**Модуль 2. Триггеры, хранимые процедуры и пользовательские функции (6 пар)**

**Цель занятия:** изучить назначение триггеров, ознакомиться с типами триггеров

**Триггеры.**

**Триггеры** являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

**Триггер** – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

**Триггеры** – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные. Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

**Триггер** представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер *только владелец базы данных*. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, *триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов*. Приведение его в действие иногда называют запуском триггера.

С помощью триггеров достигаются ***следующие цели:***

проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;

выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;

накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;

поддержка репликации.

Включите ваше воображение. У вас есть база данных и нужно прописать определенное правило, вернее, проверить набор правил. Например, есть база данным Магазин. Изначально записать и продать можно любое кол-во товара. Вы же можете осуществить проверку, чтобы кол-во взятого товара было меньше либо равно реально существующему. Это самая простая проверка, которую можно сделать при помощи Триггера. Плюс, вы можете отменить действие, которое вам не нужно.

**Общий синтаксис создания триггера:**

Create Trigger имя\_триггера

ON имя\_таблицы

FOR {AFTER | INSTEAD OF} {INSERT | UPDATE | DELETE}

[WITH ENCRYPTION]

AS Тело триггера

**Синтаксис создания триггеров в MS SQL:**

CREATE TRIGGER trigger\_name

ON {table | view}

[WITH ENCRYPTION ]

{(FOR | AFTER | INSTEAD OF) { [INSERT] [ , ] [UPDATE] }

[WITH APPEND ]

[NOT FOR REPLICATION ]

AS

{ IF UPDATE (column)

[ { AND | OR } UPDATE (column) ]

[...n]

| IF (COLUMNS\_UPDATED() {bitwise\_operator} updated\_bitmask)

{comparison\_operator} column\_bitmask [...n]

}

sql\_statement [...n]

}

Рассмотрим назначение каждого из параметров команды CREATE TRIGGER.

**trigger\_name**. Имя триггера, под которым он будет опознаваться хранимыми процедурами и командами Transact-SQL. Имя триггера должно быть уникальным в пределах базы данных. Дополнительно к имени триггера можно указать имя владельца.

**table**. Имя таблицы базы данных, с которой будет связан триггер.

**view**. Имя представления базы данных, с которым будет связан триггер.

**WITH ENCRYPTION**. При указании этого параметра сервер выполняет шифрование кода триггера, так что никто, включая администратора, не сможет получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

В SQL Server существует два параметра, определяющих поведение триггеров:

**AFTER**. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER -триггеров для каждой операции ( INSERT, UPDATE, DELETE ). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp\_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. *По умолчанию в SQL Server все триггеры являются AFTER –триггерами* (то есть, когда указывается ключевое слово FOR)*. Если единственным заданным ключевым словом является FOR, аргумент AFTER используется по умолчанию. Триггеры AFTER не могут быть определены на представлениях.*

**INSTEAD OF.** Триггер вызывается вместо выполнения команд. В отличие от AFTER -триггера INSTEAD OF -триггер может быть определен как для таблицы, так и для представления. *Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF -триггер.*

Триггеры различают по типу команд, на которые они реагируют.

**Существует три типа триггеров:**

**INSERT TRIGGER** – запускаются при попытке вставки данных с помощью команды INSERT.

**UPDATE TRIGGER** – запускаются при попытке изменения данных с помощью команды UPDATE.

**DELETE TRIGGER** – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [ DELETE] [,] [ INSERT] [,] [ UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать триггер. При его создании должна быть указана хотя бы одна команда. *Допускается создание триггера, реагирующего на две или на все три команды.*

**WITH APPEND.** Использование этого параметра требуется только в том случае, если для базы данных установлен уровень совместимости 6.5 или ниже. Так как до версии SQL Server 7.0 для таблицы было разрешено создание не более одного триггера каждого типа, то по умолчанию создание нового триггера будет приводить к удалению ранее созданного триггера. Параметр WITH APPEND позволяет создавать несколько триггеров каждого типа для базы данных с уровнем совместимости 6.5 или ниже. Для баз данных с уровнем совместимости 7.0 и выше использование этого параметра не обязательно, так как в SQL Server 7.0 и выше реализована автоматическая поддержка для одной таблицы до 16 триггеров каждого типа. Указывает, что требуется добавить триггер существующего типа. Аргумент WITH APPEND не может быть использован для триггеров INSTEAD OF или при явном указании триггера AFTER. Аргумент WITH APPEND может использоваться только при указании параметра FOR без INSTEAD OF или AFTER из соображений поддержки обратной совместимости. Аргумент WITH APPEND не может быть указан, если указан параметр EXTERNAL NAME (в случае триггера CLR).

**NOT FOR REPLICATION**. При создании триггера с этим параметром запрещается его запуск при модификации таблиц механизмами репликации. Этот параметр часто используется при создании системных триггеров поддержки подсистемы репликации.

**AS sql\_statement [. . . n].** После ключевого слова AS следует набор команд, которые будут выполнены при запуске триггера (тело триггера).

**IF UPDATE (column).** С помощью этой команды можно проверить, изменялось ли значение в определенном столбце. В случае положительного ответа будет выполнен определенный набор команд. Отметим, что эта команда может быть использована только для команд INSERT или UPDATE, но не для команды DELETE. Имя таблицы (или представления) при определении имени столбца не требуется.

**{AND | OR} UPDATE (column).** Эта конструкция применяется совместно с предыдущим параметром в случае, если необходимо проверить изменение более чем одного столбца. Аргумент column задает имя столбца, для которого необходимо отследить факт изменения данных. Ключевое слово AND предписывает запускать триггер только в том случае, если были модифицированы оба столбца (указанные в этой и предыдущей конструкциях). При использовании ключевого слова OR триггер будет выполнен при выполнении изменений в любом из столбцов.

**− [...n].** Этот параметр указывает, что допускается использование более чем одной конструкции { AND | OR} UPDATE (column).

**− IF (COLUMNS\_UPDATED()).** С помощью этой конструкции можно узнать, какие столбцы таблицы были изменены (UPDATE) или добавлены (INSERT), что допускается только для команд INSERT и UPDATE. Функция COLUMNS\_UPDATED() может быть использована в любом месте внутри триггера и возвращает двоичное число, каждый бит которого соответствует конкретному столбцу таблицы. Если тот или иной бит установлен в 1, то соответствующий столбец был изменен. Младший бит соответствует первому столбцу таблицы, второй бит справа — второму столбцу и т. д.

**bitwise\_operator.** Этот параметр задает оператор побитовой обработки, с помощью которого можно определить, изменялся конкретный столбец или нет. Например, можно использовать оператор & (битовый AND) для определения того, был ли изменен конкретный столбец. Оператор побитовой обработки выполняется для операндов COLUMNS\_UPDATED()и updated\_bitmask.

**updated\_bitmask.** Этот аргумент определяет битовую маску для проверки факта изменения в одном или нескольких столбцах. Каждый столбец представлен отдельным битом. Младший бит соответствует первому столбцу таблицы, второй бит — второму столбцу и т. д. Например, если в таблице имеется 7 столбцов, то значение updated\_bitmask для проверки факта изменения столбцов 2, 3 и 6 должно быть равно 38 (двоичное значение 0100110). Для проверки факта изменения столбцов 4, 5, 6 и 7 значение updated\_bitmask будет равно 120 (двоичное значение 1111000).

**comparison\_operator.** Оператор сравнения, используемый для проверки значения, возвращенного в результате выполнения побитовой операции сравнения, на соответствие установленным критериям. Чаще всего это оператор =, хотя допускается использование любых операторов сравнения (<, >, !=).

**column\_bitmask.** Битовая маска для проверки, были ли действительно изменены проверяемые столбцы.

Например, пусть таблица имеет следующую структуру: create table mytable (a int, b int, c int, d int, e int). Пять столбцов соответствуют 5 битам, из которых младший соответствует столбцу a, старший — столбцу e. Пусть операция, приведшая к срабатыванию триггера, изменяет столбцы a, b и e. Тогда функция Columns\_updated даст значение 10011. Пусть нас не интересует изменение столбцов b и d, но интересует изменение всех остальных столбцов (a, c и e, т. е. маска будет 10101) (напомним, что на момент написания триггера мы не знаем, какие столбцы затронет на самом деле та или иная операция изменения или вставки, т. е. какой результат даст функция columns\_updated()). Задав побитовый оператор сравнения во время выполнения, получим 10011 & 10101, что даст в результате 10001, что в десятичном представлении составляет 17. Сравнив это значение с помощью оператора сравнения и битовой маски столбца, получим ответ — удовлетворяет ли операция изменения/вставки требуемым условиям. Так, например, если бизнес-логика требует, чтобы триггер сработал при изменении все интересующих нас столбцов (a, c, e), то, естественно, битовая маска и битовая маска столбца должны иметь одинаковые значения, а оператор сравнения должен быть =. В этом случае для нашего примера, вся конструкция будет иметь вид if (columns\_updated & 17)=17. Если же требуется, чтобы изменился хотя бы один из интересующих нас столбцов, то она, очевидно, будет иметь вид if (columns\_updated & 17)>0. С помощью битовых операций можно достичь большой гибкости при составлении таких конструкций.

**При создании триггеров существует набор правил:**

1. Нельзя создавать триггеры во временных таблицах.
2. Триггеры не могут принимать параметры.
3. Триггер нельзя вызвать явно.
4. Триггеры не могут возвращать значения.
5. Для одного действия в одной таблице не может быть более одного триггера INSTEAD OF.
6. В триггерах нельзя использовать такие операторы:
   * DROP
   * CREATE
   * ALTER DATABASE
   * ALTER TABLE
   * GRANT
   * REVOKE
   * SELECT INTO

В тексте триггера нельзя использовать следующие инструкции: ALTER DATABASE, ALTER PROCEDURE, ALTER TABLE, CREATE DEFAULT, CREATE PROCEDURE, ALTER TRIGGER, ALTER VIEW, CREATE DATABASE, CREATE RULE, CREATE SCHEMA, CREATE TRIGGER, CREATE VIEW, DISK INIT, DISK RESIZE, DROP DATABASE, DROP DEFAULT, DROP PROCEDURE, DROP RULE, DROP TRIGGER, DROP VIEW, RESOTRE DATABASE, RESTORE LOG, RECONFIGURE, UPDATE STATISTICS.

***Например. При каждом запросе на обновление данных в таблице Books вы хотите видеть сообщение о том, сколько записей было изменено.***Для этого понадобится одна системная переменная, в которую автоматически записывается кол-во строк предыдущего запроса. Она называется **@@rowcount**.

Create Trigger Show\_Upd\_Amm

On Books

For Update

As

raiserror('Было изменено %d записей',0,1,@@rowcount)

Срабатывать этот триггер будет автоматически, при обновлении данных в таблице Books. Для проверки, увеличим кол-во всех книг издательства BHV на 3 экземпляра.

Update books

Set books.quantity=books.quantity+5

From press

Where books.id\_press=press.id

and press.name like '%BHV%'

Результат:

C:\Users\Гайнель\Google Drive\Мои документы\ШАГ\MS SQL\MS SQL\Уроки ПСТ\Урок 6\program\img\pict8.jpg

В SQL Server существует две таблицы - **INSERTED** и **DELETED**. Их можно и нужно использовать при написании триггеров. В эти таблицы данные попадают следующим образом: Если вы добавляете данные, то с начала они попадают в таблицу INSERTED, а только потом в основную таблицу, Если вы удаляете данные, то в начале они удаляются из основной таблицы, а потом попадают в таблицу DELETED. Структура этих таблиц абсолютно такая же как и у таблицы, над которой производится определенное действие. При **обновлении** информации производится удаление и последующая вставка данных. Т.о. с помощью данных в этих таблицах можно контролировать производимые изменения.

**Пример 2. Необходимо при занесении данных о товаре выдать ошибку, если со дня производства прошло больше месяца.**

Create Trigger Check\_Date\_trigger

On shop

for insert

as

Declare @InsDate smalldatetime

Select @InsDate=date\_in from inserted

/\*получаем дату товара, который добавляется (date\_in - поле таблицы shop, в которую производится вставка данных)\*/

if (@InsDate<=getdate()-30) /\*Проверяем, не прошло ли 30 дней\*/

Begin

raiserror('Это слишком старый товар',0,1)

raiserror('Данные о товаре сохранены не будут',0,1)

Rollback transaction

end

else

Begin

Print('Insert Ok!!!')

end

Это можно было реализовать, по сути, и с помощью Check. Без триггера не обойтись при проверке данных нескольких таблиц. Например. Нельзя удалить диск (база данных CD), если он на верхушке рейтинга продаж.

create trigger Check\_cd\_delete

on cd

for delete

as

Declare @SellAmm int,

@cd\_name varchar(25),

@best\_cd varchar(25)

Select @cd\_name=deleted.name from deleted

/\*Получаем название удаляемого диска.\*/

Declare @Svodka table (imya varchar(25), kolv int)

insert @Svodka /\*Вычитываем информацию о названиях дисков и о их популярности\*/

select cd.name,count(id\_cd) from selling,cd

where selling.id\_cd=cd.id

group by cd.name

Select @best\_cd=s.imya from @Svodka s /\*Находим название самого популярного диска по продажам\*/

where s.kolv=(Select max(kolv) from @Svodka)

if(@best\_cd=@cd\_name)/\*Проверяем совпадение названий.\*/

begin

raiserror('You can not delete this cd!!!',0,1)

rollback transaction

end

else

begin

print ('deleting query was successfull!!')

end

Также существует указать триггер вместо выполнения определенного запроса. Для этого необходимо указать INSTEAD OF.

**При удалении книги издательства 'BHV Киев' выдать ошибку.**

Create Trigger Not\_BHV

On books

Instead of Delete

As

Declare @BHV\_id int

Select @BHV\_id=id from press where press.name ='BHV Киев'

/\*

Получаем идентификатор издательства BHV Киев\*/

if (exists (select \* from deleted where id\_press=@BHV\_id))

/\*Проверяем, есть ли такой идентификатор в удаляемых книгах.\*/

raiserror ('You can not delete BHV Киев!!!',0,1)

Также, триггеры необходимы, если нужно, например, при вставке сдублировать данные в другую таблицу. Например, при добавлении какого-либо человека, скопировать данные о нем в резервную таблицу.

Create Trigger People\_copy

On People

After Insert

As

Insert into copypeople select \* from Inserted