**Базы данных**

**Отчет по лабораторной работе №14, Лешук Дмитрий ПОИБМС 7-2**

**Цель:** Изучить особенности применения разработки и использования триггеров.

**Триггер** – это особый вид хранимой процедуры, предназначенной для обработки событий в БД. Поддерживается два типа триггеров: *DDL-триггеры* и *DML-триггеры*. Для каждого типа определено свое семейство событий, обработку которых триггер этого типа может выполнять.

**DML-**триггеры бывают двух типов: AFTER-триггеры и INSTEAD OF-триггеры. Триггеры типа AFTER исполняются *после* выполнения оператора, вызвавшего соответствующее событие. При этом создаются автоматически две псевдотаблицы INSERTED и DELETED.

В первом задании необходимо было создать таблицу, которую будут заполнять триггеры, включающая в себя 4 столбца – рисунок 1.1.

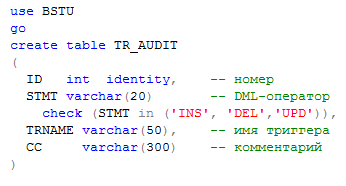
****

Рисунок 1.1 – Таблица TR\_AUDIT

Далее разработаем AFTER-триггер с именем TR\_TEACHER\_INSдля таблицы TEACHER, реагирующий на событие INSERT – рисунок 1.2. Триггер достаёт из псевотаблицы вставленные значения и заносит их в таблицу TR\_AUDIT.

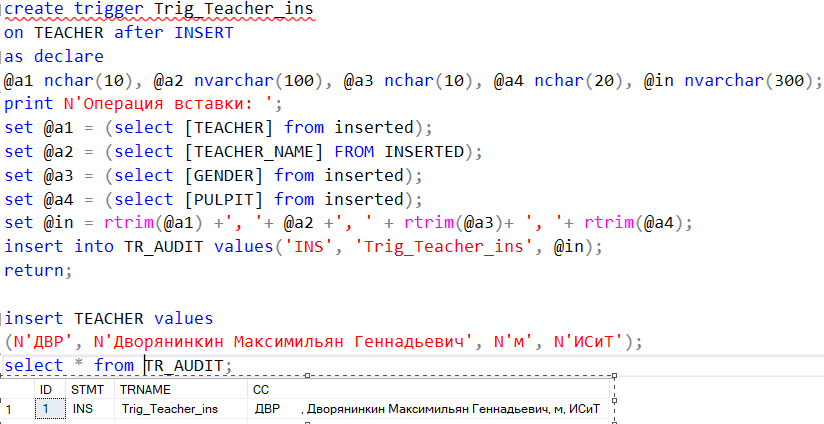


Рисунок 1.2 – AFTER-триггер Trig\_Teacher\_Ins

Второе и третье задания – рисунки 1.3, 1.4. Схожим образом создаём AFTER-триггеры, реагирующие на события DELETE - TR\_TEACHER\_DEL, и UPDATE - TR\_TEACHER\_UPD. TR\_TEACHER\_DEL работает по тому же принципу, что и Trig\_Teacher\_Ins, только данные берутся из псевдотаблицы deleted. UPDATE-триггер работает немного по-другому: мы объявляем, что триггер будет работать «AFTER UPDATE», однако те данные, которые заменили (старые) попадут в псевдотаблицу deleted, а новые данные – в псевдотаблицу inserted.

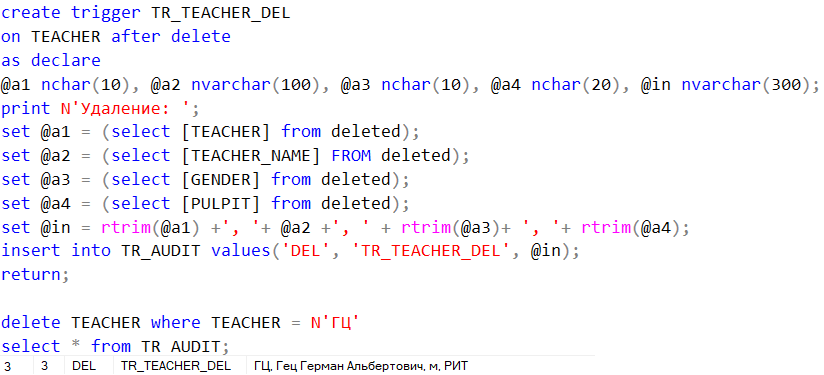


Рисунок 1.3 – AFTER-триггер, реагирующий на удаление

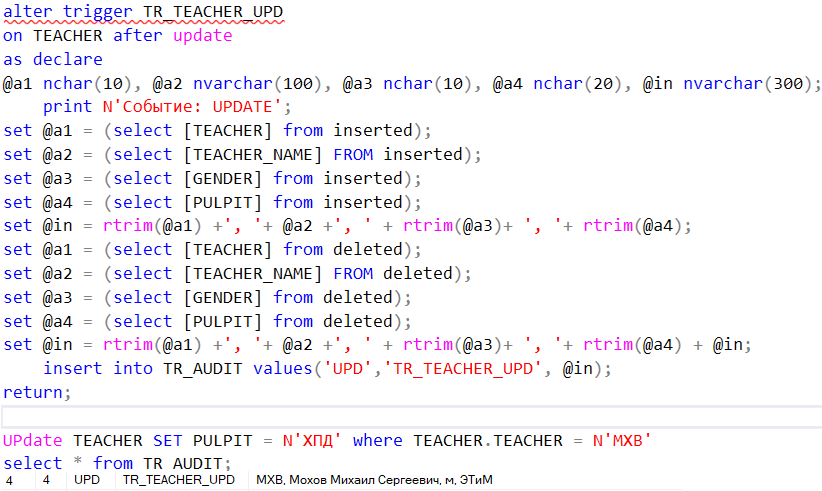


Рисунок 1.4 – UPDATE-триггер

В следующем задании требовалось создать after-триггер, который реагирует на все три события: INSERT, DELETE, UPDATE. И в зависимости от того, какое событие произошло, возвращает строку и делает соответствующую запись в таблицу TR\_AUDIT – листинг 1.1

alter trigger TR\_TEACHER on TEACHER after INSERT, DELETE, UPDATE

as

declare @a11 nchar(10), @a21 nvarchar(100), @a31 nchar(10), @a41 nchar(20), @in1 nvarchar(300);

declare @ins int = (select count(\*) from inserted),

@del int = (select count(\*) from deleted);

if @ins > 0 and @del = 0

begin

print N'Событие: INSERT';

set @a11 = (select [TEACHER] from inserted);

set @a21 = (select [TEACHER\_NAME] FROM INSERTED);

set @a31 = (select [GENDER] from inserted);

set @a41 = (select [PULPIT] from inserted);

set @in1 = rtrim(@a11) +', '+ @a21 +', ' + rtrim(@a31)+ ', '+ rtrim(@a41);

insert into TR\_AUDIT values('INS', 'TR\_TEACHER', @in1);

end;

else

if @ins = 0 and @del > 0

begin

print N'Событие: DELETE';

set @a11 = (select [TEACHER] from deleted);

set @a21 = (select [TEACHER\_NAME] FROM deleted);

set @a31 = (select [GENDER] from deleted);

set @a41 = (select [PULPIT] from deleted);

set @in1= rtrim(@a11) +', '+ @a21 +', ' + rtrim(@a31)+ ', '+ rtrim(@a41);

insert into TR\_AUDIT values('DEL', 'TR\_TEACHER', @in1);

end;

else

if @ins > 0 and @del > 0

begin

print N'Событие: UPDATE';

set @a11 = (select [TEACHER] from inserted);

set @a21 = (select [TEACHER\_NAME] FROM inserted);

set @a31 = (select [GENDER] from inserted);

set @a41 = (select [PULPIT] from inserted);

set @in1 = rtrim(@a11) +', '+ @a21 +', ' + rtrim(@a31)+ ', '+ rtrim(@a41);

set @a11 = (select [TEACHER] from deleted);

set @a21 = (select [TEACHER\_NAME] FROM deleted);

set @a31 = (select [GENDER] from deleted);

set @a41 = (select [PULPIT] from deleted);

set @in1 = rtrim(@a11) +', '+ @a21 +', ' + rtrim(@a31)+ ', '+ rtrim(@a41) + @in1;

insert into TR\_AUDIT values('UPD','TR\_TEACHER', @in1);

end;

return;

Листинг 1.1 – уникальный триггер

Следующее задание даёт нам понять, что проверка ограничения целостности выполняется до срабатывания AFTER-триггера. Если добавить ограничение not null для столбца GENDER в таблицe TEACHER, а потом попытаться установить значение NULL в этот столбец, триггер не сработает и запись «UPDATE» не сработает.

Запрос, представленный на рисунке 1.5 демонстрирует способ узнать, какие триггеры активны для определенной таблицы определенного типа.

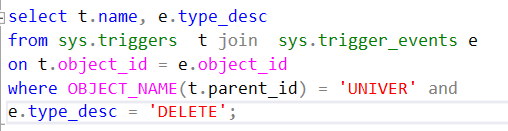


Рисунок 1.5 – Вывод триггеров

Следующее задание расширяет возможности триггеров: создаём 3 одинаковых триггера, но называем их по-разному. Если произойдёт связанное с ними событие, то триггеры выполнятся в том порядке, в котором они создавались. Однако, мы можем управлять порядком вызова триггеров с помощью встроенной процедуры SP\_SETTRIGGERORDERS, которая выглядит следующим образом – рисунок 1.6.

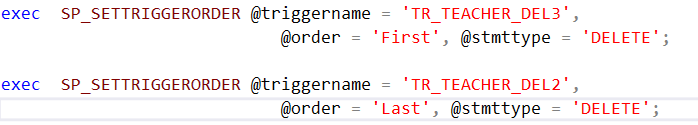


Рисунок 1.6 – Изменение порядка выполнения триггеров

Седьмое задание показывает нам, что AFTER-триггер является частью транзакции, в рамках которого выполняется оператор, активизировавший триггер – рисунок 1.7.

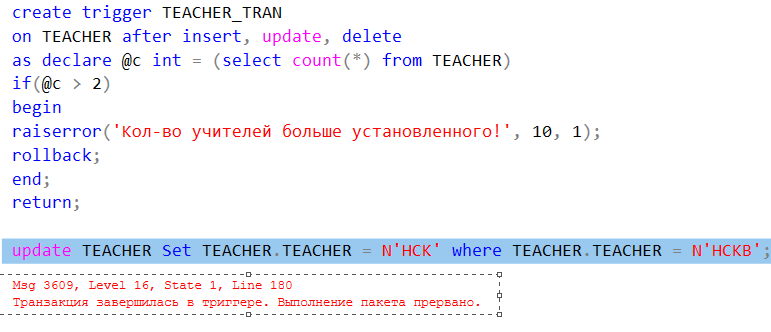


Рисунок 1.7 – Триггер TEACHER\_TRAN

Триггер типа INSTEAD OF выполняется *вместо* оператора, вызвавшего соответствующее событие. Выполнение INSTEAD OF триггера предшествует проверке установленных для таблицы ограничений целостности. Создадим триггер для таблицы FACULTY, который будет запрещать удаление – рисунок 1.8.

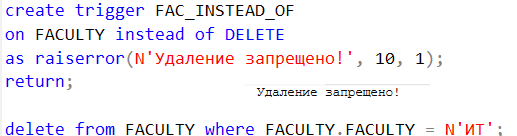


Рисунок 1.8 – INSTEAD OF триггер

Последнее задание заключает в себе создании DDL-триггера, реагирующий на все DDL-события в БД UNIVER. Триггер должен запрещать создавать новые таблицы и удалять существующие – рисунок 1.8.

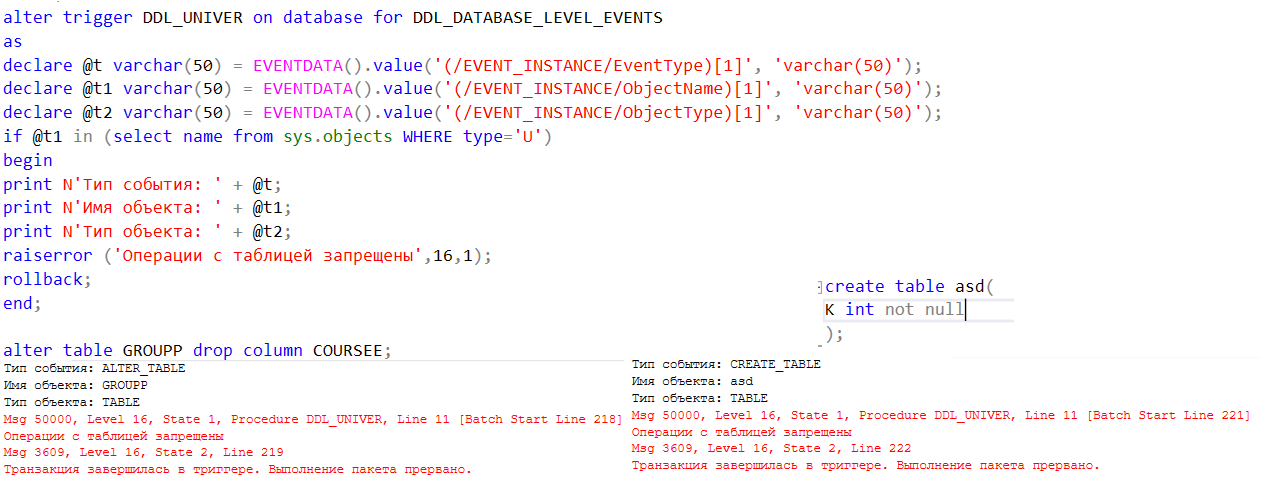


Рисунок 1.8 – Триггер базы данных

**Вывод:**

В ходе данной лабораторной работы были получены теоретические навыки о DML-триггерах, созданы AFTER-INSERT, AFTER-UPDATE, AFTER-DELETE триггеры для таблицы TEACHER, а также уникальный триггер, работающий на всю базу данных UNIVER, запрещающий создание и удаление таблиц.