Функции, процедуры, триггеры, курсоры

Функции

По поведению:

- Функция является детерминированной, если при одном и том же заданном входном значении она всегда возвращает один и тот же результат.
- Функция является недетерминированной, если она может возвращать различные значения при одном и том же заданном входном значении.

Функции

По типу возвращаемого значения:

- Скалярная функция;
- Подставляемая табличная функция;
- Многооператорная табличная функция.

Скалярная функция

Синтаксис:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS скалярный-тип-данных
[ WITH список-опций-функций ]
[ AS ]
BEGIN
    тело-функции
    RETURN скалярное-выражение
END [ ; ]
```

Пример:

```
CREATE FUNCTION dbo.AveragePrice() RETURNS smallmoney
WITH SCHEMABINDING AS
BEGIN
    RETURN (SELECT AVG(Price) FROM dbo.P)
END;

CREATE FUNCTION dbo.PriceDifference(@Price smallmoney)
RETURNS smallmoney AS
BEGIN
    RETURN @Price - dbo.AveragePrice()
END;

— Вызов функции
SELECT Pname, Price, dbo.AveragePrice() AS Average,
        dbo.PriceDifference(Price) AS Difference
FROM P
WHERE City='Смоленск'
```

Подставляемая табличная функция

Синтаксис:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS TABLE
[ WITH список-опций-функций ]
[ AS ]
RETURN [ ( ] выражение-выборки [ ) ]
END [ ; ]
```

Пример:

```
CREATE FUNCTION dbo.FullSPJ(@Pno int)
RETURNS TABLE
AS
   RETURN (
      SELECT S.Sname, P.Pname, J.Jname, SPJ.Qty
      FROM S INNER JOIN SPJ ON S.Sno=SPJ.Sno
                  INNER JOIN P ON P.Pno=SPJ.Pno
                  INNER JOIN J ON J.Jno=SPJ.Jno
     WHERE P.Pno = @Pno;
)

— Вызов функции
SELECT * FROM dbo.FullSPJ(4)
```

Многооператорная функция

Синтаксис:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS @имя-возвращаемой-переменной
TABLE определение-таблицы
[ WITH список-опций-функций ]
[ AS ]
BEGIN
RETURN
END [ ; ]
```

Пример:

```
CREATE FUNCTION dbo.fnGetReports (@EmployeeID AS int )
RETURNS @Reports TABLE ( EmployeeID int NOT NULL,
                         ReportsToID int NULL )
AS
BEGIN
    DECLARE  @Employee int;

    INSERT INTO @Reports
    SELECT EmployeeID, ReportsTo FROM Employees
    WHERE EmployeeID = @EmployeeID;

    SELECT @Employee = MIN(EmployeeID) FROM Employees
    WHERE ReportsTo = @EmployeeID;

    WHILE @Employee IS NOT NULL
    BEGIN
      INSERT INTO @Reports
      SELECT * FROM Employees;
    END
    RETURN
END;
```

Хранимая процедура

Синтаксис:

```
CREATE PROCEDURE [ имя-схемы. ] имя-процедуры [ список-объявлений-параметров ]
[ WITH список-опций-процедуры ]
[ FOR REPLICATION ]
AS
тело-процедуры
[ ;]
```

Пример:

```
CREATE PROCEDURE dbo.Factorial @ValIn bigint, @ValOut bigint output
AS
BEGIN
	IF @ValIn > 20 BEGIN
		PRINT N'Входной параметр должен быть <= 20'
		RETURN -99
	END
	DECLARE @WorkValIn bigint, @WorkValOut bigint
	IF @ValIn != 1
	BEGIN
		SET @WorkValIn = @ValIn - 1
		PRINT @@NESTLEVEL
		EXEC dbo.Factorial @WorkValIn, @WorkValOut OUTPUT
		SET @ValOut = @WorkValOut * @ValIn
	 END
	 ELSE
		 SET @ValOut = 1
 END
```

Хранимая процедура

Вызов хранимой процедуры:

```
DECLARE @FactIn int, @FactOut int
SET @FactIn = 8
EXEC dbo.Factorial @FactIn, @FactOut OUTPUT

PRINT N'Факториал ' + CONVERT(varchar(3), @FactIn) +
        N’ равен ' + CAST(@FactOut AS varchar(20))
```

Триггеры

Триггер - это хранимая процедура особого типа, которая выполняет одну или несколько инструкций в ответ на событие.

По типу события триггеры деляться на два класса:

- DDL-триггеры
- DML-триггеры

DDL-триггеры

Триггер DDL может активироваться, если выполняется такая инструкция, как ALTER SERVER CONFIGURATION, или если происходит удаление таблицы с использованием команды DROP TABLE.

Пример триггера:

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'You must disable Trigger "safety" to drop or alter tables!'
    ROLLBACK;
```

DML-триггеры

Для поддержания согласованности и точности данных используются декларативные и процедурные методы.

Триггеры применяются в следующих случаях:

- если использование методов декларативной целостности данных не отвечает функциональным потребностям приложения;
- если необходимо каскадное изменение через связанные таблицы в базе данных;
- если база данных денормализована и требуется способ автоматизированного обновления избыточных данных в нескольких таблицах;
- если необходимо сверить значение в одной таблице с неидентичным значением в другой таблице;
- если требуется вывод пользовательских сообщений и сложная обработка ошибок.

Классы DML-триггеров

Существуют два класса триггеров:

- INSTEAD OF. Триггеры этого класса выполняются в обход действий, вызывавших их срабатывание, заменяя эти действия. Например, обновление таблицы, в которой есть триггер INSTEAD OF, вызовет срабатывание этого триггера. В результате вместо оператора обновления выполняется код триггера.
- AFTER/BEFORE. Триггеры этого класса исполняются после или до действия, вызвавшего срабатывание триггера. Они считаются классом триггеров по умолчанию.

DML-триггеры

Синтаксис:

```
CREATE TRIGGER имя_триггера
ON имя_таблицы_или_представления
[ WITH ENCRYPTION ]
класс_триггера тип(ы)_триггера
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS sql_инструкции
```

Пример:

```
CREATE TRIGGER AfterUpdateSPJ
ON dbo.SPJ
AFTER UPDATE
AS
BEGIN
RAISERROR(N'Произошло обновление в таблице поставок',1,1)
END
```

Курсоры

Операции в реляционной базе данных выполняются над множеством строк. Набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE.

Курсоры можно классифицировать:

- По области видимости;
- По типу;
- По способу перемещения по курсору;
- По способу распараллеливания курсора

По области видимости

По области видимости имени курсора различают:

- Локальные курсоры (LOCAL). Область курсора локальна по отношению к пакету, хранимой процедуре или триггеру, в которых этот курсор был создан. Курсор неявно освобождается после завершения выполнения пакета, хранимой процедуры или триггера, за исключением случая, когда курсор был передан параметру OUTPUT.
- Глобальные курсоры (GLOBAL). Область курсора является глобальной по отношению к соединению.

По типу

По типу курсора различают:

- Статические курсоры (STATIC). Создается временная копия данных для использования курсором. Все запросы к курсору обращаются к указанной временной таблице в базе данных tempdb, поэтому изменения базовых таблиц не влияют на данные, возвращаемые выборками для данного курсора, а сам курсор не позволяет производить изменения.
- Динамические курсоры (DYNAMIC). Отображают все изменения данных, сделанные в строках результирующего набора при просмотре этого курсора. Значения данных, порядок, а также членство строк в каждой выборке могут меняться. Параметр выборки ABSOLUTE динамическими курсорами не поддерживается.
- Курсоры, управляемые набором ключей (KEYSET). Членство или порядок строк в курсоре не изменяются после его открытия. Набор ключей, однозначно определяющих строки, встроен в таблицу в базе данных tempdb с именем keyset.
- Быстрые последовательные курсоры (FAST_FORWARD). Параметр FAST_FORWARD указывает курсор FORWARD_ONLY, READ_ONLY, для которого включена оптимизация производительности. Параметр FAST_FORWARD не может указываться вместе с параметрами SCROLL или FOR_UPDATE.

По способу перемещения по курсору

- Последовательные курсоры (FORWARD_ONLY). Курсор может просматриваться только от первой строки к последней. Поддерживается только параметр выборки FETCH NEXT. Если параметр FORWARD_ONLY указан без ключевых слов STATIC, KEYSET или DYNAMIC, то курсор работает как DYNAMIC.
- Курсоры прокрутки (SCROLL). Перемещение осуществляется по группе записей как вперед, так и назад. В этом случае доступны все параметры выборки (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Параметр SCROLL не может указываться вместе с параметром для FAST_FORWARD.

По способу распараллеливания курсора

По способу распараллеливания курсоров различают:

- READ_ONLY. Содержимое курсора можно только считывать.
- SCROLL_LOCKS. При редактировании данной записи вами никто другой вносить в нее изменения не может. Такую блокировку прокрутки иногда еще называют «пессимистической» блокировкой.
- OPTIMISTIC. Означает отсутствие каких бы то ни было блокировок. «Оптимистическая» блокировка предполагает, что даже во время выполнения вами редактирования данных, другие пользователи смогут к ним обращаться.

Курсоры

Синтаксис:

```
DECLARE имя-курсора CURSOR
[ область-видимости-имени-курсора ]
[ возможность-перемещения-по-курсору ]
[ типы-курсоров ]
[ опции-распараллеливания-курсоров ]
[ выявление-ситуаций-с-преобразованием-типа-курсора ]
FOR инструкция_select
[ опция-FOR-UPDATE ]
```

Пример:

```
DECLARE @MyVariable CURSOR

DECLARE @MyCursor CURSOR
FOR SELECT LastName FROM
AdventureWorks.Person.Contact

SET @MyVariable = MyCursor;
```

Курсоры

Основой всех операций прокрутки курсора является ключевое слово FETCH. В качестве аргументов оператора FETCH могут выступать:

- NEXT – возвращает строку результата сразу же за текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH NEXT выполняет первую выборку в отношении курсора, она возвращает первую строку в результирующем наборе. NEXT является параметром по умолчанию выборки из курсора.
- PRIOR – возвращает строку результата, находящуюся непосредственно перед текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH PRIOR выполняет первую выборку из курсора, не возвращается никакая строка и положение курсора остается перед первой строкой.
- FIRST – возвращает первую строку в курсоре и делает ее текущей.
- LAST – возвращает последнюю строку в курсоре, и делает ее текущей.

Пример:

```
-- Объявляем курсор
DECLARE CursorTest CURSOR
GLOBAL SCROLL STATIC FOR
      SELECT OrderID, CustomerID
      FROM CursorTable

-- Объявляем переменные для хранения
DECLARE @OrderID int
DECLARE @CustomerID varchar(5)

-- Откроем курсор и запросим первую запись
OPEN CursorTest
FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID --
Обработаем в цикле все записи курсора
WHILE @@FETCH_STATUS=0
BEGIN
      PRINT CONVERT(varchar(5),@OrderID) + ' ' + @CustomerID
      FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END

CLOSE CursorTest
```