**Лабораторная работа № 14. Применение триггеров**

**КУРНОСЕНКО СОФЬЯ**

**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.

Триггер типа INSTEAD OF выполняется *вместо* оператора, вызвавшего соответствующее событие. Выполнение INSTEAD OF триггера предшествует проверке установленных для таблицы ограничений целостности.

1.С помощью сценария, представленного на рисунке, создать таблицу **TR\_AUDIT**.

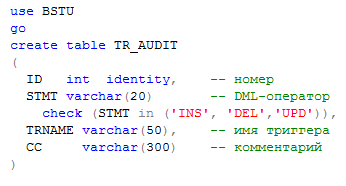
****

Таблица предназначена для добавления в нее строк триггерами.

В столбец **STMT** триггер должен поместить событие, на которое он среагировал, а в столбец **TRNAME −** собственное имя.

Разработать AFTER-триггер с именем **TR\_TEACHER\_INS** для таблицы **TEACHER**, реагирующий на событие **INSERT**. Триггер должен записывать строки вводимых данных в таблицу **TR\_AUDIT**. В столбец **СС** помещаются значения столбцов вводимой строки.

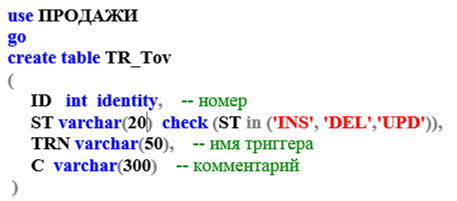
***→ Теория (из методички)***

Событие INSERT при выполнении AFTER-триггера приводит к тому, что в псевдотаблицу INSERTED помещаются строки, добавленные оператором INSERT, вызвавшим это событие. Псевдотаблица DELETED остается пустой.

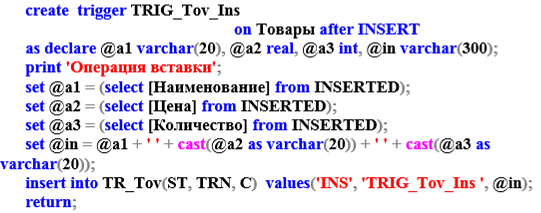
При возникновении события DELETE в таблицу DELETED копируются удаленные строки, а таблица INSERTED остается пустой.

При изменении строк таблицы с помощью оператора UPDATE заполняются обе псевдотаблицы, при этом таблица INSERTED содержит обновленные версии строк, а таблица DELETED − версию строк до их изменения.

Пусть вспомогательная таблица для отражения информации об операциях с таблицей **Товары** для базы данных ПРОДАЖИ имеет имя **TR\_Tov**:



AFTER-триггер с именем **TRIG\_Tov\_Ins** для таблицы **Товары** реагирует на событие **INSERT**. В столбец **ST** триггер помещает событие, на которое он среагировал, в столбец **TRN −** собственное имя, в столбец **С −** значения столбцов вводимой строки, которые берутся из псевдотаблицы INSERTED.



Триггер реагирует на событие вставки информации:

**insert into Товары(Наименование, Цена, Количество)**

**values('Планшет', 140, 20);**

Содержимое таблицы **TR\_Tov** отображается с помощью запроса:

**select \* from TR\_Tov**

***→ Выполнение заданий***

use UNIVERSITY;

-- вспомогательная таблица для отображения инф-ции об операциях

create table TR\_AUDIT

(

ID int identity, -- номер

STMT varchar(20) -- DML-оператор

check (STMT in ('INS', 'DEL', 'UPD')),

TRNAME varchar(50), -- имя триггера

CC varchar(300), -- комментарий

)

go

create trigger TR\_TEACHER\_INS -- create trigger название\_триггера

on TEACHER -- триггер будет регировать на события в таблице TEACHER

after insert -- триггер срабатывает после (after) события insert

as

begin

declare @teacher char(10) = (select TEACHER from INSERTED),

@teacher\_name varchar(100) = (select TEACHER\_NAME from INSERTED),

@gender char(1) = (select GENDER from INSERTED),

@pulpit char(20) = (select PULPIT from INSERTED);

declare @allInsertedInfo varchar(200) = @teacher + ' ' +

@teacher\_name + ' ' + @gender + ' ' +

@pulpit;

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

end;

-- вставим строку в таблицу TEACHER

go

insert into TEACHER values('КРНСНК', 'Курносенко Софья Андреевна', 'ж', 'ИСиТ');

-- выведем значения из таблицы TR\_AUDIT

select \* from TR\_AUDIT;

******

2.Создать AFTER-триггер с именем **TR\_TEACHER\_DEL** для таблицы **TEACHER**, реагирующий на событие **DELETE**. Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой удаляемой строки. В столбец **СС** помещаются значения столбца **TEACHER** удаляемой строки.

***→ Выполнение заданий***

go

create trigger TR\_TEACHER\_DEL

on TEACHER

after DELETE

as

begin

declare @teacher char(10) = (select TEACHER from DELETED),

@teacher\_name varchar(100) = (select TEACHER\_NAME from DELETED),

@gender char(1) = (select GENDER from DELETED),

@pulpit char(20) = (select PULPIT from DELETED);

declare @allDeletedInfo varchar(200) = @teacher + ' ' +

@teacher\_name + ' ' + @gender + ' ' +

@pulpit;

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

end;

-- удалим строку из таблицы TEACHER

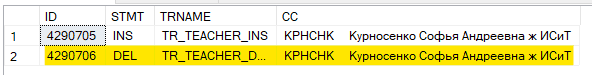
go

delete from TEACHER

where TEACHER = 'КРНСНК';

-- выведем значения из таблицы TR\_AUDIT

select \* from TR\_AUDIT;

******

3.Создать AFTER-триггер с именем **TR\_TEACHER\_UPD** для таблицы **TEACHER**, реагирующий на событие **UPDATE**. Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой изменяемой строки. В столбец **СС** помещаются значения столбцов изменяемой строки до и после изменения.

***→ Выполнение заданий***

go

create trigger TR\_TEACHER\_UPD

on TEACHER

after UPDATE

as

begin

declare

@oldTeacher char(10) = (select TEACHER from DELETED where TEACHER is not null),

@oldTeacher\_name varchar(100) = (select TEACHER\_NAME from DELETED where TEACHER\_NAME is not null),

@oldGender char(1) = (select GENDER from DELETED where GENDER is not null),

@oldPulpit char(20) = (select PULPIT from DELETED where PULPIT is not null),

@newTeacher char(10) = (select TEACHER from INSERTED where TEACHER is not null),

@newTeacher\_name varchar(100) = (select TEACHER\_NAME from INSERTED where TEACHER\_NAME is not null),

@newGender char(1) = (select GENDER from INSERTED where GENDER is not null),

@newPulpit char(20) = (select PULPIT from INSERTED where PULPIT is not null);

declare @oldInfo varchar(200) = @oldTeacher + ' ' +

@oldTeacher\_name + ' ' + @oldGender + ' ' +

@oldPulpit;

declare @newInfo varchar(200) = @newTeacher + ' ' +

@newTeacher\_name + ' ' + @newGender + ' ' +

@newPulpit;

declare @result varchar(400) = 'deleted:' + char(10) + @oldInfo +

'inserted:' + char(10) + @newInfo;

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

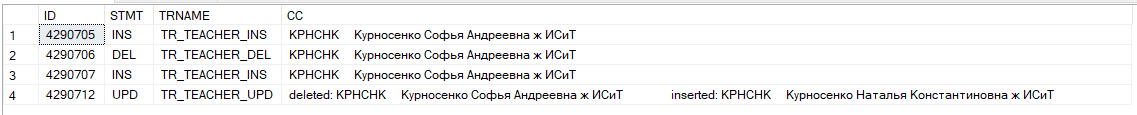
end;

insert into TEACHER values('КРНСНК', 'Курносенко Софья Андреевна', 'ж', 'ИСиТ');

update TEACHER set TEACHER\_NAME = 'Курносенко Наталья Константиновна'

where TEACHER\_NAME = 'Курносенко Софья Андреевна';

select \* from TR\_AUDIT;

******

4.Создать AFTER-триггер с именем **TR\_TEACHER** для таблицы **TEACHER**, реагирующий на события **INSERT**, **DELETE**, **UPDATE**.

Триггер должен записывать строку данных в таблицу **TR\_AUDIT** для каждой изменяемой строки. В коде триггера определить событие, активизировавшее триггер и поместить в столбец **СС** соответствующую событию информацию.

Разработать сценарий, демонстрирующий работоспособность триггера.

***→ Теория (из методички)***

Пример триггера **TRIG\_Tov**,которыйзаписывает данные в таблицу **TR\_Tov** для каждой изменяемой строки таблицы **Товары**:

**create trigger TRIG\_Tov on Товары after INSERT, DELETE, UPDATE**

**as declare @a1 varchar(20), @a2 real, @a3 int, @in varchar(300);**

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

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

**if @ins > 0 and @del = 0**

**begin**

**print 'Событие: INSERT';**

**set @a1 = (select [Наименование] from INSERTED);**

**set @a2 = (select [Цена] from INSERTED);**

**set @a3 = (select [Количество] from INSERTED);**

**set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20));**

**insert into TR\_Tov(ST, TRN, C) values('INS', 'TRIG\_Tov', @in);**

**end;**

**else**

**if @ins = 0 and @del > 0**

**begin**

**print 'Событие: DELETE';**

**set @a1 = (select [Наименование] from deleted);**

**set @a2 = (select [Цена] from deleted);**

**set @a3 = (select [Количество] from deleted);**

**set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20));**

**insert into TR\_Tov(ST, TRN, C) values('DEL', 'TRIG\_Tov', @in);**

**end;**

**else**

**if @ins > 0 and @del > 0**

**begin**

**print 'Событие: UPDATE';**

**set @a1 = (select [Наименование] from inserted);**

**set @a2 = (select [Цена] from inserted);**

**set @a3 = (select [Количество] from inserted);**

**set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20));**

**set @a1 = (select [Наименование] from deleted);**

**set @a2 = (select [Цена] from deleted);**

**set @a3 = (select [Количество] from deleted);**

**set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20))+' '+@in;**

**insert into TR\_Tov(ST, TRN, C) values('UPD', 'TRIG\_Tov', @in);**

**end;**

**return;**

DML-операторы изменения таблицы **Товары**:

**insert into Товары(Наименование, Цена, Количество)**

**values('Стол', 140, 20);**

**delete from Товары where Наименование = 'Стол';**

**update Товары set Количество = 20 where Наименование = 'Стул';**

Оператор **select \* from TR\_Tov** позволяет проверить содержимое **TR\_Tov**.

***→ Выполнение заданий***

go

create trigger TR\_TEACHER

on TEACHER

after insert, update, delete

as

begin

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

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

-- вставка

if @ins>0 and @del=0

begin

declare @teacherI char(10) = (select TEACHER from INSERTED),

@teacher\_nameI varchar(100) = (select TEACHER\_NAME from INSERTED),

@genderI char(1) = (select GENDER from INSERTED),

@pulpitI char(20) = (select PULPIT from INSERTED);

declare @allInsertedInfo varchar(200) = @teacherI + ' ' +

@teacher\_nameI + ' ' + @genderI + ' ' +

@pulpitI;

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

end;

-- удаление

else if @del>0 and @ins=0

begin

declare @teacherD char(10) = (select TEACHER from DELETED),

@teacher\_nameD varchar(100) = (select TEACHER\_NAME from DELETED),

@genderD char(1) = (select GENDER from DELETED),

@pulpitD char(20) = (select PULPIT from DELETED);

declare @allDeletedInfo varchar(200) = @teacherD + ' ' +

@teacher\_nameD + ' ' + @genderD + ' ' +

@pulpitD;

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

end;

-- обновление

else if @del>0 and @ins>0

begin

declare

@oldTeacher char(10) = (select TEACHER from DELETED where TEACHER is not null),

@oldTeacher\_name varchar(100) = (select TEACHER\_NAME from DELETED where TEACHER\_NAME is not null),

@oldGender char(1) = (select GENDER from DELETED where GENDER is not null),

@oldPulpit char(20) = (select PULPIT from DELETED where PULPIT is not null),

@newTeacher char(10) = (select TEACHER from INSERTED where TEACHER is not null),

@newTeacher\_name varchar(100) = (select TEACHER\_NAME from INSERTED where TEACHER\_NAME is not null),

@newGender char(1) = (select GENDER from INSERTED where GENDER is not null),

@newPulpit char(20) = (select PULPIT from INSERTED where PULPIT is not null);

declare @oldInfo varchar(200) = @oldTeacher + ' ' +

@oldTeacher\_name + ' ' + @oldGender + ' ' +

@oldPulpit;

declare @newInfo varchar(200) = @newTeacher + ' ' +

@newTeacher\_name + ' ' + @newGender + ' ' +

@newPulpit;

declare @result varchar(400) = 'deleted:' + char(10) + @oldInfo +

'inserted:' + char(10) + @newInfo;

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

end;

end;

-- операции вставки, обновления и удаления

insert into TEACHER values('СКТ', 'Гинтоки Саката', 'м', 'ИСиТ');

update TEACHER set PULPIT = 'ОХ'

where TEACHER = 'СКТ';

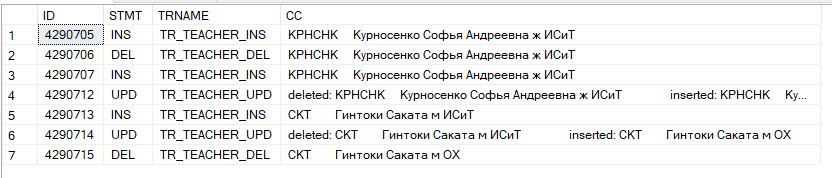
delete from TEACHER

where TEACHER = 'СКТ';

select \* from TR\_AUDIT;

-- удаляем триггер

drop trigger TR\_TEACHER;



5.Разработать сценарий, который демонстрирует на примере базы данных X\_UNIVER, что проверка ограничения целостности выполняется до срабатывания AFTER-триггера.

***→ Теория (из методички)***

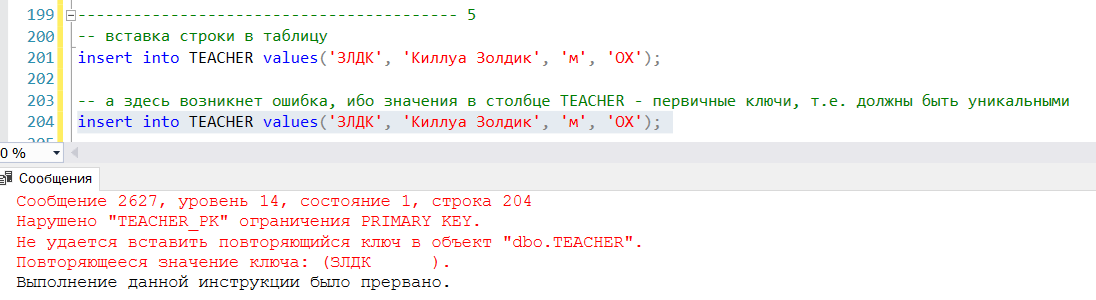
Важной особенностью AFTER-триггера является то, что он вызывается после выполнения активизирующего его оператора. Поэтому, если оператор нарушает ограничение целостности, то возникшая ошибка не допускает выполнения этого оператора и соответствующих триггеров.

**alter table Товары add constraint Цена check(Цена >= 15)**

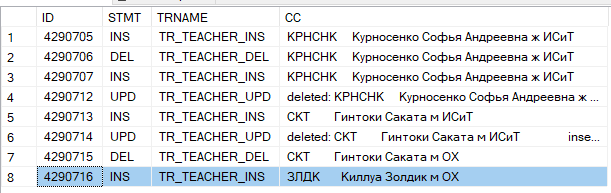
**go**

**update Товары set Цена = 10 where Наименование = 'Стул';**

***→ Выполнение заданий***



В таблице триггеров добавлена только одна запись, так как событие не сработало, ибо инструкция прервалась из-за исключения:



6.Создать для таблицы **TEACHER** три AFTER-триггера с именами: **TR\_TEACHER\_ DEL1**, **TR\_TEACHER\_DEL2** и **TR\_TEA-CHER\_ DEL3**. Триггеры должны реагировать на событие DELETE и формировать соответствующие строки в таблицу **TR\_AUDIT**.

Получить список триггеров таблицы **TEACHER**.

Упорядочить выполнение триггеров для таблицы **TEACHER**, реагирующих на событие **DELETE** следующим образом: первым должен выполняться триггер с именем **TR\_TEACHER\_DEL3**, последним – триггер **TR\_TEACHER\_DEL2**.

Примечание: использовать системные представления **SYS.TRIGGERS** и **SYS.TRIGGERS\_EVENTS**, а также системную процедуру **SP\_SETTRIGGERORDERS**.

***→ Теория (из методички)***

Пусть для таблицы **Товары** имеется три триггера, реагирующих на событие UPDATE:

**create trigger AUD\_AFTER\_UPDA on Товары after UPDATE**

**as print 'AUD\_AFTER\_UPDATE\_A';**

**return;**

**go**

**create trigger AUD\_AFTER\_UPDB on Товары after UPDATE**

**as print 'AUD\_AFTER\_UPDATE\_B';**

**return;**

**go**

**create trigger AUD\_AFTER\_UPDC on Товары after UPDATE**

**as print 'AUD\_AFTER\_UPDATE\_C';**

**return;**

**go**

Проверить порядок выполнения триггеров можно следующим запросом:

**select t.name, e.type\_desc**

**from sys.triggers t join sys.trigger\_events e**

**on t.object\_id = e.object\_id**

**where OBJECT\_NAME(t.parent\_id) = 'Товары' and**

**e.type\_desc = 'UPDATE' ;**

Изменение порядка выполнения триггеров выполняется с помощью системных процедур:

**exec SP\_SETTRIGGERORDER @triggername = 'AUD\_AFTER\_UPDC',**

**@order = 'First', @stmttype = 'UPDATE';**

**exec SP\_SETTRIGGERORDER @triggername = 'AUD\_AFTER\_UPDA',**

**@order = 'Last', @stmttype = 'UPDATE';**

***→ Выполнение заданий***

Создаем триггеры:

-- первый триггер

go

create trigger TR\_TEACHER\_DEL1

on TEACHER

after DELETE

as

begin

declare @teacher char(10) = (select TEACHER from DELETED),

@teacher\_name varchar(100) = (select TEACHER\_NAME from DELETED),

@gender char(1) = (select GENDER from DELETED),

@pulpit char(20) = (select PULPIT from DELETED);

declare @allDeletedInfo varchar(200) = @teacher + ' ' +

@teacher\_name + ' ' + @gender + ' ' +

@pulpit;

insert into TR\_AUDIT values('DEL', 'TR\_TEACHER\_DEL1', @allDeletedInfo);

end;

-- второй триггер

go

create trigger TR\_TEACHER\_DEL2

on TEACHER

after DELETE

as

begin

declare @teacher char(10) = (select TEACHER from DELETED),

@teacher\_name varchar(100) = (select TEACHER\_NAME from DELETED),

@gender char(1) = (select GENDER from DELETED),

@pulpit char(20) = (select PULPIT from DELETED);

declare @allDeletedInfo varchar(200) = @teacher + ' ' +

@teacher\_name + ' ' + @gender + ' ' +

@pulpit;

insert into TR\_AUDIT values('DEL', 'TR\_TEACHER\_DEL2', @allDeletedInfo);

end;

-- третий триггер

go

create trigger TR\_TEACHER\_DEL3

on TEACHER

after DELETE

as

begin

declare @teacher char(10) = (select TEACHER from DELETED),

@teacher\_name varchar(100) = (select TEACHER\_NAME from DELETED),

@gender char(1) = (select GENDER from DELETED),

@pulpit char(20) = (select PULPIT from DELETED);

declare @allDeletedInfo varchar(200) = @teacher + ' ' +

@teacher\_name + ' ' + @gender + ' ' +

@pulpit;

insert into TR\_AUDIT values('DEL', 'TR\_TEACHER\_DEL3', @allDeletedInfo);

end;

Получаем список триггеров таблицы TEACHER:

-- получение имен триггеров таблицы TEACHER

select t.name,

e.type\_desc

from

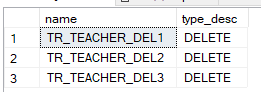
sys.triggers as t

join

sys.trigger\_events as e

on t.object\_id = e.object\_id

where OBJECT\_NAME(t.parent\_id) = 'TEACHER'



Упорядочиваем выполнение триггеров:

-- упорядочим выполнение триггеров: первым TR\_TEACHER\_DEL3, последним - TR\_TEACHER\_DEL2

exec sp\_settriggerorder @triggername = 'TR\_TEACHER\_DEL3',

@order = 'first',

@stmttype = 'delete';

exec sp\_settriggerorder @triggername = 'TR\_TEACHER\_DEL2',

@order = 'last',

@stmttype = 'delete';

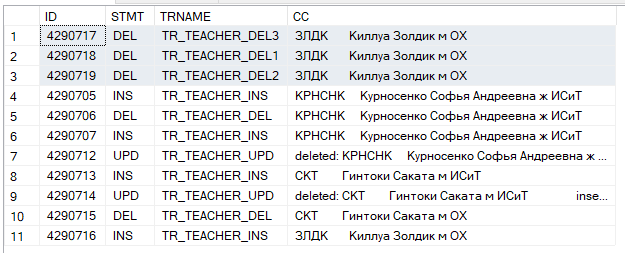
Проверим порядок выполнения триггеров:

-- удалим запись из таблицы TEACHER и посмотрим, в каком порядке отработали триггеры

delete from TEACHER

where TEACHER = 'ЗЛДК';

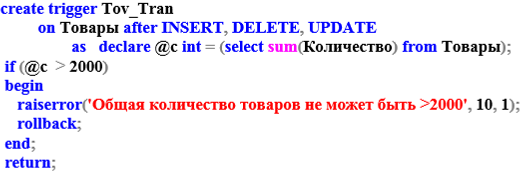
select \* from TR\_AUDIT;



7.Разработать сценарий, демонстрирующий на примере базы данных X\_UNIVER утверждение: AFTER-триггер является частью транзакции, в рамках которого выполняется оператор, активизировавший триггер.

***→ Теория (из методички)***

Пусть разработан триггер, который ограничивает общее количество товаров в таблице **Товары** некоторой величиной:



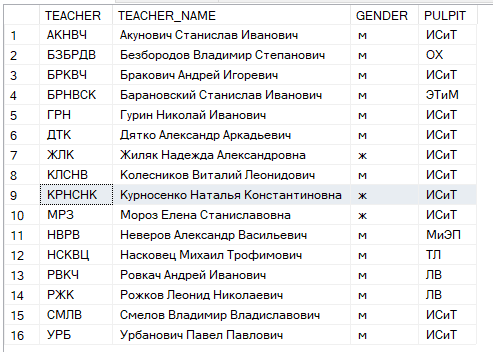
Если попробовать обновить информацию с помощью оператора UPDATE, то это вызовет сообщение об ошибке и транзакция завершится аварийно:

**update Товары set Количество = 1990**

**where Наименование = 'Стол'**

***→ Выполнение заданий***

Таблица TEACHER до каких-либо действий:



Создаем триггер, запрещающий удаление записей из таблицы TEACHER:

go

create trigger noDeletingInTeacherTableTrig

on TEACHER

after delete

as

begin

raiserror('Триггер не допускает удаление записей в таблице TEACHER.', 10, 1);

rollback;

end;

delete from TEACHER

where TEACHER = 'КРНСНК';

Пробуем удалить запись из таблицы TEACHER:

delete from TEACHER

where TEACHER = 'КРНСНК';

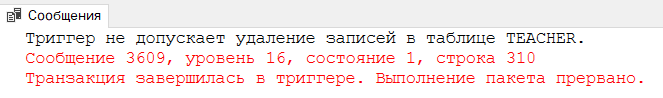
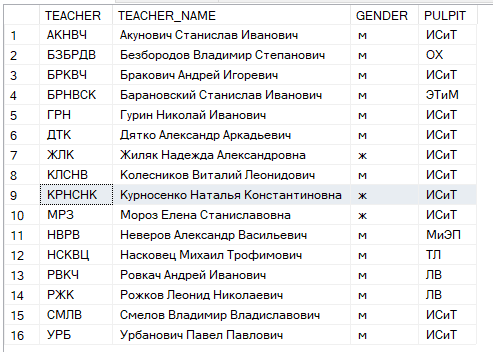


Таблица теперь:



Как видим запись, которую мы пытались удалить, осталась, потому что если произойдет *откат**триггера*, то будут *отклонены и пользовательские изменения*.

8.Для таблицы **FACULTY** создать **INSTEAD OF**-триггер, запрещающий удаление строк в таблице.

Разработать сценарий, который демонстрирует на примере базы данных X\_UNIVER, что проверка ограничения целостности выполнена, если есть INSTEAD OF-триггер.

С помощью оператора DROP удалить все DML-триггеры, созданные в этой лабораторной работе.

***→ Теория (из методички)***

|  |
| --- |
| INSTEAD OF-триггер, запрещающий удалять строки из таблицы **Товары**:    Операция удаления строки не будет выполнена:  **delete from Товары where Наименование = 'Стол';** |

***→ Выполнение заданий***

go

create trigger noDeletingInFacultyTableTrig

on FACULTY

instead of delete

as

raiserror('Триггер не допускает удаление записей в таблице FACULTY.', 10, 1);

delete from FACULTY

where FACULTY = 'ИЭФ';



9.Создать DDL-триггер, реагирующий на все DDL-события в БД **UNIVER**. Триггер должен запрещать создавать новые таблицы и удалять существующие. Свое выполнение триггер должен сопровождать сообщением, которое содержит: тип события, имя и тип объекта, а также пояснительный текст, в случае запрещения выполнения оператора.

Разработать сценарий, демонстрирующий работу триггера.

***→ Теория (из методички)***

|  |
| --- |
| Пусть разработан триггер **DDL\_PRODAJI**, который запрещает вносить изменения в базу данных ПРОДАЖИ:  **use ПРОДАЖИ**  **go**  **create trigger DDL\_PRODAJI on database**  **for DDL\_DATABASE\_LEVEL\_EVENTS as**  **declare @t varchar(50) = EVENTDATA().value('(/EVENT\_INS- TANCE/EventType)[1]', varchar(50)');**  **declare @t1 varchar(50) = EVENTDATA().value('(/EVENT\_INS-TANCE/ObjectName)[1]', 'varchar(50)');**  **declare @t2 varchar(50) = EVENTDATA().value('(/EVENT\_INS-TANCE/ObjectType)[1]', 'varchar(50)');**  **if @t1 = 'Товары'**  **begin**  **print 'Тип события: '+@t;**  **print 'Имя объекта: '+@t1;**  **print 'Тип объекта: '+@t2;**  **raiserror( N'операции с таблицей Товары запрещены', 16, 1);**  **rollback;**  **end;**  Если попытаться внести изменения в таблицу **Товары**, например:  **alter table Товары Drop Column Количество;**  то будет выведена соответствующая информация. |

***→ Выполнение заданий***

go

create trigger ddlTrigger

on database

for DDL\_DATABASE\_LEVEL\_EVENTS

as

begin

declare @eventType varchar(50) = EVENTDATA().value('(/EVENT\_INSTANCE/

EventType)[1]', 'varchar(50)');

declare @objName varchar(50) = EVENTDATA().value('(/EVENT\_INSTANCE/

ObjectName)[1]', 'varchar(50)');

declare @objType varchar(50) = EVENTDATA().value('(/EVENT\_INSTANCE/

ObjectType)[1]', 'varchar(50)');

print 'Тип события: ' + @eventType;

print 'Имя объекта: ' + @objName;

print 'Тип объекта: ' + @objType;

-- если событие является собятием создания или удаления таблицы

if(@eventType in ('CREATE\_TABLE', 'DROP\_TABLE'))

begin

raiserror('Операции создания и удаления таблиц в БД UNIVERSITY запрещены', 16, 1);

rollback;

end;

end;

create table bitch(id int identity(1,1));

drop table TR\_AUDIT;

drop trigger ddlTrigger on database

