, работающих над определенным проектом, номер которого задается входным параметром @projectNumber. Тогда как функция в общем случае возвращает набор строк, предложение RETURNS в определение данной функции содержит ключевое слово TABLE, указывающее, что функция возвращает табличное значение. (Обратите внимание на то, что в примере блок BEGIN/END необходимо опустить, а предложение RETURN содержит инструкцию SELECT.)

Использование функции Employees_in_Project приведено в примере ниже:

```
USE SampleDb;

SELECT *
   FROM EmployeesInProject('p3')
```

Результат выполнения:



	FirstName	LastName
1	Анна	Иванова
2	Василий	Фролов
3	Елена	Лебеденко

Возвращающие табличное значение функции и инструкция APPLY

Реляционная инструкция APPLY позволяет вызывать возвращающую табличное значение функцию для каждой строки табличного выражения. Эта инструкция задается в предложении FROM соответствующей инструкции SELECT таким же образом, как и инструкция JOIN. Инструкция APPLY может быть объединена с табличной функцией для получения результата, похожего на результирующий набор операции соединения двух таблиц. Существует две формы инструкции APPLY:

- CROSS APPLY
- OUTER APPLY

Инструкция CROSS APPLY возвращает те строки из внутреннего (левого) табличного выражения, которые совпадают с внешним (правым) табличным выражением. Таким образом, логически, инструкция CROSS APPLY функционирует так же, как и инструкция INNER JOIN.

Инструкция OUTER APPLY возвращает все строки из внутреннего (левого) табличного выражения. (Для тех строк, для которых нет совпадений

во внешнем табличном выражении, он содержит значения NULL в столбцах внешнего табличного выражения.) Логически, инструкция OUTER APPLY эквивалентна инструкции LEFT OUTER JOIN.

Применение инструкции APPLY показано в примерах ниже:

```
USE SampleDb;

GO
-- Создать функцию
CREATE FUNCTION GetJob (@empid AS INT)
   RETURNS TABLE AS
   RETURN
      SELECT Job
      FROM Works_on
      WHERE EmpId = @empid
         AND Job IS NOT NULL
         AND ProjectNumber = 'p1';
```

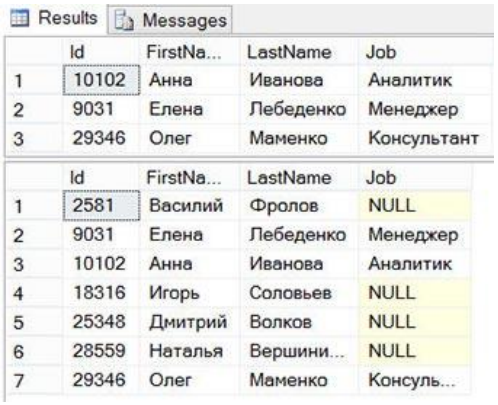
Функция GetJob() возвращает набор строк с таблицы Works_on. В примере ниже этот результирующий набор "соединяется" предложением APPLY с содержимым таблицы Employee:

```
USE SampleDb;

-- Используется CROSS APPLY
SELECT E.Id, FirstName, LastName, Job
   FROM Employee as E
   CROSS APPLY GetJob(E.Id) AS A

-- Используется OUTER APPLY
SELECT E.Id, FirstName, LastName, Job
   FROM Employee as E
   OUTER APPLY GetJob(E.Id) AS A
```

Результатом выполнения этих двух функций будут следующие две таблицы (отображаются после выполнения второй функции):



Results | Messages

	Id	FirstNa...	LastName	Job
1	10102	Анна	Иванова	Аналитик
2	9031	Елена	Лебеденко	Менеджер
3	29346	Олег	Маменко	Консультант

	Id	FirstNa...	LastName	Job
1	2581	Василий	Фролов	NULL
2	9031	Елена	Лебеденко	Менеджер
3	10102	Анна	Иванова	Аналитик
4	18316	Игорь	Соловьев	NULL
5	25348	Дмитрий	Волков	NULL
6	28559	Наталья	Вершини...	NULL
7	29346	Олег	Маменко	Консуль...

В первом запросе примера результирующий набор табличной функции GetJob() "соединяется" с содержимым таблицы Employee посредством инструкции CROSS APPLY. Функция GetJob() играет роль правого ввода, а таблица Employee - левого. Выражение правого ввода вычисляется для каждой строки левого ввода, а полученные строки комбинируются, создавая конечный результат.

Второй запрос похожий на первый (но в нем используется инструкция OUTER APPLY), который логически соответствует операции внешнего соединения двух таблиц.

Возвращающие табличное значение параметры

Во всех версиях сервера, предшествующих SQL Server 2008, задача передачи подпрограмме множественных параметров была сопряжена со значительными сложностями. Для этого сначала нужно было создать временную таблицу, вставить в нее передаваемые значения, и только затем можно было вызывать подпрограмму. Начиная с версии SQL Server 2008, эта задача упрощена, благодаря возможности использования возвращающих табличное значение параметров, посредством которых результирующий набор может быть передан соответствующей подпрограмме.

Использование возвращающего табличное значение параметра показано в примере ниже:

```
USE SampleDb;

CREATE TYPE departmentType AS TABLE
   (Number CHAR(4), DepartmentName CHAR(40), Location CHAR(40));
GO

CREATE TABLE #moscowTable
   (Number CHAR(4), DepartmentName CHAR(40), Location CHAR(40));
GO

CREATE PROCEDURE InsertProc
   @Moscow departmentType READONLY
```

```
    AS SET NOCOUNT ON
        INSERT INTO #moscowTable (Number, DepartmentName, Location)
        SELECT * FROM @Moscow
GO

DECLARE @Moscow AS departmentType;

INSERT INTO @Moscow (Number, DepartmentName, Location)
    SELECT * FROM department
        WHERE location = 'Москва';

EXEC InsertProc @Moscow;
```

В этом примере сначала определяется табличный тип departmentType. Это означает, что данный тип является типом данных TABLE, вследствие чего он разрешает вставку строк. В процедуре InsertProc объявляется переменная @Moscow с типом данных departmentType. (Предложение READONLY указывает, что содержимое этой таблицы нельзя изменять.) В последующем пакете в эту табличную переменную вставляются данные, после чего процедура запускается на выполнение. В процессе исполнения процедура вставляет строки из табличной переменной во временную таблицу #moscowTable. Вставленное содержимое временной таблицы выглядит следующим образом:

Results | Messages

	Number	DepartmentName	Location
1	d1	Исследования	Москва
2	d3	Маркетинг	Москва

Использование возвращающих табличное значение параметров предоставляет следующие преимущества:

– упрощается модель программирования подпрограмм;

– уменьшается количество обращений к серверу и получений соответствующих ответов;

– таблица результата может иметь произвольное количество строк.

Изменение структуры определяемых пользователями инструкций

Язык Transact-SQL также поддерживает инструкцию ALTER FUNCTION, которая модифицирует структуру определяемых пользователями инструкций (UDF). Эта инструкция обычно используется для удаления привязки функции к схеме. Все параметры инструкции ALTER FUNCTION имеют такое же значение, как и одноименные параметры инструкции CREATE FUNCTION.

Для удаления UDF применяется инструкция DROP FUNCTION. Удалить функцию может только ее владелец или член предопределенной роли db_owner или sysadmin.

Ограничения

– Определяемые пользователем функции не могут выполнять действия, изменяющие состояние базы данных.

– Определяемые пользователем функции не могут содержать предложение OUTPUT INTO, целью которого является таблица.

– Определяемые пользователем функции не могут возвращать несколько результирующих наборов. Используйте хранимую процедуру, если нужно возвращать несколько результирующих наборов.

– Обработка ошибок в функциях, определяемых пользователем, ограниченна. UDF не поддерживает тип TRY...CATCH, @ERROR и RAISERROR.

– Определяемые пользователем функции не могут вызывать хранимую процедуру, но могут вызывать расширенную хранимую процедуру.

– Определяемые пользователем функции не могут использовать динамический SQL и временные таблицы. Табличные переменные разрешены к использованию.

– Инструкцию SET нельзя использовать в определяемых пользователем функциях.

– Пустое предложение FOR XML запрещено.

– Определяемые пользователем функции могут быть вложенными, то есть из одной функции может быть вызвана другая. Уровень вложенности увеличивается на единицу каждый раз, когда начинается выполнение вызванной функции и уменьшается на единицу, когда ее выполнение завершается. Вложенность определяемых пользователем функций не может превышать 32 уровней. Превышение максимального уровня вложенности приводит к ошибке выполнения для всей цепочки вызываемых функций. Каждый вызов управляемого кода из определяемой пользователем функции Transact-SQL считается одним уровнем вложенности из 32 возможных. Методы, вызываемые из управляемого кода, под это ограничение не подпадают.

– Следующие инструкции компонента Service Broker не могут быть включены в определение пользовательской функции Transact-SQL:

- BEGIN DIALOG CONVERSATION
- END CONVERSATION
- GET CONVERSATION GROUP
- MOVE CONVERSATION
- RECEIVE
- SEND

Примеры

1 Рассмотрим пример создания функции в SQL Server (Transact-SQL). Ниже приведен простой пример функции:

```
CREATE FUNCTION ReturnSite
( @site_id INT )
RETURNS VARCHAR(50)
AS
BEGIN
  DECLARE @site_name VARCHAR(50);
  IF @site_id < 10
    SET @site_name = 'yandex.com';
  ELSE
    SET @site_name = 'google.com';
  RETURN @site_name;
END;
```

Эта функция называется ReturnSite. Она имеет один параметр, называемый @site_id, который является типом данных INT. Функция возвращает значение VARCHAR (50), указанное в предложении RETURNS.

Затем вы можете ссылаться на новую функцию ReturnSite следующим образом:

```
USE [test]
GO
SELECT dbo.ReturnSite(8);
GO
```

2 Результатом следующего примера является встроенная функция, возвращающая табличное значение (TVF), в базе данных AdventureWorks2012. Функция имеет один входной параметр — идентификатор клиента (магазина) — и возвращает столбцы ProductID, Nameи столбец YTD Total со сведениями о продажах продукта за текущий год.

```
IF OBJECT_ID (N'Sales.ufn_SalesByStore', N'IF') IS NOT NULL
   DROP FUNCTION Sales.ufn_SalesByStore;
GO
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
(
   SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'
```

```
    FROM Production.Product AS P
    JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
    JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID
    JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
    WHERE C.StoreID = @storeid
    GROUP BY P.ProductID, P.Name
);
```

3 В следующем примере создается **скалярная функция (скалярная UDF)** из нескольких инструкций в базе данных AdventureWorks2012. Функция имеет один входной параметр ProductIDи возвращает одно значение — количество указанного товара на складе.

```
IF OBJECT_ID (N'dbo.ufnGetInventoryStock', N'FN') IS NOT NULL
    DROP FUNCTION ufnGetInventoryStock;
GO
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
-- Returns the stock level for the product.
BEGIN
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID
        AND p.LocationID = '6';
     IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
```

Далее функция ufnGetInventoryStock используется для получения сведений о количестве товаров с идентификаторами ProductModelID от 75 до 80.

```
SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS
CurrentSupply
FROM Production.Product
WHERE ProductModelID BETWEEN 75 and 80;
```

Список источников

– https://professorweb.ru/my/sql-server/2012/level3/3_3.php
– https://oracleplsql.ru/functions-sql-server.html
– https://docs.microsoft.com/ru-ru/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine?view=sql-server-ver15