Лабораторная работа №8:
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.

Ниже приведен синтаксис создания триггера:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

trigger_name — название триггера;

trigger_time — время срабатывания триггера:

BEFORE — перед событием;

AFTER — после события.

trigger_event — Событие:

insert — событие активируется операторами insert, data load, replace;

update — событие активируется оператором update

delete — событие активируется операторами delete, replace.

Операторы DROP TABLE и TRUNCATE не активируют выполнение триггера

tbl_name — название таблицы;

trigger_stmt - выражение, которое выполняется при активации триггера

Рассмотрим несколько примеров создания триггеров для существующей базы данных ShopDB.

Задание 1.

Триггер на добавление записи в таблицу Order. Данный триггер в случае успешного добавления данных выводит «Запись добавлена».

```
CREATE TRIGGER UPDATE_ORDER             --определение имени триггера
ON [ORDER]                  --для какой таблицы создается триггер
AFTER INSERT            --когда выполнять триггер
--INSERT – при создании записи в таблице,
--DELETE – при удалении записи в таблице,
--UPDATE – при изменении записи в таблице,
--AFTER – после выполнения операции,
--INSTEAD OF – вместо выполнения операции
AS
BEGIN       --тело триггера
    SET NOCOUNT ON;
    PRINT'Запись добавлена!'
END
GO
```

Тестирование работы триггера:

```
INSERT INTO [ORDER] -- проверка работы тригера
(IdCust)
VALUES
(3)
```

Самостоятельно создайте триггер на Обновление данных в таблице Order.

ПРИМЕЧАНИЕ. Удаление триггера осуществляется следующим образом:

```
DROP TRIGGER UPDATE_ORDER
```

Задание 2. Создайте для таблицы Orderitem триггер, демонстрирующий откат, а именно данный триггер будет срабатывать при попытке создания заказа с количеством меньше 1.

```
CREATE TRIGGER ROLLBACK_ORDER --триггер, демонстрирующий откат
ON OrdItem
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    IF (SELECT Qty FROM inserted) < 1
        BEGIN
            ROLLBACK TRAN
            PRINT 'Вы не можете создать заказ с количеством меньше 1!'
        END
RETURN
END
GO
```

Самостоятельно осуществите проверку работы триггера.

Задание 3. Создайте для таблицы Orderitem триггер, который будет проверять остаток товара в таблице Product и в случае, если количество товара в заказе больше, чем на складе, будет выдавать ошибку.

В случае успешной транзакции – уменьшит количество товара в таблице Product на количество проданного товара.

```
CREATE TRIGGER Qty_UPDATE --тригер на изменение количества товаров при их заказе
ON OrdItem
AFTER INSERT
AS
DECLARE @X INT, @Y INT
BEGIN
    SET NOCOUNT ON;
    IF NOT EXISTS(SELECT * FROM inserted
    WHERE inserted.Qty <= ALL
        (SELECT Product.InStock FROM Product
        WHERE inserted.IdProduct=Product.IdProduct))
BEGIN
ROLLBACK TRAN
PRINT 'Не допустимое кол-во товара '
END
SELECT @Y=inserted.idProduct, @X=inserted.Qty
FROM inserted
UPDATE Product
SET Product.InStock=Product.InStock-@X
WHERE Product.IdProduct=@Y
END
GO
```

<u>Самостоятельно</u> осуществите проверку работы триггера.

<u>Самостоятельно</u> создайте триггер, который будет увеличивать количество товара на складе (в таблице Product) на количество поставленного товара (таблица Поставка).

Задание 4.

Создайте триггер, который будет записывать историю добавления товаров в таблицу Товаров.

Для начала создайте дополнительную таблицу History с атрибутами (Id, IdProduct, Operation, CreateAt).

```
CREATE TABLE History -- Создаем таблицу, в которой будем хранить историю добавления продуктов
(
    Id INT IDENTITY PRIMARY KEY,
    idProduct INT NOT NULL,
    Operation NVARCHAR(200) NOT NULL,
    CreateAt DATETIME NOT NULL DEFAULT GETDATE()
);
```

Далее создаем сам триггер.

```
CREATE TRIGGER Products_INSERT   -- Создаем триггер, добавляющий в таблицу сведения о добавлении продукта
ON Product
AFTER INSERT
AS
INSERT INTO History (IdProduct, Operation)
SELECT IdProduct, 'Добавлен товар ' + [Description]
FROM INSERTED
```

Самостоятельно осуществите проверку работы триггера.

Самостоятельно создайте триггер, который будет записывать историю удаления и обновления товаров в таблицу Товаров.

Задание 5.

Триггер INSTEAD OF срабатывает вместо операции с данными. Он определяется в принципе также, как триггер AFTER, за тем исключением, что он может определяться только для одной операции - INSERT, DELETE или UPDATE. И также он может применяться как для таблиц, так и для представлений.

Для реализации работы триггера создайте атрибут в таблице Product под названием IsDeleted с типом данных Bit. Данный столбец будет указывать, удалена ли запись. То есть вместо жесткого удаления полностью из базы данных мы хотим выполнить мягкое удаление, при котором запись остается в базе данных.

Определим триггер для удаления записи:

```
CREATE TRIGGER products_delete
ON Product
INSTEAD OF DELETE
AS
UPDATE Product
SET IsDeleted = 1
WHERE IdProduct =(SELECT IdProduct FROM deleted)

DELETE FROM Product
WHERE [Description]='iPhone X';

SELECT * FROM Product
```

Самостоятельно осуществите проверку работы триггера.

Самостоятельно создайте триггер, который позволит реализовать принцип «мягкого удаления» сведений о Заказчиках (таблица Customer).