Триггеры

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE.

Триггером называют процедуру SQL, выполняемую автоматически, когда происходит некоторое событие, которое называется триггерным событием.

К примеру, мы можем писать триггеры, которые срабатывают в процессе выполнения над таблицей операций UPDATE, INSERT либо DELETE; во время выдачи команд DDL; во время входа пользователя в систему либо его выхода; во время запуска либо остановки БД; во время возникновения ошибок.

Существуют 3 различия между процедурами SQL и триггерами:

1. Триггеры невозможно вызвать из кода программы. Они вызываются автоматически как ответ на некоторое событие.
2. У триггеров нет списка параметров.
3. Спецификации триггера и процедуры немного отличаются.

Триггер - это механизм, который вызывается, когда в указанной таблице происходит определенное действие. Каждый триггер имеет следующие основные составляющие: имя, действие и исполнение. Имя триггера может содержать максимум 128 символов. Действием триггера может быть или инструкция DML (INSERT, UPDATE или DELETE), или инструкция DDL. Таким образом, существует два типа триггеров: триггеры DML и триггеры DDL. Исполнительная составляющая триггера обычно состоит из хранимой процедуры или пакета.

Формальное определение триггера

Синтаксис триггера DML:

```
CREATE TRIGGER имя_триггера
ON {имя_таблицы | имя_представления}
{AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]
AS выражения_sql
```

Для создания триггера применяется выражение CREATE TRIGGER, после которого идет имя триггера. Как правило, имя триггера отражает тип операций и имя таблицы, над которой производится операция.

Каждый триггер ассоциируется с определенной таблицей или представлением, имя которых указывается после слова ON.

Затем устанавливается тип триггера. Мы можем использовать один из двух типов:

AFTER: выполняется после выполнения действия. Определяется только для таблиц.

INSTEAD OF: выполняется вместо действия (то есть по сути действие - добавление, изменение или удаление - вообще не выполняется). Определяется для таблиц и представлений

После типа триггера идет указание операции, для которой определяется триггер: INSERT, UPDATE или DELETE.

Для триггера AFTER можно применять сразу для нескольких действий, например, UPDATE и INSERT. В этом случае операции указываются через запятую. Для триггера INSTEAD OF можно определить только одно действие.

И затем после слова AS идет набор выражений SQL, которые собственно и составляют тело триггера.

Создадим триггер. Допустим, у нас есть база данных productsdb со следующим определением:

```
CREATE DATABASE productdb;
GO

USE productdb;
CREATE TABLE Products
(
   Id INT IDENTITY PRIMARY KEY,
   ProductName NVARCHAR(30) NOT NULL,
   Manufacturer NVARCHAR(20) NOT NULL,
   ProductCount INT DEFAULT 0,
   Price MONEY NOT NULL
);
```

Определим триггер, который будет срабатывать при добавлении и обновлении данных:

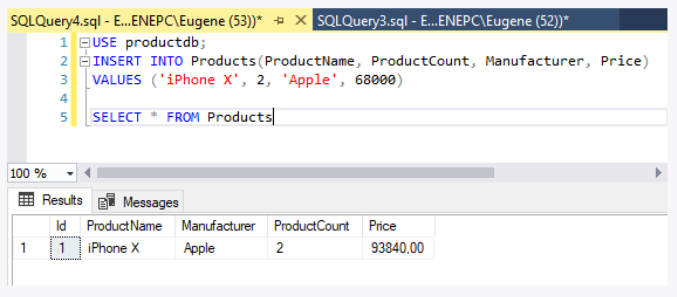
```
USE productdb;
GO
CREATE TRIGGER Products_INSERT_UPDATE
ON Products
AFTER INSERT, UPDATE
AS
UPDATE Products
SET Price = Price + Price * 0.38
```

WHERE Id = (SELECT Id FROM inserted)

Допустим, в таблице Products хранятся данные о товарах. Но цена товара нередко содержит различные надбавки типа налога на добавленную стоимость, налога на добавленную коррупцию и так далее. Человек, добавляющий данные, может не знать все эти тонкости с налоговой базой, и он определяет чистую цену. С помощью триггера мы можем поправить цену товара на некоторую величину.

Таким образом, триггер будет срабатывать при любой операции INSERT или UPDATE над таблицей Products. Сам триггер будет изменять цену товара, а для получения того товара, который был добавлен или изменен, находим этот товар по Id. Но какое значение должен иметь Id такой товар? Дело в том, что при добавлении или изменении данные сохраняются в промежуточную таблицу inserted. Она создается автоматически. И из нее мы можем получить данные о добавленных/измененных товарах.

И после добавления товара в таблицу Products в реальности товар будет иметь несколько большую цену, чем та, которая была определена при добавлении:



Удаление триггера

Для удаления триггера необходимо применить команду DROP TRIGGER:

```
DROP TRIGGER Products_INSERT_UPDATE
```

Отключение триггера

Бывает, что мы хотим приостановить действие триггера, но удалять его полностью не хотим. В этом случае его можно временно отключить с помощью команды DISABLE TRIGGER:

```
DISABLE TRIGGER Products_INSERT_UPDATE ON Products
```

А когда триггер понадобится, его можно включить с помощью команды ENABLE TRIGGER:

```
ENABLE TRIGGER Products_INSERT_UPDATE ON Products
```

Области применения DML-триггеров

Такие триггеры применяются для решения разнообразных задач. Рассмотрим несколько областей применения триггеров DML, в частности триггеров AFTER и INSTEAD OF.

Триггеры AFTER

Триггеры AFTER вызываются после того, как выполняется действие, запускающее триггер. Триггер AFTER задается с помощью ключевого слова AFTER или FOR. Триггеры AFTER можно создавать только для базовых таблиц. Триггеры этого типа можно использовать для выполнения, среди прочих, следующих операций:

- создания журнала логов действий в таблицах базы данных;
- реализации бизнес-правил;
- принудительного обеспечения ссылочной целостности.

Создание журнала логов

В SQL Server можно выполнять отслеживание изменения данных, используя систему перехвата изменения данных CDC (change data capture). Эту задачу можно также решить с помощью триггеров DML. В примере ниже показывается, как с помощью триггеров можно создать журнал логов действий в таблицах базы данных:

```
USE SampleDb;

    /* Таблица AuditBudget используется в качестве
    журнала логов действий в таблице Project */
GO
CREATE TABLE AuditBudget (
        ProjectNumber CHAR(4) NULL,
        UserName CHAR(16) NULL,
        Date DATETIME NULL,
        BudgetOld FLOAT NULL,
        BudgetNew FLOAT NULL
    );

GO
CREATE TRIGGER trigger_ModifyBudget
        ON Project AFTER UPDATE
        AS IF UPDATE(budget)
    BEGIN
```

```
        DECLARE @budgetOld FLOAT
        DECLARE @budgetNew FLOAT
        DECLARE @projectNumber CHAR(4)

        SELECT @budgetOld = (SELECT Budget FROM deleted)
        SELECT @budgetNew = (SELECT Budget FROM inserted)
        SELECT @projectNumber = (SELECT Number FROM deleted)

        INSERT INTO AuditBudget VALUES
          (@projectNumber, USER_NAME(), GETDATE(), @budgetOld,
@budgetNew)
    END
```

В этом примере создается таблица AuditBudget, в которой сохраняются все изменения столбца Budget таблицы Project. Изменения этого столбца будут записываться в эту таблицу посредством триггера trigger_ModifyBudget.

Этот триггер активируется для каждого изменения столбца Budget с помощью инструкции UPDATE. При выполнении этого триггера значения строк таблиц deleted и inserted присваиваются соответствующим переменным @budgetOld, @budgetNew и @projectNumber. Эти присвоенные значения, совместно с именем пользователя и текущей датой, будут затем вставлены в таблицу AuditBudget.

В этом примере предполагается, что за один раз будет обновление только одной строки. Поэтому этот пример является упрощением общего случая, когда триггер обрабатывает многострочные обновления. Если выполнить следующие инструкции Transact-SQL:

```
USE SampleDb;

UPDATE Project
        SET Budget = 200000
        WHERE Number = 'p2';
```

то содержимое таблицы AuditBudget будет таким:

Results | Messages

	ProjectNumber	UserName	Date	BudgetOld	BudgetNew
1	p2	dbo	2015-05-20 15:28:36.623	95000	200000

Реализация бизнес-правил

С помощью триггеров можно создавать бизнес-правила для приложений. Создание такого триггера показано в примере ниже:

```
USE SampleDb;

        -- Триггер trigger_TotalBudget является примером использования
        -- триггера для реализации бизнес-правила
GO
CREATE TRIGGER trigger_TotalBudget
          ON Project AFTER UPDATE
          AS IF UPDATE (Budget)
BEGIN
          DECLARE @sum_old1
          FLOAT DECLARE @sum_old2
          FLOAT DECLARE @sum_new FLOAT

          SELECT @sum_new = (SELECT SUM(Budget) FROM inserted)
          SELECT @sum_old1 = (SELECT SUM(p.Budget)
           FROM project p WHERE p.Number
             NOT IN (SELECT d.Number FROM deleted d))
          SELECT @sum_old2 = (SELECT SUM(Budget) FROM deleted)

          IF @sum_new > (@sum_old1 + @sum_old2) * 1.5
          BEGIN
             PRINT 'Бюджет не изменился'
             ROLLBACK TRANSACTION
          END
          ELSE
             PRINT 'Изменение бюджета выполнено'
        END
```

Здесь создается правило для управления модификацией бюджетов проектов. Триггер trigger_TotalBudget проверяет каждое изменение бюджетов и выполняет только такие инструкции UPDATE, которые увеличивают сумму всех бюджетов не более чем на 50%. В противном случае для инструкции UPDATE выполняется откат посредством инструкции ROLLBACK TRANSACTION.

Принудительное обеспечение ограничений целостности

В системах управления базами данных применяются два типа ограничений для обеспечения целостности данных: декларативные ограничения, которые определяются с помощью инструкций языка CREATE TABLE и ALTER TABLE; процедурные ограничения целостности, которые реализуются посредством триггеров.

В обычных ситуациях следует использовать декларативные ограничения для обеспечения целостности, поскольку они поддерживаются системой и не требуют реализации пользователем. Применение триггеров

рекомендуется только в тех случаях, для которых декларативные ограничения для обеспечения целостности отсутствуют.

В примере ниже показано принудительное обеспечение ссылочной целостности посредством триггеров для таблиц Employee и Works_on:

```
USE SampleDb;
GO
CREATE TRIGGER trigger_WorksonIntegrity
        ON Works_on AFTER INSERT, UPDATE
        AS IF UPDATE(EmpId)
        BEGIN
            IF (SELECT Employee.Id
                  FROM Employee, inserted
                  WHERE Employee.Id = inserted.EmpId) IS NULL
            BEGIN
              ROLLBACK TRANSACTION
              PRINT 'Строка не была вставлена/модифицирована'
             END
              ELSE
                 PRINT 'Строка была вставлена/модифицирована'
           END
```

Триггер trigger_WorksonIntegrity в этом примере проверяет ссылочную целостность для таблиц Employee и Works_on. Это означает, что проверяется каждое изменение столбца Id в ссылочной таблице Works_on, и при любом нарушении этого ограничения выполнение этой операции не допускается. (То же самое относится и к вставке в столбец Id новых значений.) Инструкция ROLLBACK TRANSACTION во втором блоке BEGIN выполняет откат инструкции INSERT или UPDATE в случае нарушения ограничения для обеспечения ссылочной целостности.

В этом примере триггер выполняет проверку на проблемы ссылочной целостности первого и второго случая между таблицами Employee и Works_on. А в примере ниже показан триггер, который выполняет проверку на проблемы ссылочной целостности третьего и четвертого случая между этими же таблицами:

```
USE SampleDb;

GO
CREATE TRIGGER trigger_RefintWorkson2
          ON Employee AFTER DELETE, UPDATE
          AS IF UPDATE (Id)
          BEGIN
            IF (SELECT COUNT(*)
                  FROM Works_on, deleted
```

```
        WHERE Works_on.EmpId = deleted.Id) > 0
    BEGIN
      ROLLBACK TRANSACTION
      PRINT 'Строка не была вставлена/модифицирована'
    END
    ELSE
      PRINT 'Строка была вставлена/модифицирована'
  END
```

Триггеры INSTEAD OF

Триггер с предложением INSTEAD OF заменяет соответствующее действие, которое запустило его. Этот триггер выполняется после создания соответствующих таблиц inserted и deleted, но перед выполнением проверки ограничений целостности или каких-либо других действий.

Триггеры INSTEAD OF можно создавать как для таблиц, так и для представлений. Когда инструкция Transact-SQL ссылается на представление, для которого определен триггер INSTEAD OF, система баз данных выполняет этот триггер вместо выполнения любых действий с любой таблицей. Данный тип триггера всегда использует информацию в таблицах inserted и deleted, созданных для представления, чтобы создать любые инструкции, требуемые для создания запрошенного события.

Значения столбцов, предоставляемые триггером INSTEAD OF, должны удовлетворять определенным требованиям:

- значения не могут задаваться для вычисляемых столбцов;
- значения не могут задаваться для столбцов с типом данных timestamp;
- значения не могут задаваться для столбцов со свойством IDENTITY, если только параметру IDENTITY_INSERT не присвоено значение ON.

Эти требования действительны только для инструкций INSERT и UPDATE, которые ссылаются на базовые таблицы. Инструкция INSERT, которая ссылается на представления с триггером INSTEAD OF, должна предоставлять значения для всех столбцов этого представления, не допускающих пустые значения NULL. (То же самое относится и к инструкции UPDATE. Инструкция UPDATE, ссылающаяся на представление с триггером INSTEAD OF, должна предоставить значения для всех столбцов представления, которое не допускает пустых значений и на которое осуществляется ссылка в предложении SET.)

В примере ниже показана разница в поведении при вставке значений в вычисляемые столбцы, используя таблицу и ее соответствующее представление:

```
USE SampleDb;

CREATE TABLE Orders (
```

```
        OrderId INT NOT NULL,
        Price MONEY NOT NULL,
        Quantity INT NOT NULL,
        OrderDate DATETIME NOT NULL,
        Total AS Price * Quantity,
        ShippedDate AS DATEADD (DAY, 7, orderdate)
    );

GO
CREATE VIEW view_AllOrders
  AS SELECT *
  FROM Orders;

GO
CREATE TRIGGER trigger_orders
  ON view_AllOrders INSTEAD OF INSERT
  AS BEGIN
    INSERT INTO Orders
    SELECT OrderId, Price, Quantity, OrderDate
    FROM inserted
END
```

В этом примере используется таблица Orders, содержащая два вычисляемых столбца. Представление view_AllOrders содержит все строки этой таблицы. Это представление используется для задания значения в его столбце, которое соотносится с вычисляемым столбцом в базовой таблице, на которой создано представление. Это позволяет использовать триггер INSTEAD OF, который в случае инструкции INSERT заменяется пакетом, который вставляет значения в базовую таблицу посредством представления view_AllOrders. (Инструкция INSERT, обращающаяся непосредственно к базовой таблице, не может задавать значение вычисляемому столбцу.)

Триггеры first и last

Компонент Database Engine позволяет создавать несколько триггеров для каждой таблицы или представления и для каждой операции (INSERT, UPDATE и DELETE) с ними. Кроме этого, можно указать порядок выполнения для нескольких триггеров, определенных для конкретной операции. С помощью системной процедуры sp_settriggerorder можно указать, что один из определенных для таблицы триггеров AFTER будет выполняться первым или последним для каждого обрабатываемого действия. Эта системная процедура имеет параметр @order, которому можно присвоить одно из трех значений:

first - указывает, что триггер является первым триггером AFTER, выполняющимся для модифицирования действия;

last - указывает, что данный триггер является последним триггером AFTER, выполняющимся для инициирования действия;

none - указывает, что для триггера отсутствует какой-либо определенный порядок выполнения. (Это значение обычно используется для того, чтобы выполнить сброс ранее установленного порядка выполнения триггера как первого или последнего.)

Изменение структуры триггера посредством инструкции ALTER TRIGGER отменяет порядок выполнения триггера (первый или последний). Применение системной процедуры sp_settriggerorder показано в примере ниже:

```
USE SampleDb;

EXEC sp_settriggerorder @triggername = 'trigger_ModifyBudget',
   @order = 'first', @stmttype='update'
```

Для таблицы разрешается определить только один первый и только один последний триггер AFTER. Остальные триггеры AFTER выполняются в неопределенном порядке. Узнать порядок выполнения триггера можно с помощью системной процедуры sp_helptrigger или функции OBJECTPROPERTY.

Возвращаемый системной процедурой sp_helptrigger результирующий набор содержит столбец order, в котором указывается порядок выполнения указанного триггера. При вызове функции objectproperty в ее втором параметре указывается значение ExeclsFirstTrigger или ExeclsLastTrigger, а в первом параметре всегда указывается идентификационный номер объекта базы данных. Если указанное во втором параметре свойство имеет значение true, функция возвращает значение 1.

Поскольку триггер INSTEAD OF исполняется перед тем, как выполняются изменения в его таблице, для триггеров этого типа нельзя указать порядок выполнения "первым" или "последним".

Триггеры DDL и области их применения

Ранее мы рассмотрели триггеры DML, которые задают действие, предпринимаемое сервером при изменении таблицы инструкциями INSERT, UPDATE или DELETE. Компонент Database Engine также позволяет определять триггеры для инструкций DDL, таких как CREATE DATABASE, DROP TABLE и ALTER TABLE. Триггеры для инструкций DDL имеют следующий синтаксис:

```
CREATE TRIGGER [schema_name.]trigger_name
   ON {ALL SERVER | DATABASE }
   [WITH {ENCRYPTION | EXECUTE AS clause_name]
   {FOR | AFTER } { event_group | event_type | LOGON}
```

AS {batch | EXTERNAL NAME method_name}

Как можно видеть по их синтаксису, триггеры DDL создаются таким же способом, как и триггеры DML. А для изменения и удаления этих триггеров используются те же инструкции ALTER TRIGGER и DROP TRIGGER, что и для триггеров DML. Поэтому рассмотрим только те параметры инструкции CREATE TRIGGER, которые новые для синтаксиса триггеров DDL.

Первым делом при определении триггера DDL нужно указать его область действия. Предложение DATABASE указывает в качестве области действия триггера DDL текущую базу данных, а предложение ALL SERVER - текущий сервер.

После указания области действия триггера DDL нужно в ответ на выполнение одной или нескольких инструкций DDL указать способ запуска триггера. В параметре event_type указывается инструкция DDL, выполнение которой запускает триггер, а в альтернативном параметре event_group указывается группа событий языка Transact-SQL. Триггер DDL запускается после выполнения любого события языка Transact-SQL, указанного в параметре event_group. Ключевое слово LOGON указывает триггер входа.

Кроме сходства триггеров DML и DDL, между ними также есть несколько различий. Основным различием между этими двумя видами триггеров является то, что для триггера DDL можно задать в качестве его области действия всю базу данных или даже весь сервер, а не всего лишь отдельный объект. Кроме этого, триггеры DDL не поддерживают триггеров INSTEAD OF. Для триггеров DDL не требуются таблицы inserted и deleted, поскольку эти триггеры не изменяют содержимого таблиц.

Триггеры DDL уровня базы данных

В примере ниже показано, как можно реализовать триггер DDL, чья область действия распространяется на текущую базу данных:

```
USE SampleDb;

GO
CREATE TRIGGER trigger_PreventDrop
   ON DATABASE FOR DROP_TRIGGER
   AS PRINT 'Перед тем, как удалить триггер, вы должны отключить
"trigger_PreventDrop"'
   ROLLBACK
```

Триггер в этом примере предотвращает удаление любого триггера для базы данных SampleDb любым пользователем. Предложение DATABASE указывает, что триггер trigger_PreventDrop является триггером уровня базы

данных. Ключевое слово DROP_TRIGGER указывает предопределенный тип события, запрещающий удаление любого триггера.

Триггеры DDL уровня сервера

Триггеры уровня сервера реагируют на серверные события. Триггер уровня сервера создается посредством использования предложения ALL SERVER в инструкции CREATE TRIGGER. В зависимости от выполняемого триггером действия, существует два разных типа триггеров уровня сервера: обычные триггеры DDL и триггеры входа. Запуск обычных триггеров DDL основан на событиях инструкций DDL, а запуск триггеров входа - на событиях входа.

Список источников

- https://metanit.com/sql/sqlserver/12.1.php
- https://zen.yandex.ru/media/id/5bbcbc1ba5bd5400a990e7d9/osnovy-pl-sql-struktura-funkcii-triggery-peremennye-zapisi-5e8c34cf82d52277064ce45b?utm_source=serp
- https://professorweb.ru/my/sql-server/2012/level3/3_18.php