Базы данных

14 | Пользовательские типы. Функции.

План

- Пользовательские типы
- Скалярные функции
- Табличные функции

Пользовательские типы (UDT)

Вводятся при недостаточной выразительности встроенных в СУБД стандартных типах.

Виды:

- Псевдоним – на базе стандартного типа;
- Определяемый пользователем тип (.Net CLR);
- Определяемый пользователем табличный тип.

https://docs.microsoft.com/ru-ru/sql/relational-databases/clr-integration-database-objects-user-defined-types/clr-user-defined-types?view=sql-server-ver15

Пользовательские типы–псевдонимы

```
-- создаем тип для номера телефона
CREATE TYPE [dbo].[Phone] FROM [nvarchar](25) NULL

-- определение таблицы
CREATE TABLE [SalesLT].[Customer](
[CustomerID] [int] IDENTITY(1,1) NOT NULL,
[Phone] [dbo].[Phone] NULL,
...
)
```

https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-type-transact-sql?view=sql-server-ver15

Пользовательские типы .Net CLR и не только…

- Создаем проект в Visual Studio (SQL Server Database Project);
- Реализуем в проекте типы, функции, триггеры, хранимые процедуры и получаем DLL-сборку (assembly);
- Включаем SQLCLR на SQL Server и загружаем в него сборку;
- Создаем объекты в БД и указываем, что они реализованы в загруженной сборке.

```
CREATE ASSEMBLY utf8string AUTHORIZATION [dbo] FROM
0x4D... ;
GO
CREATE TYPE Utf8String EXTERNAL NAME
utf8string.[Microsoft.Samples.SqlServer.utf8string]
```

https://docs.microsoft.com/ru-ru/sql/relational-databases/clr-integration/assemblies/creating-an-assembly?view=sql-server-ver15

Пользовательские табличные типы

```
-- создаем табличный тип
CREATE TYPE ErrMsgTableType AS TABLE (ErrorNumber INT, ErrorMessage nvarchar(4000));
GO
-- создаем хранимую процедуру для получения данных из табличной переменной
CREATE PROCEDURE dbo.usp_LogSomeErrors
@TVP ErrMsgTableType READONLY
AS
INSERT INTO dbo.ErrorLog
(ErrorTime, UserName, ErrorNumber, ErrorMessage)
SELECT GETDATE(), CURRENT_USER, ErrorNumber, ErrorMessage FROM @TVP;
GO
-- создаем табличную переменную
DECLARE @err AS ErrMsgTableType;
-- заполняем ее данными
INSERT INTO @err (ErrorNumber, ErrorMessage)
    SELECT 0,'Ошибка0' UNION ALL
SELECT 1,'Ошибка1' UNION ALL
SELECT 2,'Ошибка2';
-- передаем в хранимую процедуру
EXEC dbo.usp_LogSomeErrors @err;
-- проверяем, что получилось в логе
select top 5 * from dbo.ErrorLog ORDER BY ErrorTime desc
```

DEMO

Работа с пользовательскими типами

Типы пользовательских функций (UDF)

Три типа функций:

- Скалярные;
  - Возвращают одно значение;
  - Вычисляются для каждой строки, если используются в SELECT;
- Встраиваемые (inline) табличные;
  - Возвращают переменную табличного типа;
  - Одиночный оператор SELECT определяет возвращаемую таблицу;
- Многооператорные (multi-statement) табличные;
  - Используется несколько операторов для формирования возвращаемой таблицы.

https://docs.microsoft.com/ru-ru/sql/relational-databases/user-defined-functions/user-defined-functions?view=sql-server-ver15

Ограничения пользовательских функций (UDF)

- не могут выполнять действия, изменяющие состояние БД;
- не могут содержать предложение OUTPUT INTO <таблица>;
- не могут возвращать несколько результирующих наборов;
- нет поддержки TRY…CATCH, @@ERROR и RAISERROR;
- нельзя вызывать хранимую процедуру (можно расшир. проц.);
- нельзя использовать динамический SQL и временные таблицы;
- нельзя использовать SET <option> и FOR XML;
- вложенность UDF не может превышать 32 уровня.

https://docs.microsoft.com/ru-ru/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine?view=sql-server-ver15

Соглашения в пользовательских функциях

Основные соглашения при использовании UDF:

- Нельзя использовать недетерминированные функции (например, getdate());
- Обязательно надо указывать имя схемы.

Создание системных функций:

- Создайте функцию в БД master;
- Измените владельца на system_function_schema:

```
sp_changeobjectowner
'fn_someFunc', 'system_function_schema'
```

Использование пользовательских функций

Основные варианты использования:

- помогают упростить выбираемые значения в SELECT;
- улучшают надежность и читаемость кода уменьшая количество соединений и инкапсулируя запросы;
- могут использоваться для получения агрегированных значений.

Пример скалярной функции:

```
CREATE FUNCTION [dbo].[ufnGetOrdersCount](@CustomerID int)
RETURNS int
AS
BEGIN
      DECLARE @ret int
      select @ret=count(SalesOrderID) from SalesLT.SalesOrderHeader
            WHERE CustomerID=@CustomerID
      RETURN @ret
END
GO

-- примеры использования
print [dbo].[ufnGetOrdersCount](1)-- 0
select [dbo].[ufnGetOrdersCount](30113)as [Count]-- 1
```

Пример встраиваемой табличной функции:

```
CREATE FUNCTION [dbo].[ufnGetOrderedProducts](@CustomerID int)
RETURNS TABLE
AS           -- Возвращаем все заказанные клиентом товары
RETURN (
    SELECT c.CustomerID, c.FirstName, c.LastName,
       h.SalesOrderID,h.SalesOrderNumber,
       p.[Name] as ProductName, p.ListPrice,
       d.UnitPrice as OrderedPrice,
       [dbo].[ufnGetOrdersCount](c.CustomerID) as OrdersCount
    FROM SalesLT.SalesOrderDetail as d
    INNER JOIN SalesLT.Product as p ON p.ProductID=d.ProductID
    INNER JOIN SalesLT.SalesOrderHeader as h ON d.SalesOrderID=h.SalesOrderID
    INNER JOIN SalesLT.Customer as c ON c.CustomerID=h.CustomerID
    WHERE h.CustomerID = @CustomerID    -- ORDER BY => ERROR
)
GO

-- пример использования табличной функции
select * from [dbo].[ufnGetOrderedProducts](30113)
```

Пример многооператорной табличной функции:

```
CREATE FUNCTION [dbo].[ufnGetAllCategories]()
RETURNS @retCategoryInformation TABLE (    --Структура возвращаемой таблицы
       [ParentProductCategoryName] [nvarchar](50) NULL, [ProductCategoryName] [nvarchar](50)
NOT NULL,[ProductCategoryID] [int] NOT NULL
) AS BEGIN
   WITH CategoryCTE([ParentProductCategoryID], [ProductCategoryID], [Name]) AS (
     SELECT [ParentProductCategoryID], [ProductCategoryID], [Name]
       FROM SalesLT.ProductCategory WHERE ParentProductCategoryID IS NULL
     UNION ALL
       SELECT C.[ParentProductCategoryID], C.[ProductCategoryID], C.[Name]
       FROM SalesLT.ProductCategory AS C
       INNER JOIN CategoryCTE AS BC ON BC.ProductCategoryID = C.ParentProductCategoryID
)
INSERT INTO @retCategoryInformation        SELECT PC.[Name] AS [ParentProductCategoryName],
CCTE.[Name] as [ProductCategoryName], CCTE.[ProductCategoryID]
FROM CategoryCTE AS CCTE INNER JOIN SalesLT.ProductCategory AS PC ON
PC.[ProductCategoryID]=CCTE.[ParentProductCategoryID];
RETURN;
END;

-- пример использования
select * FROM [dbo].[ufnGetAllCategories]()
```

DEMO

Работа с функциями

Изучено

- Пользовательские типы
- Скалярные функции
- Табличные функции