МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологий программирования

Дисциплина: Базы данных

Отчёт по лабораторной работе

Выполнил:

Проверил:

Полоцк, 20__

Вариант задания №23 – Летопись острова Санта-Белинда

Где-то в великом океане находится воображаемый остров Санта-Белинда. Вот уже триста лет ведется подробная летопись острова. В эту летопись заносятся и данные обо всех людях, какое-то время проживавших на острове. Записываются их имена, пол, даты рождения и смерти. Хранятся там и имена их родителей, если известно, кто они. У некоторых отсутствуют сведения об отце, у некоторых — о матери, а часть людей, судя по записям, — круглые сироты. Из летописи можно узнать, когда был построен каждый дом, стоящий на острове (а если сейчас его уже нет, то когда он был снесен), точный адрес и подробный план этого дома, кто и когда в нем жил.

Точно так же, как и столетия назад, на острове действуют предприниматели, занимающиеся, в частности, ловлей рыбы, заготовкой сахарного тростника и выращиванием табака Большинство из них занимается своим промыслом самостоятельно, а некоторые нанимают работников, заключая с ними контракты разной продолжительности. Имеются записи и о том, кто кого нанимал, на какую работу, когда начался и закончился контракт. Собственно, круг занятий жителей острова крайне невелик и не меняется веками. Неудивительно поэтому, что в летописи подробно описывается каждое дело, будь то рыбная ловля или выпечка хлеба. Все предприниматели — уроженцы острова. Некоторые объединяются в кооперативы, и по записям можно установить, кто участвовал в деле, когда вступил и когда вышел из него, каким паем владел. Имеются краткие описания деятельности каждого предпринимателя или кооператива, сообщающие в том числе, когда было начато дело, когда и почему прекращено.

Исходные данные:

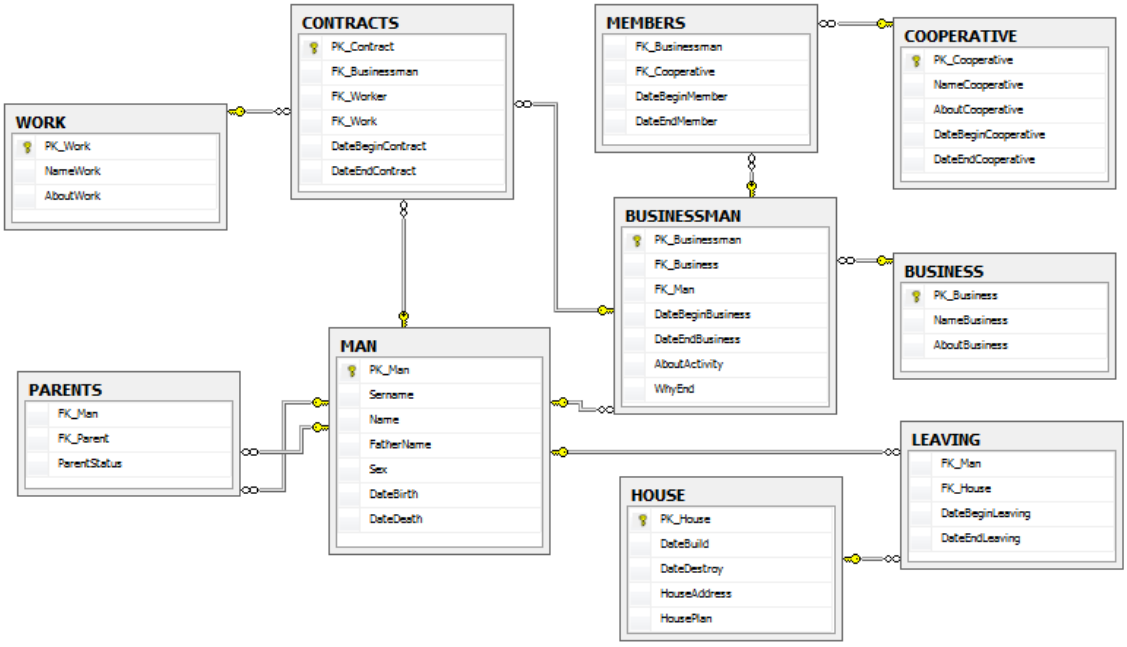


Рисунок 1 – Схема разработанной БД в третьей нормальной форме

Скрипт базы данных согласно варианту задания

```
USE MASTER
GO

-------------------------------------------------------------

IF EXISTS (SELECT * FROM sys.databases WHERE name = 'Island')
DROP DATABASE Island

-------------------------------------------------------------

CREATE DATABASE Island

-------------------------------------------------------------

USE Island
GO

-------------------------------------------------------------

create table MAN
(
	PK_Man int UNIQUE NOT NULL,
	Sername nvarchar(20) NOT NULL,
	Name nvarchar(20) NULL,
	FatherName nvarchar(20) NULL,
	Sex nvarchar(8) NOT NULL,
	DateBirth date NOT NULL,
	DateDeath date NULL,

	constraint PK_Man_Man primary key (PK_Man),

	constraint CH_Sex_Man check (sex in ('м','ж','муж','жен',
					'мужской','женский','мужчина','женщина',
					'М','Ж','Муж','Жен',
				'Мужской','Женский','Мужчина','Женщина')),
	constraint CH_DB_Man check (DateBirth<=GetDate()),
	constraint CH_DD_Man check (DateDeath<=GetDate()),
	constraint CH_DB_DD_Man check (DateBirth<=DateDeath)
)

-------------------------------------------------------------

create table HOUSE
(
	PK_House int UNIQUE NOT NULL,
	DateBuild date NOT NULL,
	DateDestroy date NULL,
	HouseAddress nvarchar(30) NOT NULL,
	HousePlan varbinary(max) NULL,

	constraint PK_House_House primary key (PK_House),

	constraint CH_DB_House check (DateBuild<=GetDate()),
	constraint CH_DD_House check (DateDestroy<=GetDate()),
	constraint CH_DB_DD_House check (DateBuild<=DateDestroy)
)

-------------------------------------------------------------
```

```
create table LEAVING
(
	FK_Man int NOT NULL,
	FK_House int NOT NULL,
	DateBeginLeaving date NOT NULL,
	DateEndLeaving date NULL,

	constraint FK_Man_Leaving foreign key (FK_Man) references MAN (PK_Man),
	constraint FK_House_Leaving foreign key (FK_House) references HOUSE
(PK_House),

	constraint CH_DBL_Leaving check (DateBeginLeaving<=GetDate()),
	constraint CH_DEL_Leaving check (DateEndLeaving<=GetDate()),
	constraint CH_DBL_DEL_Leaving check (DateBeginLeaving<=DateEndLeaving)
)

-------------------------------------------------------------

create table PARENTS
(
	FK_Man int NOT NULL,
	FK_Parent int NOT NULL,
	ParentStatus nvarchar(5) NOT NULL,

	constraint FK_Man_Parents foreign key (FK_Man) references MAN (PK_Man),
	constraint FK_Parent_Parents foreign key (FK_Parent) references MAN
(PK_Man),

	constraint CH_PS_Parents check (ParentStatus in
('мать','отец','Мать','Отец')),
)

-------------------------------------------------------------

create table BUSINESS
(
	PK_Business int UNIQUE NOT NULL,
	NameBusiness nvarchar(50) NOT NULL,
	AboutBusiness nvarchar(max) NULL,

	constraint PK_Business_Business primary key (PK_Business)
)

-------------------------------------------------------------

create table WORK
(
	PK_Work int UNIQUE NOT NULL,
	NameWork nvarchar(50) NOT NULL,
	AboutWork nvarchar(max) NULL,

	constraint PK_Work_Work primary key (PK_Work)
)

-------------------------------------------------------------
```

```
create table BUSINESSMAN
(
	PK_Businessman int UNIQUE NOT NULL,
	FK_Business int NOT NULL,
	FK_Man int NOT NULL,
	DateBeginBusiness date NOT NULL,
	DateEndBusiness date NULL,
	AboutActivity nvarchar(max) NULL,
	WhyEnd nvarchar(2000) NULL,

	constraint PK_Businessman_Businessman primary key (PK_Businessman),
	constraint FK_Business_Businessman foreign key (FK_Business) references
BUSINESS (PK_Business),
	constraint FK_Man_Businessman foreign key (FK_Man) references MAN (PK_Man),

	constraint CH_DBB_Businessman check (DateBeginBusiness<=GetDate()),
	constraint CH_DEB_Businessman check (DateEndBusiness<=GetDate()),
	constraint CH_DBB_DEB_Businessman check
(DateBeginBusiness<=DateEndBusiness)
)

-------------------------------------------------------------

create table CONTRACTS
(
	PK_Contract int UNIQUE NOT NULL,
	FK_Businessman int NOT NULL,
	FK_Worker int NOT NULL,
	FK_Work int NOT NULL,
	DateBeginContract date NOT NULL,
	DateEndContract date NULL,

	constraint PK_Contract_Contracts primary key (PK_Contract),
	constraint FK_Businessman_Contracts foreign key (FK_Businessman) references
BUSINESSMAN (PK_Businessman),
	constraint FK_Worker_Contracts foreign key (FK_Worker) references MAN
(PK_Man),
	constraint FK_Work_Contracts foreign key (FK_Work) references WORK
(PK_Work),

	constraint CH_DBC_Contract check (DateBeginContract<=GetDate()),
	constraint CH_DEC_Contract check (DateEndContract<=GetDate()),
	constraint CH_DBC_DEC_Contract check (DateBeginContract<=DateEndContract)
)

-------------------------------------------------------------

create table COOPERATIVE
(
	PK_Cooperative int UNIQUE NOT NULL,
	NameCooperative nvarchar(50) NOT NULL,
	AboutCooperative nvarchar(max) NULL,
	DateBeginCooperative date NOT NULL,
	DateEndCooperative date NULL,

	constraint PK_Cooperative_Cooperative primary key (PK_Cooperative),
	constraint CH_DBC_Cooperative check (DateBeginCooperative<=GetDate()),
	constraint CH_DEC_Cooperative check (DateEndCooperative<=GetDate()),
	constraint CH_DBC_DEC_Cooperative check
(DateBeginCooperative<=DateEndCooperative)
)

-------------------------------------------------------------
```

```
create table MEMBERS
(
	FK_Businessman int NOT NULL,
	FK_Cooperative int NOT NULL,
	DateBeginMember date NOT NULL,
	DateEndMember date NULL,

	constraint FK_Businessman_Members foreign key (FK_Businessman) references
BUSINESSMAN (PK_Businessman),
	constraint FK_Cooperative_Members foreign key (FK_Cooperative) references
COOPERATIVE (PK_Cooperative),

	constraint CH_DBM_Members check (DateBeginMember<=GetDate()),
	constraint CH_DEM_Members check (DateEndMember<=GetDate()),
	constraint CH_DBM_DEM_Members check (DateBeginMember<=DateEndMember)
)
```

Лабораторная работа №7

ТЕМА: Знакомство с основными особенностями программирования в MS SQL Server* средствами языка Transact-SQL*.

ЦЕЛЬ: Знакомство с особенностями построения запросов средствами встроенного языка Transact SQL при использовании переменными типа Table и Cursor. Знакомство с особенностями построения процедур и триггеров.

Ход работы:

Триггер – это хранимая процедура, которая выполняется автоматически при изменении таблицы SQL Server с использованием инструкции UPDATE, INSERT или DELETE.

В ходе данной работы создадим четыре триггера:

- для обеспечения целостности данных (откаты);
- с использованием функций для работы со строковыми переменными;
- с использованием условной конструкции IF;
- триггер с использованием цикла WHILE.

1 Триггер для обеспечения целостности данных (откаты).

Предположим, что требуется внести данные о заселении жителя в дом. Однако может возникнуть ситуация, что житель уже умер, следовательно, заселиться он уже никак не может. Для обеспечения целостности данных создадим триггер, который будет проверять, жив ли еще заселяющийся человек, и в случае, если это не так, триггер будет выполнять откат, то есть неверная запись фиксироваться не будет.

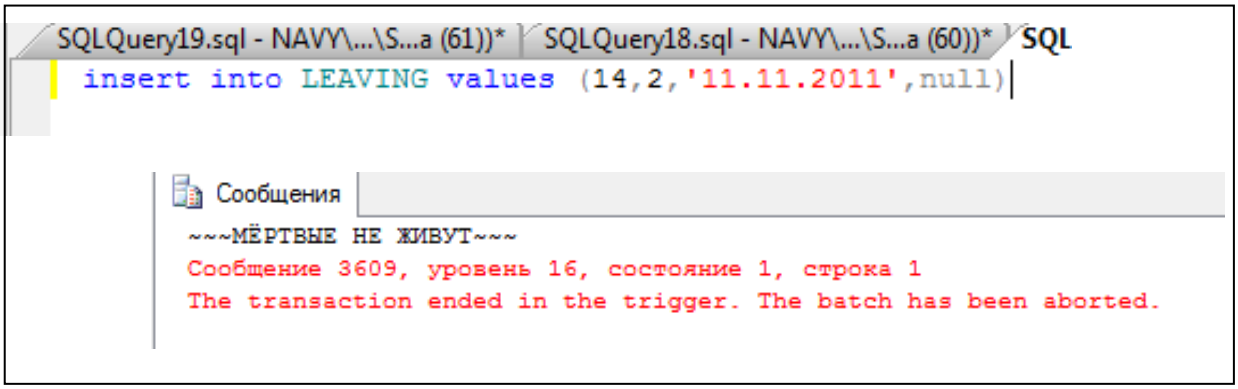
Листинг триггера (откат):

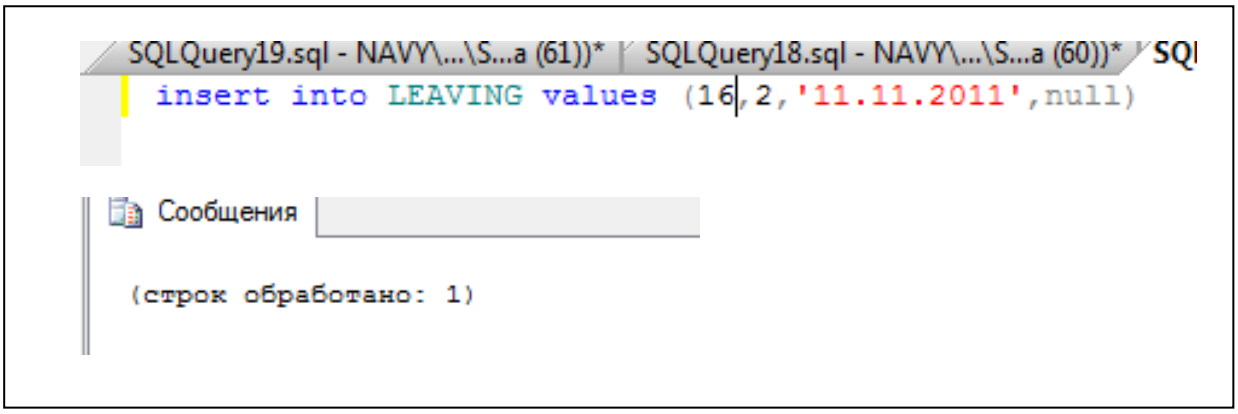
```
create trigger AddLeavingTrigger
on LEAVING
after INSERT
as
begin
	declare @ins_FK_Man int
	set @ins_FK_Man = (select FK_Man from inserted)

	declare @dead date
	set @dead = (select MAN.DateDeath from MAN where PK_Man = @ins_FK_Man)

	if (@dead is not null)
		begin
			ROLLBACK TRAN
			print ('~~~МЁРТВЫЕ НЕ ЖИВУТ~~~')
		end

	end
```

Проверим работу триггера. Рассмотрим жителей с номерами 14 (уже умер) и 16 (еще жив):

13	13	Стреленко	Стрелка	Михалыч	м	1990-03-04	2011-11-16
14	14	Иванов	Павел	Иванович	м	1945-01-01	1990-02-04
15	15	Титенков	Андрей	NULL	м	2005-01-01	NULL
16	16	Жаховский	Утюг	NULL	м	1980-11-11	NULL
17	17	Штыкова	Елена	NULL	ж	2005-02-02	NULL

Результаты выполнения запроса на добавление записей:





Как видим, запрос на заселение мертвого человека был завершен в триггере и запись не добавилась, а запрос на добавление живого человека выполнился успешно, запись появилась в таблице LEAVING:

	FK_Man	FK_House	DateBeginLeaving	DateEndLeaving
15	10	14	2011-01-01	NULL
16	16	2	2011-11-11	NULL

Следовательно, триггер работает правильно.

2 Триггер с использованием функций для работы со строковыми переменными.

Создадим простой, но весьма актуальный триггер на добавление записей в таблицу ЖИТЕЛЬ (MAN), который будет исправлять регистр букв в фамилии, имени и отчестве жителя: они должны начинаться с заглавной буквы, остальные же буквы должны быть прописными.

Листинг триггера (строковые функции):

```
create trigger UpLettersTrigger
instead of insert
as
begin
	declare @ins_PK int
	declare @ins_S nvarchar(20)
	declare @ins_N nvarchar(20)
	declare @ins_F nvarchar(20)
	declare @ins_Sex nvarchar(8)
	declare @ins_DB date
	declare @ins_DD date

	set @ins_PK = (select PK_Man from inserted)
	set @ins_S = (select lower(Sername) from inserted)
	set @ins_N = (select lower(Name) from inserted)
	set @ins_F = (select lower(FatherName) from inserted)
	set @ins_Sex = (select Sex from inserted)
	set @ins_DB = (select DateBirth from inserted)
	set @ins_DD = (select DateDeath from inserted)

	declare @new_S nvarchar(20)
	set @new_S = upper(substring(@ins_S,1,1)) + substring(@ins_S,2,LEN(@ins_S))

	declare @new_N nvarchar(20)
	declare @new_F nvarchar(20)

	if (@ins_N is not null)
		begin
				set @new_N = upper(substring(@ins_N,1,1)) +
				substring(@ins_N,2,LEN(@ins_N))
		end
	else
		begin
			set @new_N = @ins_N
		end

	if (@ins_F is not null)
		begin
			set @new_F = upper(substring(@ins_F,1,1)) +
			substring(@ins_F,2,LEN(@ins_F))
		end
	else
		begin
			set @new_F = @ins_F
		end

	insert into MAN values
(@ins_PK,@new_S,@new_N,@new_F,@ins_Sex,@ins_DB,@ins_DD)


	end
```

Проверим работу триггера. Напишем несколько запросов на добавление записи, причем проверяемые поля с ошибками, учтем также нулевые поля:

```
insert into MAN values (22,'литвиенко','андрей','семенович','м','12.12.1960',null)
insert into MAN values (23,'маринов','михаил',null,'м','12.12.1960',null)
insert into MAN values (24,'вЕсЕлОв','чУдАк',null,'м','12.12.1960',null)
```

Теперь посмотрим, какие строки добавились в таблицу MAN:

20	20	Балахнин	Игорь	Романович	м	1940-01-01	NULL
21	21	Петренко	Лариса	Игоревна	ж	1980-12-12	NULL
22	22	Литвиенко	Андрей	Семенович	м	1960-12-12	NULL
23	23	Маринов	Михаил	NULL	м	1960-12-12	NULL
24	24	Веселов	Чудак	NULL	м	1960-12-12	NULL

Как видим, все буквы в правильном регистре, нулевые поля проблем не вызвали. Следовательно, триггер работает правильно.

3 Триггер с использованием условной конструкции IF.

Создадим триггер, который будет проверять правильность заполнения таблицы РОДИТЕЛИ (PARENTS). Записи этой таблицы должны удовлетворять очевидным требованиям.

1) Родитель и ребенок должны быть разными людьми.
2) Статус должен соответствовать полу (отец должен быть мужчиной, а мать – женщиной).
3) Родителей должно быть не больше двух.
4) Не должно быть два отца или две матери.
5) Один и тот же человек не может быть одновременно и матерью, и отцом.
6) Должно выполняться соответствие возрастов: родитель должен быть старше ребенка. Также установим следующие рамки: разница возраста родителя и ребенка не может быть меньше 10 лет и не может превышать 90 лет.

Если добавляемая запись удовлетворяет всем этим требованиям, триггер разрешает занести ее в таблицу. Если же нет – триггер сообщит об ошибке и запретит внесение записи.

Листинг триггера (конструкция IF):

```
create trigger AddParentsTrigger
instead of insert
```

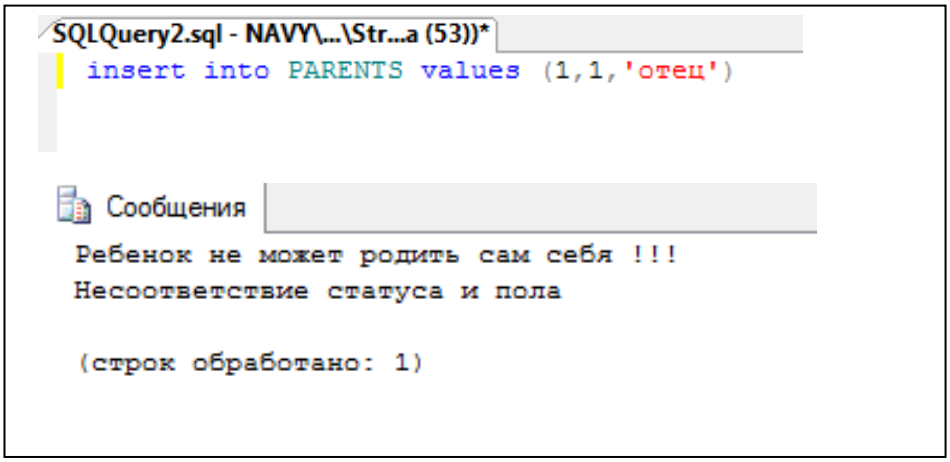
```
as
begin
	-- Заносим в переменные вставляемые данные

	declare @FK_ins_son int
	set @FK_ins_son = (select FK_Man from inserted)

	declare @FK_ins_parent int
	set @FK_ins_parent = (select FK_Parent from inserted)

	declare @ins_status nvarchar(5)
	set @ins_status = (select ParentStatus from inserted)

	declare @check_ins int -- 0 нормально, 1 ошибка
	set @check_ins = 0

	-- Проверяем на совпадение родителя и ребенка

	if (
		@FK_ins_parent = @FK_ins_son
		)
		begin
			print (' Ребенок не может родить сам себя !!! ')
			set @check_ins = 1
		end

	-- Проверяем соответствие статуса и пола

	declare @parent_sex nvarchar(5)
	set @parent_sex = (select MAN.Sex
					from MAN
					where PK_Man = @FK_ins_parent)

	if	(
		@parent_sex in
('ж','жен','женский','женщина','Ж','Жен','Женский','Женщина')
		and @ins_status in ('отец','Отец')
		)
		begin
			print ('Несоответствие статуса и пола');
			set @check_ins = 1
		end

	if (
		@parent_sex in
('м','муж','мужской','мужчина','М','Муж','Мужской','Мужчина')
		and @ins_status in ('мать','Мать')
		)
		begin
			print ('Несоответствие статуса и пола');
			set @check_ins = 1
		end

	-- Подсчитаем количество родителей

	declare @count_parents int
	set @count_parents = (select COUNT(FK_Man) from PARENTS where FK_Man =
@FK_ins_son)

	-- Если родителей уже двое, больше данных заносить нельзя

	if (
		@count_parents = 2
		)
```

```
		begin
			print ('Уже внесены данные об обоих родителях');
			set @check_ins = 1
		end

	-- Если один родитель уже есть, нужно проверить какой

	if (
		@count_parents = 1
		)
		begin
			if	(
				(select PARENTS.FK_Parent
					from PARENTS
					where FK_Man = @FK_ins_son) = @FK_ins_parent
									-- чтобы не один и тот же
									OR
				(select PARENTS.ParentStatus
					from PARENTS
					where FK_Man = @FK_ins_son) = @ins_status
								-- чтобы не два раза один статус

				)
				begin
					print('Неверные данные о родителе')
					set @check_ins = 1
				end
		end

	-- Если всё ещё не было ошибки, имеет смысл проверить соответствие
-- возрастов

	if (
		@check_ins = 0
		)
		begin -- Проверить соответствие возрастов
			declare @date_parent_B date
			declare @date_parent_D date
			declare @date_man date
			set @date_parent_B = (select MAN.DateBirth
									from MAN
									where @FK_ins_parent =
PK_Man)
			set @date_parent_D = (select MAN.DateDeath
									from MAN
									where @FK_ins_parent =
PK_Man)
			set @date_man =		(select MAN.DateBirth
									from MAN
									where @FK_ins_son = PK_Man)
			if (
				(YEAR(@date_man)-YEAR(@date_parent_B)) < 10
				OR
				@date_man > @date_parent_D
				OR
				YEAR(@date_man)-YEAR(@date_parent_B) > 90

				)
				begin -- Несоответствие возрастов
					print('Несоответствие возрастов')
					set @check_ins = 1
				end

		end
```

```
-- Если @check_ins = 0, то ошибок не было, добавляем запись.
-- В противном случае выводим сообщение о том, что запись не добавлена.


if (
      @check_ins = 0
      )
      begin
            insert into PARENTS
            values (@FK_ins_son,@FK_ins_parent,@ins_status)
            print('Запись добавлена')
      end

end
```

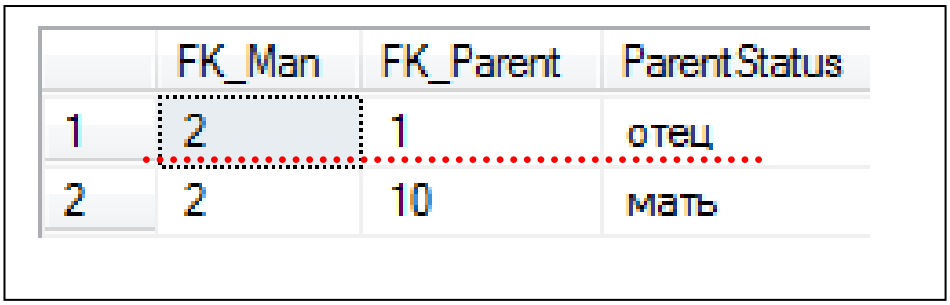
Проверим работу триггера. Ниже приведены различные варианты некорректных данных.

1) Ребенок и родитель – один человек



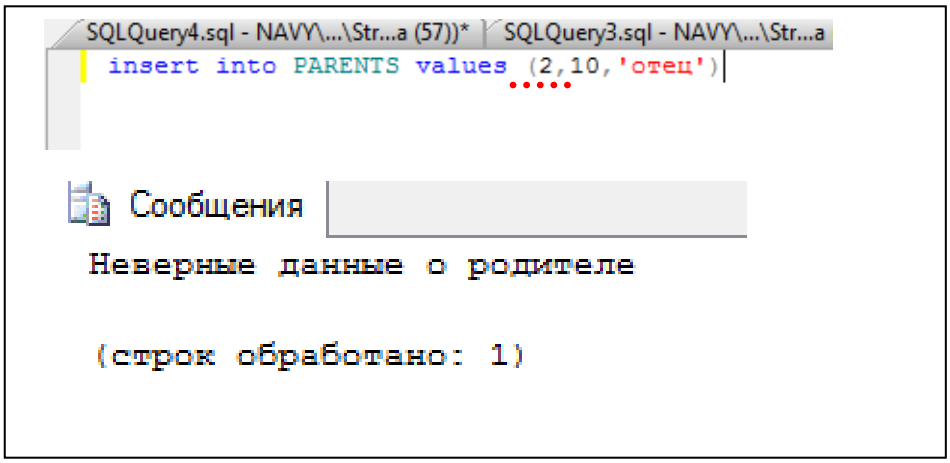
Строка не добавлена.

2) Попытка внести данные об отце, когда у этого ребенка отец уже есть.



	FK_Man	FK_Parent	ParentStatus
1	2	1	отец
2	2	10	мать

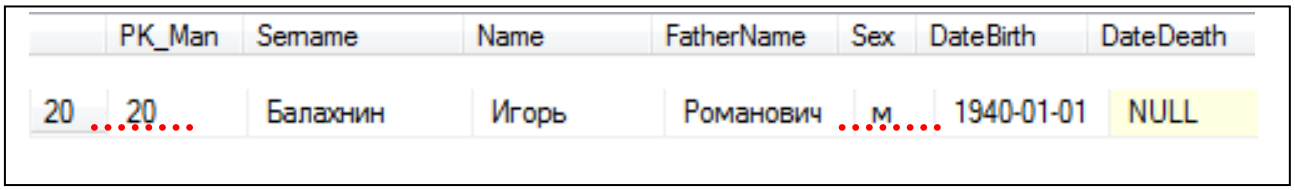
У ребенка с номером 2 есть отец.



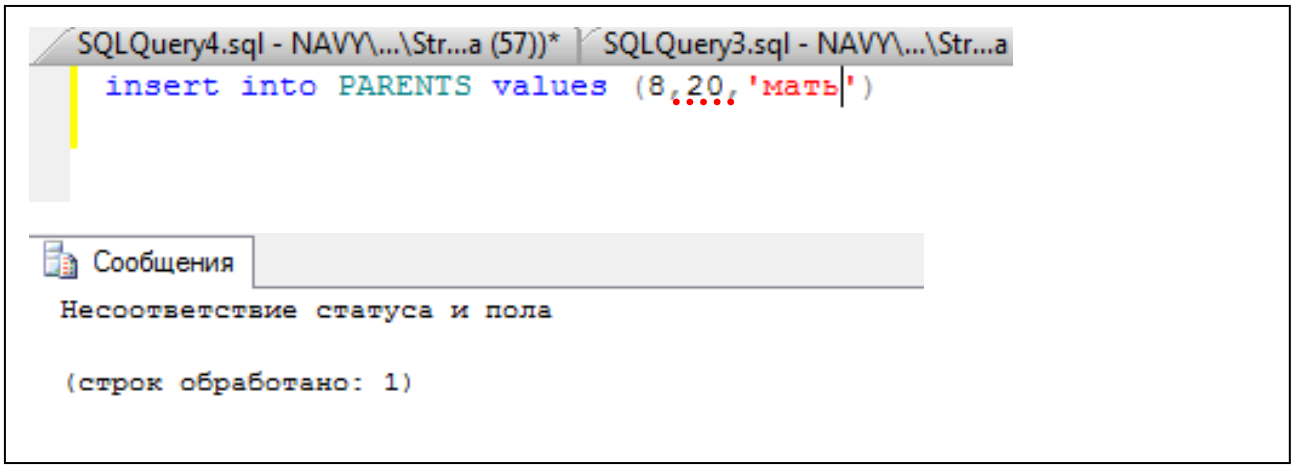
Строка не добавлена.

3) Попытка добавить мужчину со статусом «мать».

Житель с номером 20 мужского пола.



	PK_Man	Semame	Name	FatherName	Sex	DateBirth	DateDeath
20	20	Балахнин	Игорь	Романович	м	1940-01-01	NULL

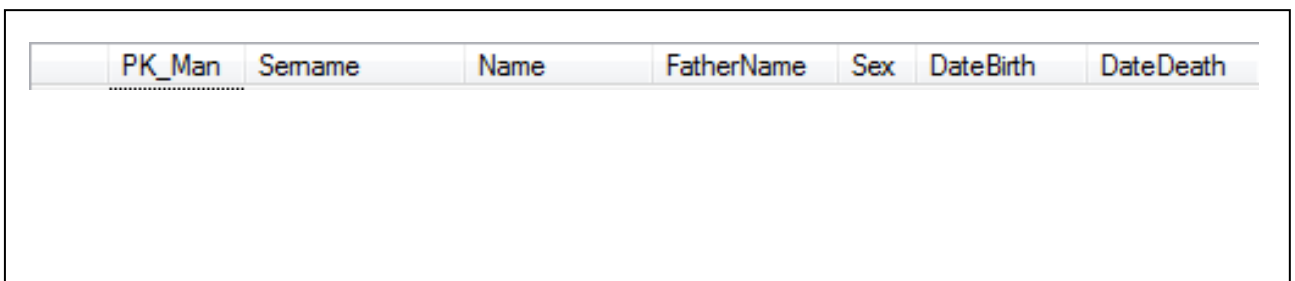


Строка не добавлена.

4) Попытка внести данные о родителе, который младше своего ребенка.

Год рождения ребенка с номером 8 – 1970 г.

Год рождения родителя с номером 17 – 2005 г.



PK_Man	Semame	Name	FatherName	Sex	DateBirth	DateDeath

8	8	Волкова	Александра	Игоревна	ж	1970-10-10	NULL
9	9	Кадушко	Елена	NULL	ж	1965-06-04	1987-01-01
10	10	Иванова	Иванина	Ивановна	ж	1950-01-22	1970-01-01
11	11	Зубревич	Елена	Сергеевна	ж	1991-11-11	NULL
12	12	Провалинская	Вика	Петровна	ж	1971-10-11	1990-01-01
13	13	Стреленко	Стрелка	Михалыч	м	1990-03-04	2011-11-16
14	14	Иванов	Павел	Иванович	м	1945-01-01	1990-02-04
15	15	Титенков	Андрей	NULL	м	2005-01-01	NULL
16	16	Жаховский	Утюг	NULL	м	1980-11-11	NULL
17	17	Штыкова	Елена	NULL	ж	2005-02-02	NULL

SQLQuery4.sql - NAVY\...\Str...a (57))* SQLQuery3.sql - NAVY\...\Str...a (56))*

```
insert into PARENTS values (8,17,'мать')
```

Сообщения

```
Несоответствие возрастов

(строк обработано: 1)
```

Строка не добавлена.

Дальнейшие попытки добавления ошибочных строк также были запрещены.

Введем верные данные:

SQLQuery4.sql - NAVY\...\Str...a (57))* SQLQuery3.sql - NAVY\...\Str...a (56))

```
insert into PARENTS values (11,1,'отец')
```

Сообщения

```
(строк обработано: 1)
Запись добавлена

(строк обработано: 1)
```

На этот раз строка успешно записана в таблицу PARENTS:

	FK_Man	FK_Parent	ParentStatus
12	8	1	отец
13	11	1	отец
14	11	8	мать
15	12	10	мать

Таким образом, триггер работает корректно.

4 Триггер с использованием цикла WHILE.

Предположим, что на острове существует традиция – когда завершается строительство нового дома, проводится лотерея, кому будет принадлежать этот дом. Создадим триггер, проводящий лотерею среди всех жителей острова. Будем осуществлять проверку, новый ли дом (так как летописец может вносить данные и о старых домах), а затем, проведя лотерею, записываем победителя в таблицу ПРОЖИВАНИЕ (LEAVING) вместе с выигранным им домом.

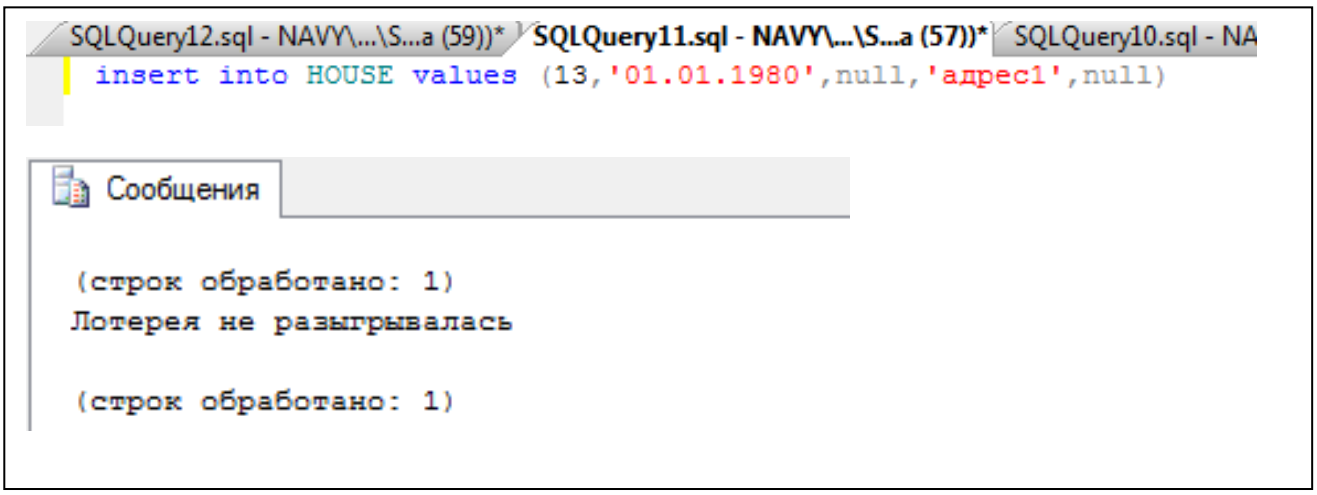
Листинг триггера (цикл WHILE):

```
create trigger LotteryTrigger
instead of insert
as
begin
	declare @ins_PK_H int
	set @ins_PK_H = (select PK_House from inserted)

	declare @ins_DB date
	set @ins_DB = (select DateBuild from inserted)

	declare @ins_DD date
	set @ins_DD = (select DateDestroy from inserted)

	declare @ins_HA nvarchar(30)
	set @ins_HA = (select HouseAddress from inserted)

	declare @ins_HP varbinary(max)
	set @ins_HP = (select HousePlan from inserted)

	if (
		@ins_DD is null
		AND
		year(@ins_DB) = YEAR (getdate())
		)
		begin			-- если дом новый проводим лотерею
			declare @count_man int
			set @count_man = (select COUNT(MAN.PK_Man) from MAN)

			declare @counter int		-- циклсчетчик ключа жителя
			set @counter = 1

			declare @check_iter int
```

```
                  set @check_iter = 0

                  while @check_iter = 0                -- бесконечный
                        begin
                              declare @buf float
                              set @buf = @counter*10*RAND()+3*RAND()
                              if (@buf > 10 and @buf < 15)
                                    begin
                                          break -- завершение цикла
                                                -- при выполнении условия
                                    end
                              else
                                    begin
                                          if (@counter = @count_man)
                                                begin
                                                      set @counter = 1
                                                end
                                          else
                                                begin
                                                      set @counter = @counter + 1
                                                end
                                    end
                        end

                  print ('Лотерея розыграна')
insert into HOUSE values (@ins_PK_H,@ins_DB,@ins_DD,@ins_HA,@ins_HP)
insert into LEAVING values (@counter,@ins_PK_H,@ins_DB,null)
                  end
            else
                  begin
insert into HOUSE values (@ins_PK_H,@ins_DB,@ins_DD,@ins_HA,@ins_HP)
                  print ('Лотерея не разыгрывалась')
                  end

            end
```

Проверим работу триггера.

1) В таблицу HOUSE заносятся данные о здании, построенном в 1980 году.



Лотерея не проводилась, так как здание не является только что построенным. Данные в таблицу LEAVING не заносились, однако в таблицу HOUSE запись добавлена (как и положено):

	PK_House	DateBuild	DateDestroy	HouseAddress	HousePlan
1	1	1945-01-01	2000-02-02	Молодежная-3	NULL
2	2	1990-01-01	2010-01-01	Парковая-6	NULL
3	3	1900-01-01	2000-01-01	Калинина-5	NULL
4	4	2007-11-11	2011-01-01	Молодежная-1	NULL
5	5	1980-11-11	NULL	Парковая-24	NULL
6	6	1940-11-11	NULL	Калинина-8	NULL
7	7	1990-11-11	NULL	Молодежная-4	NULL
8	8	1976-11-11	NULL	Парковая-12	NULL
9	9	1930-11-11	NULL	Молодежная-2	NULL
10	10	1903-11-11	NULL	Молодежная-5	NULL
11	11	1910-11-11	NULL	Калинина-7	NULL
12	12	1999-01-01	NULL	адрес1	NULL
13	13	1980-01-01	NULL	адрес1	NULL

Данные занесены в таблицу HOUSE

	FK_Man	FK_House	DateBeginLeaving	DateEndLeaving
1	1	1	1952-11-11	NULL
2	3	5	2011-11-16	NULL
3	4	8	2000-11-11	NULL
4	5	7	2005-11-11	NULL
5	7	3	1995-11-11	2000-11-11
6	8	6	1999-11-11	NULL
7	12	9	1971-11-11	NULL
8	14	10	1960-11-11	NULL
9	15	2	2005-11-11	NULL
10	17	2	2005-11-11	2010-11-11
11	7	11	2000-11-11	NULL
12	17	4	2010-11-11	NULL
13	13	5	1990-11-11	2011-11-16
14	4	12	1999-01-01	NULL

2) В таблицу HOUSE заносятся данные о здании, построенном в 2011 году (новое здание).

SQLQuery12.sql - NAVY\...\S...a (59))* | SQLQuery11.sql - NAVY\...\S...a (57))* | SQLQuery10.sql - NAV\

```
insert into HOUSE values (14,'01.01.2011',null,'адрес5',null)
```

Сообщения

```
Лотерея розыграна

(строк обработано: 1)

(строк обработано: 1)

(строк обработано: 1)
```

Лотерея разыграна, так как здание является только что построенным. Данные занесены в таблицу HOUSE (как и положено), а также в таблицу LEAVING вместе с новым владельцем дома:

Данные занесены
в таблицу HOUSE

	PK_House	DateBuild	DateDestroy	HouseAddress	HousePlan
1	1	1945-01-01	2000-02-02	Молодежная-3	NULL
2	2	1990-01-01	2010-01-01	Парковая-6	NULL
3	3	1900-01-01	2000-01-01	Калинина-5	NULL
4	4	2007-11-11	2011-01-01	Молодежная-1	NULL
5	5	1980-11-11	NULL	Парковая-24	NULL
6	6	1940-11-11	NULL	Калинина-8	NULL
7	7	1990-11-11	NULL	Молодежная-4	NULL
8	8	1976-11-11	NULL	Парковая-12	NULL
9	9	1930-11-11	NULL	Молодежная-2	NULL
10	10	1903-11-11	NULL	Молодежная-5	NULL
11	11	1910-11-11	NULL	Калинина-7	NULL
12	12	1999-01-01	NULL	адрес1	NULL
13	13	1980-01-01	NULL	адрес1	NULL
14	14	2011-01-01	NULL	адрес5	NULL

Данные занесены
в таблицу LEAVING.
Победитель лотереи –
житель с номером 10.

Результаты | Сообщения

	FK_Man	FK_House	DateBeginLeaving	DateEndLeaving
1	1	1	1952-11-11	NULL
2	3	5	2011-11-16	NULL
3	4	8	2000-11-11	NULL
4	5	7	2005-11-11	NULL
5	7	3	1995-11-11	2000-11-11
6	8	6	1999-11-11	NULL
7	12	9	1971-11-11	NULL
8	14	10	1960-11-11	NULL
9	15	2	2005-11-11	NULL
10	17	2	2005-11-11	2010-11-11
11	7	11	2000-11-11	NULL
12	17	4	2010-11-11	NULL
13	13	5	1990-11-11	2011-11-16
14	4	12	1999-01-01	NULL
15	10	14	2011-01-01	NULL

Таким образом, триггер работает корректно.

Вывод:

В результате выполнения работы созданы различные триггеры согласно заданию и приведены примеры их работы. Тестирование триггеров показало, что они работают корректно и помогают избежать грубых ошибок нарушения целостности данных.