**Лабораторная работа № 11. Особенности использования транзакций**

**КУРНОСЕНКО СОФЬЯ**

**Транзакция** − это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменяющих базу данных, чтобы при выполнении этой совокупности операторов они или все выполнились или все не выполнились.

Основные свойства транзакции: *атомарность* (операторы изменения БД, включенные в транзакцию, либо выполнятся все, либо не выполнится ни один); *согласованность* (транзакция должна фиксировать новое согласованное состояние БД); *изолированность* (отсутствие взаимного влияния параллельных транзакций на результаты их выполнения); *долговечность* (изменения в БД, выполненные и зафиксированные транзакцией, могут быть отменены только с помощью новой транзакции).

1. Разработать сценарий, демонстрирующий работу в режиме *неявной* транзакции.

Проанализировать пример, приведенный справа, в котором создается таблица Х, и создать сценарий для другой таблицы.

***→ Теория (методичка)***

Режим *неявной транзакции* может быть включен для текущего соединения с сервером БД с помощью специальной инструкции:

**SET IMPLICIT\_TRANSACTIONS ON**

Обратное переключение осуществляется с использованием ключевого слова OFF вместо ON.

Неявная транзакция начинается, если выполняется один из следующих операторов:

CREATE, DROP; ALTER TABLE; INSERT, DELETE, UPDATE, SELECT, TRUNCATE TABLE; OPEN, FETCH; GRANT (выдача разрешений), REVOKE (запрещение разрешений).

Неявная транзакция продолжается до тех пор, пока не будет выполнен оператор фиксации (COMMIT) или оператор отката (ROLLBACK) транзакции.

В примере ниже неявная транзакция стартует при выполнении оператора CREATE TABLE и завершается фиксацией изменений с помощью оператора COMMIT.

После этого осуществляется возврат в режим автофиксации (инструкция SET OFF). Созданная таблица c именем X сохранилась.

**set nocount on**

**if exists (select \* from SYS.OBJECTS** -- таблица X есть?

**where OBJECT\_ID= object\_id(N'DBO.X') )**

**drop table X;**

**declare @c int, @flag char = 'c';** -- commit или rollback?

**SET IMPLICIT\_TRANSACTIONS ON** -- включ. режим неявной транзакции

**CREATE table X(K int );** -- начало транзакции

**INSERT X values (1),(2),(3);**

**set @c = (select count(\*) from X);**

**print 'количество строк в таблице X: ' + cast( @c as varchar(2));**

**if @flag = 'c' commit; --** завершение транзакции: фиксация

**else rollback; --** завершение транзакции: откат

**SET IMPLICIT\_TRANSACTIONS OFF --** выключ. режим неявной транзакции

**if exists (select \* from SYS.OBJECTS** -- таблица X есть?

**where OBJECT\_ID= object\_id(N'DBO.X') )**

**print 'таблица X есть';**

**else print 'таблицы X нет'**

Если в данном сценарии изменить значение переменной **@flag = 'r',** то будетвыполняться оператор ROLLBACK, что приведет к отмене всех изменений в БД, осуществленных последней транзакцией (созданная таблица X не сохранится).

***→ Выполнение заданий***

-- 1

set nocount on;

-- если таблица #tempTable существует - удаляем её

use tempdb;

if exists (select \* from sys.objects

where object\_id=object\_id('dbo.#tempTable'))

drop table #tempTable;

declare @amountOfRows int;

set implicit\_transactions on; -- вкл режим неявной транзакции

-- начало транзакции, хотя оператора begin tran мы не используем

create table #tempTable(number int);

insert #tempTable values (1), (2), (3);

set @amountOfRows = (select count(\*) from #tempTable);

print 'Кол-во строк в таблице #tempTable : ' + cast(@amountOfRows as varchar(2));

commit; -- завершение транзакции: фиксация

-- rollback; -- завершение транзакции: откат

set implicit\_transactions off; -- выкл режим неявной транзакции

if exists(select \* from sys.objects

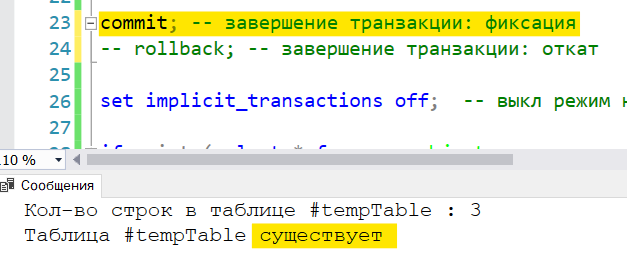
where object\_id=object\_id('dbo.#tempTable'))

print 'Таблица #tempTable существует';

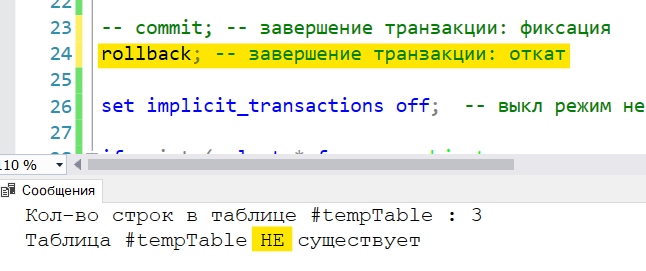
else

print 'Таблица #tempTable НЕ существует';

COMMIT:



ROLLBACK:



2.Разработать сценарий, демонстрирующий свойство *атомарности* *явной* транзакции на примере базы данных X\_UNIVER.

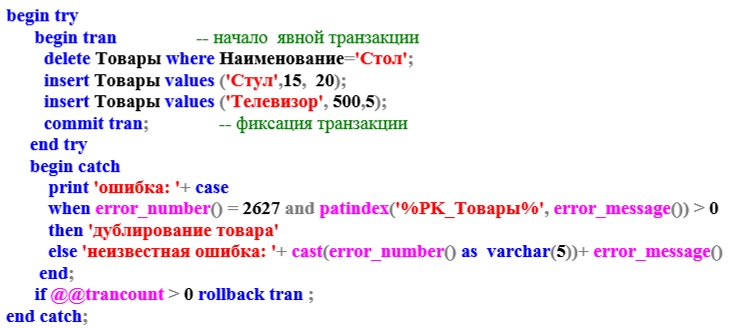
В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных операторов модификации таблиц.

***→ Теория (методичка)***

Переключение в режим *явной транзакции* осуществляется с помощью оператора BEGIN TRANSACTION. Транзакцию должен завершать один из операторов: COMMIT TRAN или ROLLBACK TRAN. После завершения явной транзакции происходит возврат в исходный режим (автофиксации или неявной транзакции).

Пусть требуется осуществить изменения в таблице **Товары** так, чтобы в случае ошибки изменения не были осуществлены.



Транзакция начинается внутри TRY-блока и в случае успешного выполнения завершается оператором COMMIT.

Если при выполнении возникла ошибка, то в CATCH-блоке формируется и выводится соответствующее сообщение, а затем выполняется откат (ROLLBACK).

Системная функция @@TRANCOUNT возвращает уровень вложенности транзакции. (если значение больше нуля, то транзакция не завершена).

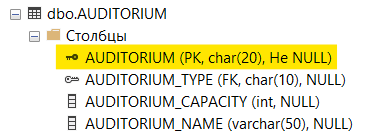
Встроенная функция PATINDEX определяет в строке позицию первого символа подстроки, заданную шаблоном. С помощью этой функции в тексте сообщения об ошибке отыскивается имя ограничения целостности.

Изменения в таблице будут осуществлены только в случае отсутствия ошибок в операторах удаления и вставки.

***→ Выполнение заданий***

**Атомарность** – все команды в транзакции либо полностью выполняются, и соответственно, фиксируются все изменения данных, либо ничего не выполняется и ничего не фиксируется.

Поставим значение null в столбец первичного ключа, чтобы вызвать ошибку:

******

use UNIVERSITY;

begin try

begin tran;

insert AUDITORIUM values('000-1', 'ЛК', 89, '000-1');

insert AUDITORIUM values(null, 'ЛК', 89, 'null'); -- ошибка, т.к. первый столбец - первичный ключ

commit tran;

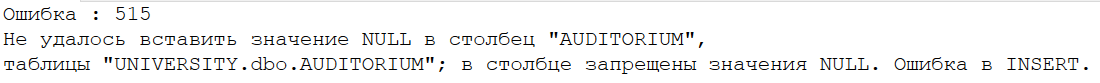
end try

begin catch

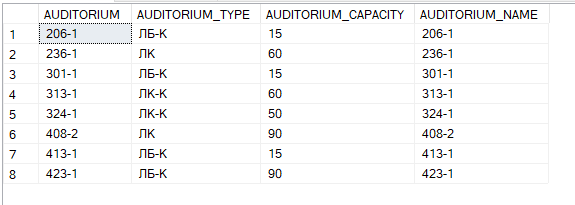
print 'Ошибка : ' + cast(error\_number() as varchar(5)) + char(10) + error\_message();

if @@trancount > 0 rollback tran; -- откат вложенной транзакции

end catch;

******

При этом, хотя первая инструкция (insert AUDITORIUM values('000-1', 'ЛК', 89, '000-1');) в транзакции и была правильной, но в таблице мы не увидим новой строки, так как сделали откат из-за ошибки (insert AUDITORIUM values(null, 'ЛК', 89, null);):

******

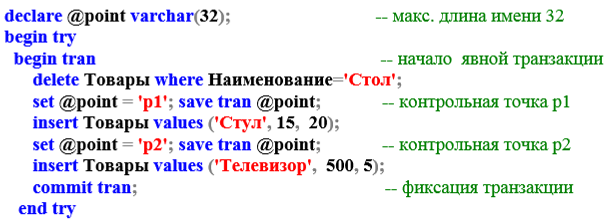
3.Разработать сценарий, демонстрирующий применение оператора SAVE TRAN на примере базы данных X\_UNIVER.

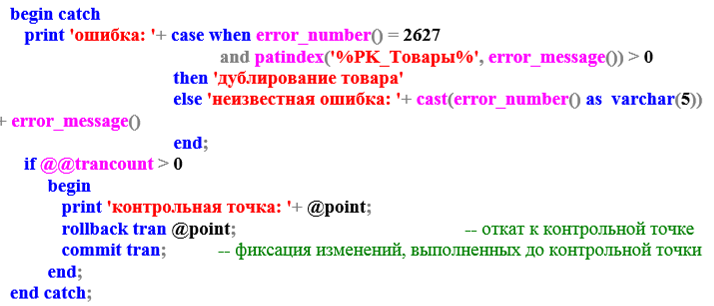
В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных контрольных точек и различных операторов модификации таблиц.

***→ Теория (методичка)***

Если транзакция состоит из нескольких независимых блоков операторов T-SQL, изменяющих базу данных, то может быть использован оператор SAVE TRANSACTION, формирующий *контрольную точку* транзакции.





***→ Выполнение заданий***

use UNIVERSITY;

declare @point varchar(32);

begin try

begin tran

insert AUDITORIUM values('000-1', 'ЛК', 89, '000-1');

set @point = 'p1'; -- промежуточная точка сохранения

save tran @point;

insert AUDITORIUM values(null, 'ЛК', 89, 'null'); -- ошибка, т.к. первый столбец - первичный ключ

commit tran

end try

begin catch

print 'Ошибка : ' + cast(error\_number() as varchar(5)) + char(10) + error\_message();

if @@trancount > 0

begin

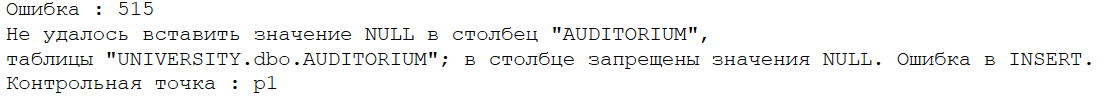
print 'Контрольная точка : ' + @point;

rollback tran @point; -- откат к контрольной точке

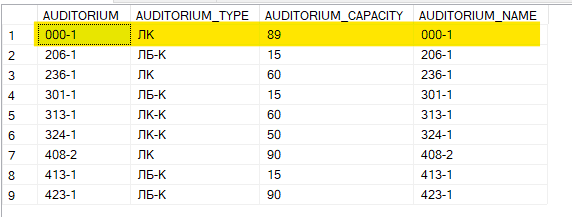
commit tran; -- фиксация изменений до контрольной точки

end;

end catch

******

В таблице AUDITORIUM сохранилась первая правильная операция вставки строки, благодаря откату к последней контрольной точке и фиксации изменений:



4.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности READ UNCOMMITED, сценарий B – явную транзакцию с уровнем изолированности READ COMMITED (по умолчанию).

Сценарий A должен демонстрировать, что уровень READ UNCOMMITED допускает неподтвержденное, неповторяющееся и фантомное чтение.

***→ Теория (методичка)***

Пусть имеются две параллельные транзакции **А** и **В** для базы данных ПРОДАЖИ:

**-- A ---**

**set transaction isolation level READ UNCOMMITTED**

**begin transaction**

**-------------------------- t1 ------------------**

**select @@SPID, 'insert Товары' 'результат', \* from Товары**

**where Наименование = 'Блокнот';**

**select @@SPID, 'update Заказы' 'результат', Наименование\_товара,**

**Цена\_продажи from Заказы where Наименование\_товара = 'Блокнот';**

**commit;**

**-------------------------- t2 -----------------**

**--- B --**

**begin transaction**

**select @@SPID**

**insert Товары values ('Блокнот', 2, 80);**

**update Заказы set Наименование\_товара = 'Блокнот'**

**where Наименование\_товара = 'Стол'**

**-------------------------- t1 --------------------**

**-------------------------- t2 --------------------**

**rollback;**

Здесь системная функция @@SPID возвращает системный идентификатор процесса, назначенный сервером текущему подключению.

При параллельных транзакциях могут возникать три проблемы.

*Неподтвержденное чтение.* До момента t1 транзакцией B выполняются два оператора: INSERT и UPDATE. Эти операторы изменяют таблицы БД, но до момента времени t2 не фиксируют и не откатывают эти изменения. После момента t1 транзакция A считывает содержимое таблиц, измененных транзакцией B и «видит» измененные или добавленные строки. При этом изменения остаются до момента t2 в неподтвержденном состоянии, т. е. могут быть как зафиксированными, так и отмененными.

*Неповторяющееся чтение.* Одна транзакция читает данные несколько раз, а другая изменяет те же данные между двумя операциями чтения в первом процессе. По этой причине данные, прочитанные в различных операциях, будут разными.

*Фантомное чтение.* Две последовательные операции чтения могут получать различные значения, т. к. дополнительные строки, называемые фантом­ными, могут добавляться другими транзакциями.

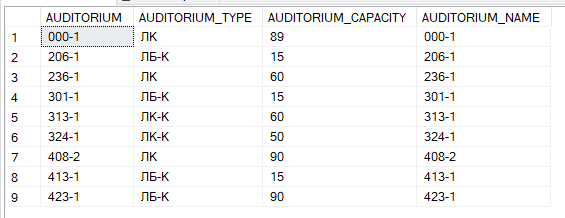
Чтобы такие проблемы не возникали, определяются различные уровни изолированности: READ COMMITED, REPEATABLE READ, SERIALIZABLE и др.

Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа: последовательно оба сценария до отметки t1; последовательно оба сценария c отметки t1 до отметки t2; последовательно оба сценария c отметки t2 и до конца.

***→ Выполнение заданий***

**Изоляция** – механизм предоставления доступа к данным. Транзакция изолирует данные, с которыми она работает, для того чтобы другие транзакции получали только согласованные данные.

Таблица до транзакций:



Скрипт:

use UNIVERSITY;

-- A --

set transaction isolation level READ UNCOMMITTED

begin tran

--------------------- t1 ---------------------

select @@spid as [id A (insert)],

'insert AUDITORIUM''результат',

\*

from AUDITORIUM

where AUDITORIUM='111-1';

select @@spid as [id A (update)],

'update AUDITORIUM''результат',

\*

from AUDITORIUM

where AUDITORIUM\_CAPACITY=100;

commit;

--------------------- t2 ---------------------

-- B --

begin tran

select @@spid as [id B];

insert AUDITORIUM values('111-1', 'ЛК', 89, '111-1');

update AUDITORIUM set AUDITORIUM\_CAPACITY=100

where AUDITORIUM\_CAPACITY=89;

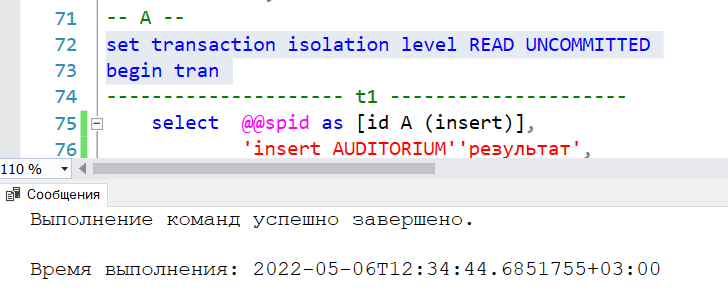
--------------------- t1 ---------------------

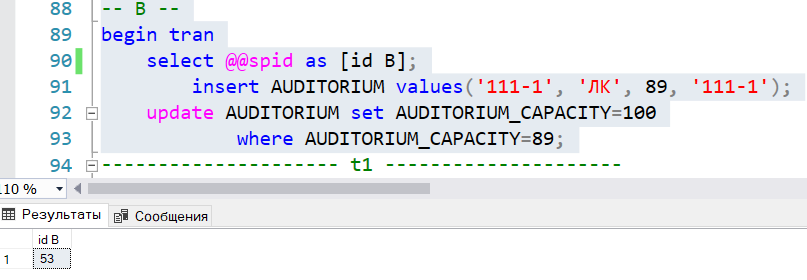
--------------------- t2 ---------------------

rollback;

Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа (выделяем часть скрипта и нажимаем «Выполнить»):

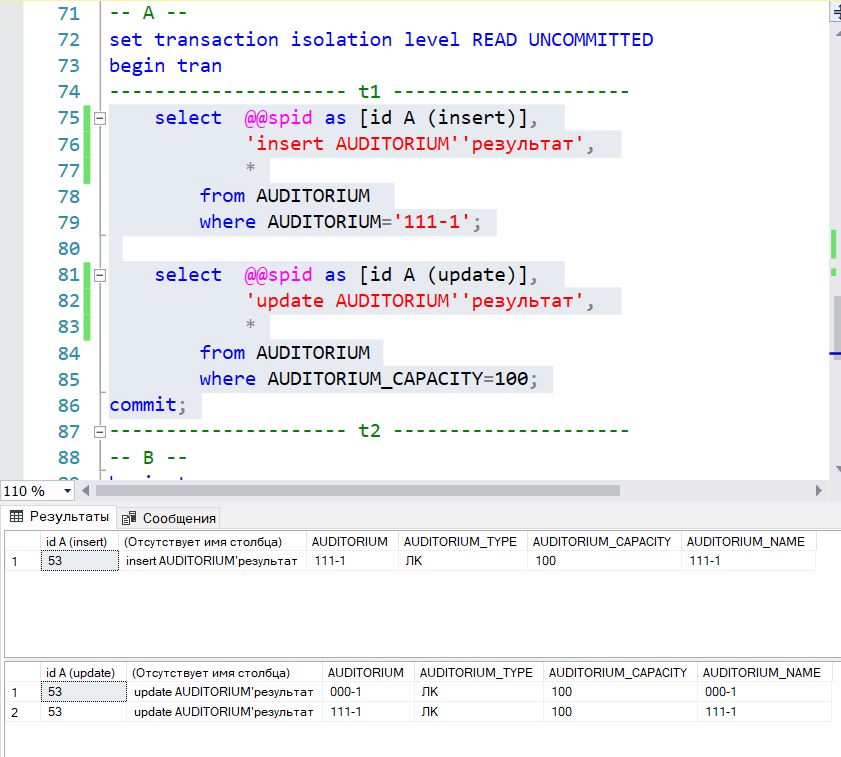
1. последовательно оба сценария до отметки t1;

******

******

1. последовательно оба сценария c отметки t1 до отметки t2;

Как видим, select-выборка вернула и те значения, которые ещё не были закоммичены в транзакции:

******

1. последовательно оба сценария c отметки t2 и до конца.

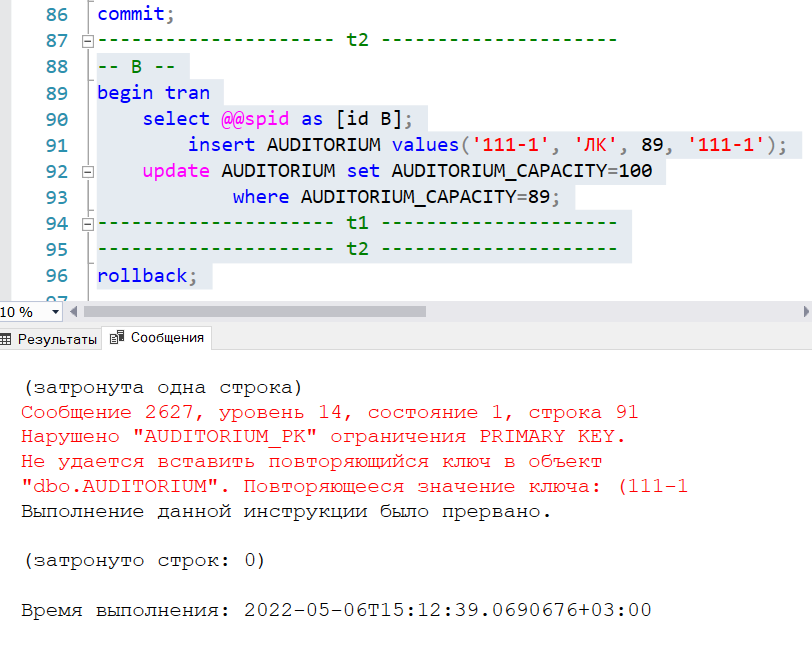
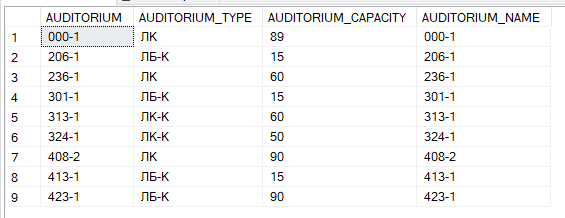
******

Таблица после транзакций:



5.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарии A и В представляют собой явные транзакции с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать, что уровень READ COMMITED не допускает неподтвержденного чтения, но при этом возможно неповторяющееся и фантомное чтение.

***→ Теория (методичка)***

Примердля базы данных ПРОДАЖИ:

**-- A ---**

**set transaction isolation level READ COMMITTED**

**begin transaction**

**select count(\*) from Заказы where Наименование\_товара = 'Стул';**

**-------------------------- t1 ------------------**

**-------------------------- t2 -----------------**

**select 'update Заказы' 'результат', count(\*)**

**from Заказы where Наименование\_товара = 'Стул';**

**commit;**

**--- B ---**

**begin transaction**

**-------------------------- t1 --------------------**

**update Заказы set Наименование\_товара = 'Стул'**

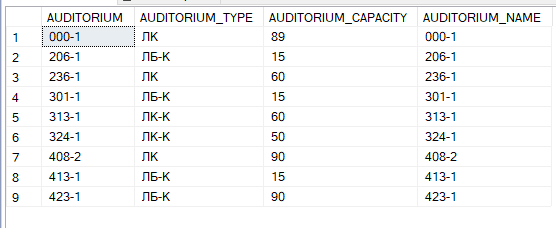
**where Наименование\_товара = 'Стол'**

**commit;**

**-------------------------- t2 --------------------**

***→ Выполнение заданий***

Таблица AUDITORIUM до транзакций:



Скрипт:

-- A --

set transaction isolation level READ COMMITTED

begin transaction

select count(\*) as [кол-во строк, где AUDITORIUM\_CAPACITY=100]

from AUDITORIUM

where AUDITORIUM\_CAPACITY=100;

--------------------- t1 ---------------------

--------------------- t2 ---------------------

select 'update AUDITORIUM''результат',

count(\*) as [кол-во строк, где AUDITORIUM\_CAPACITY=100]

from AUDITORIUM

where AUDITORIUM\_CAPACITY=100;

commit;

-- B --

begin transaction

--------------------- t1 ---------------------

update AUDITORIUM set AUDITORIUM\_CAPACITY=100

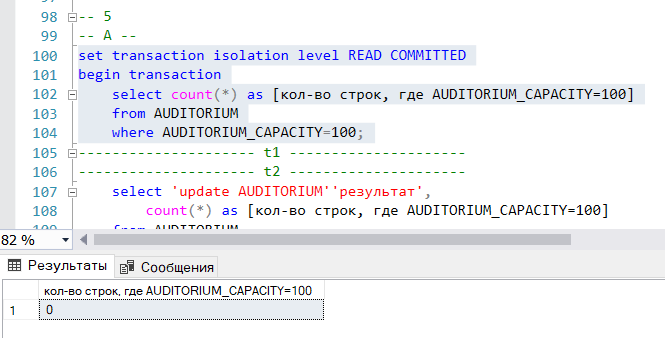
where AUDITORIUM\_CAPACITY=89;

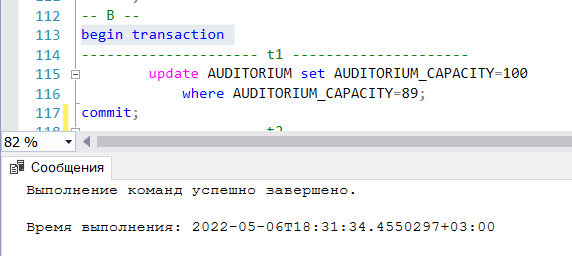
commit;

--------------------- t2 ---------------------

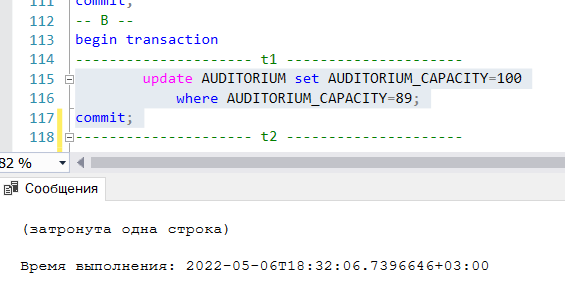
Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа (выделяем часть скрипта и нажимаем «Выполнить»):

1. последовательно оба сценария до отметки t1;





1. последовательно оба сценария c отметки t1 до отметки t2;

****

1. последовательно оба сценария c отметки t2 и до конца.

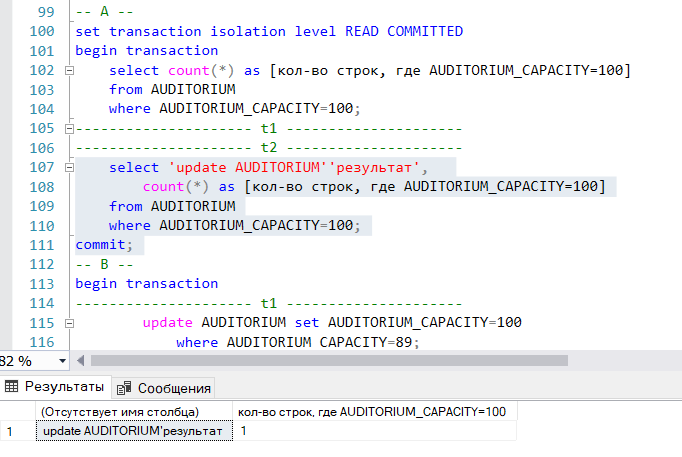
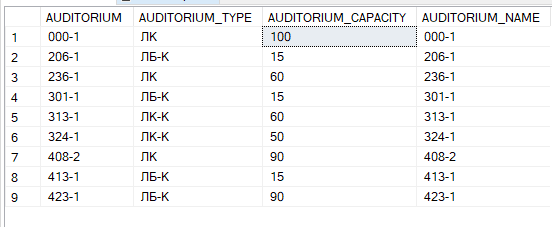


Таблица после транзакций:



6.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности REPEATABLE READ. Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать, что уровень REAPETABLE READ не допускает неподтвержденного чтения и неповторяющегося чтения, но при этом возможно фантомное чтение.

***→ Теория (методичка)***

Примердля базы данных ПРОДАЖИ:

**-- A ---**

**set transaction isolation level REPEATABLE READ**

**begin transaction**

**select Заказчик from Заказы where Наименование\_товара = 'Стул';**

**-------------------------- t1 ------------------**

**-------------------------- t2 -----------------**

**select**

**case**

**when Заказчик = 'Луч'**

**then 'insert Заказы'**

**else ' '**

**end**

**'результат', Заказчик from Заказы where Наименование\_товара = 'Стул';**

**commit;**

**--- B ---**

**begin transaction**

**-------------------------- t1 --------------------**

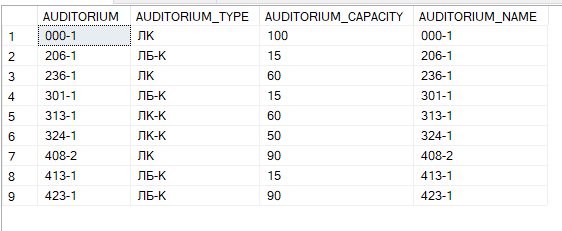
**insert Заказы values (12, 'Стул', 78, 10, '01.12.2014', 'Луч');**

**commit;**

**-------------------------- t2 --------------------**

***→ Выполнение заданий***

Таблица AUDITORIUM до транзакций:



Скрипт:

-- A --

set transaction isolation level REPEATABLE READ

begin tran

select AUDITORIUM from AUDITORIUM

where AUDITORIUM='666-6';

--------------------- t1 ---------------------

--------------------- t2 ---------------------

select

(case

when AUDITORIUM='666-6'

then 'inser AUDITORIUM'

else ''

end) + ' результат',

\*

from AUDITORIUM

where AUDITORIUM='666-6';

commit;

-- B --

set transaction isolation level READ COMMITTED

begin tran

--------------------- t1 ---------------------

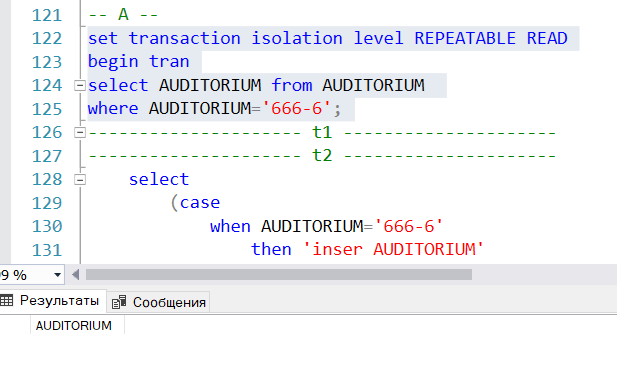
insert AUDITORIUM values('666-6', 'ЛК', 66, '666-6');

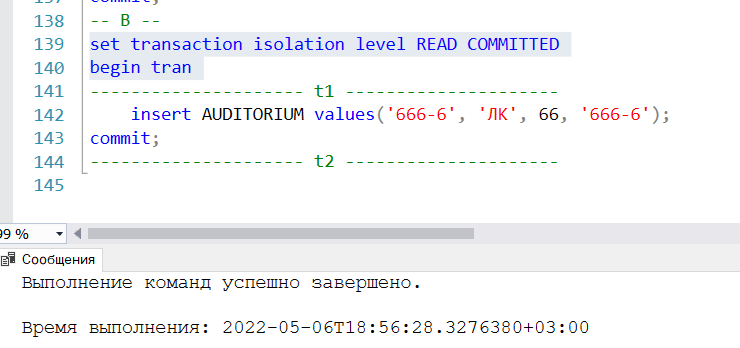
commit;

--------------------- t2 ---------------------

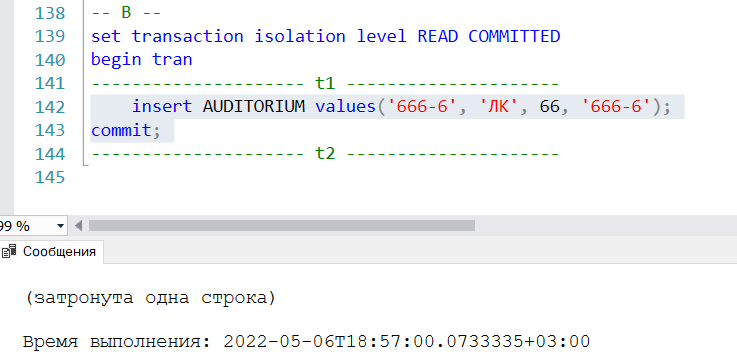
Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа (выделяем часть скрипта и нажимаем «Выполнить»):

1. последовательно оба сценария до отметки t1;

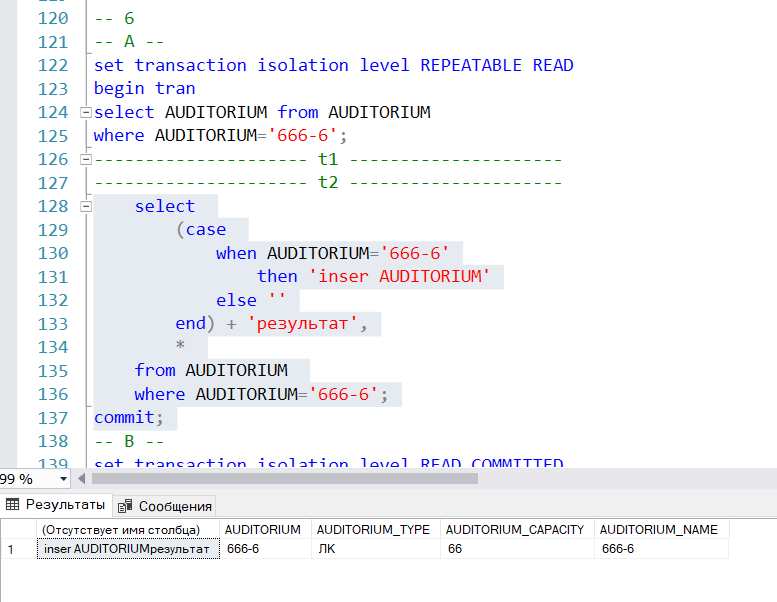




1. последовательно оба сценария c отметки t1 до отметки t2;

****

1. последовательно оба сценария c отметки t2 и до конца.



7.Разработать два сценария A и B на примере базы данных X\_UNIVER.

Сценарий A представляет собой явную транзакцию с уровнем изолированности SERIALIZABLE.

Сценарий B – явную транзакцию с уровнем изолированности READ COMMITED.

Сценарий A должен демонстрировать отсутствие фантомного, неподтвержденного и неповторяющегося чтения.

***→ Теория (методичка)***

Пример для базы данных ПРОДАЖИ:

**-- A ---**

**set transaction isolation level SERIALIZABLE**

**begin transaction**

**delete Заказы where Заказчик = 'Луч';**

**insert Заказы values (14, 'Стул', 78, 10, '01.12.2014', 'Луч');**

**update Заказы set Заказчик = 'Луч' where Наименование\_товара = 'Стул';**

**select Заказчик from Заказы where Наименование\_товара = 'Стул';**

**-------------------------- t1 -----------------**

**select Заказчик from Заказы where Наименование\_товара = 'Стул';**

**-------------------------- t2 ------------------**

**commit;**

**--- B ---**

**begin transaction**

**delete Заказы where Заказчик = 'Луч';**

**insert Заказы values (14, 'Стул', 78, 10, '01.12.2014', 'Луч');**

**update Заказы set Заказчик = 'Луч' where Наименование\_товара = 'Стул';**

**select Заказчик from Заказы where Наименование\_товара = 'Стул';**

**-------------------------- t1 --------------------**

**commit;**

**select Заказчик from Заказы where Наименование\_товара = 'Стул';**

**-------------------------- t2 --------------------**

***→ Выполнение заданий***

Скрипт:

-- A --

set tran isolation level SERIALIZABLE

begin tran

delete AUDITORIUM where AUDITORIUM='666-6';

insert AUDITORIUM values('171-7', 'ЛК', 17, '171-7');

update AUDITORIUM set AUDITORIUM\_TYPE='\*'

where AUDITORIUM='171-7';

select \*

from AUDITORIUM

where AUDITORIUM='171-7';

--------------------- t1 ---------------------

select \*

from AUDITORIUM

where AUDITORIUM='171-7';

--------------------- t2 ---------------------

commit;

-- B --

set tran isolation level READ COMMITTED

begin tran

delete AUDITORIUM where AUDITORIUM='666-6';

insert AUDITORIUM values('171-7', 'ЛК', 17, '171-7');

update AUDITORIUM set AUDITORIUM\_TYPE='\*'

where AUDITORIUM='171-7';

select \*

from AUDITORIUM

where AUDITORIUM='171-7';

--------------------- t1 ---------------------

commit;

select \*

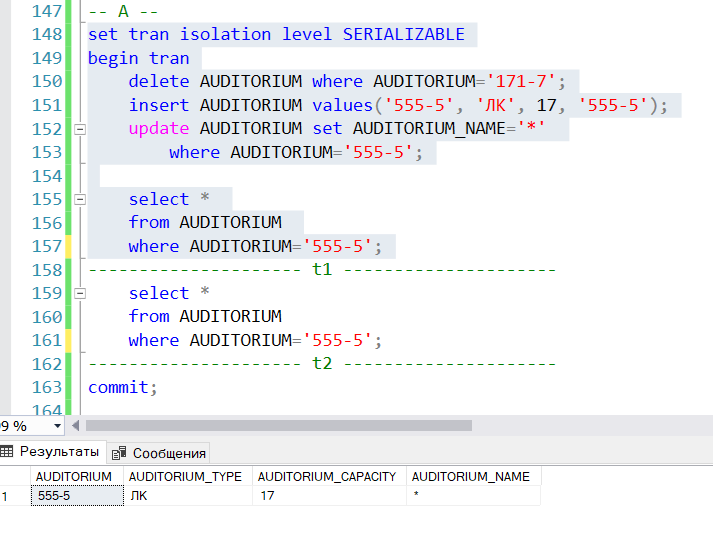
from AUDITORIUM

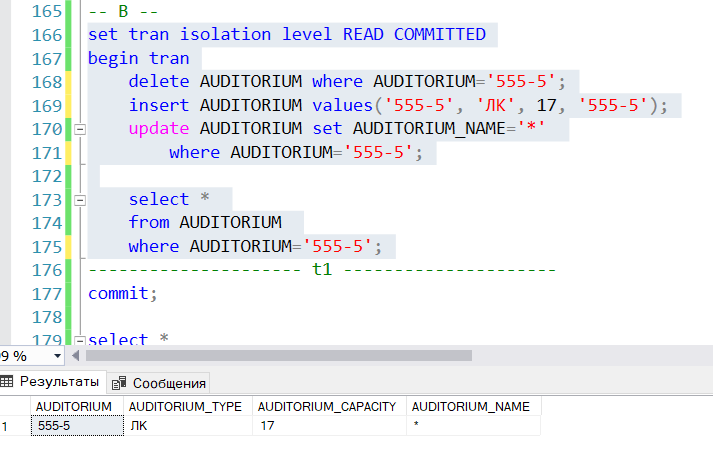
where AUDITORIUM='171-7';

--------------------- t2 ---------------------

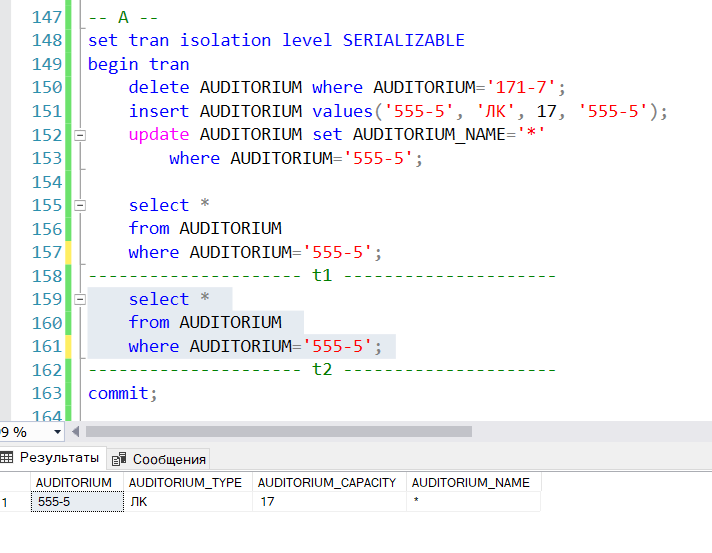
Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа (выделяем часть скрипта и нажимаем «Выполнить»):

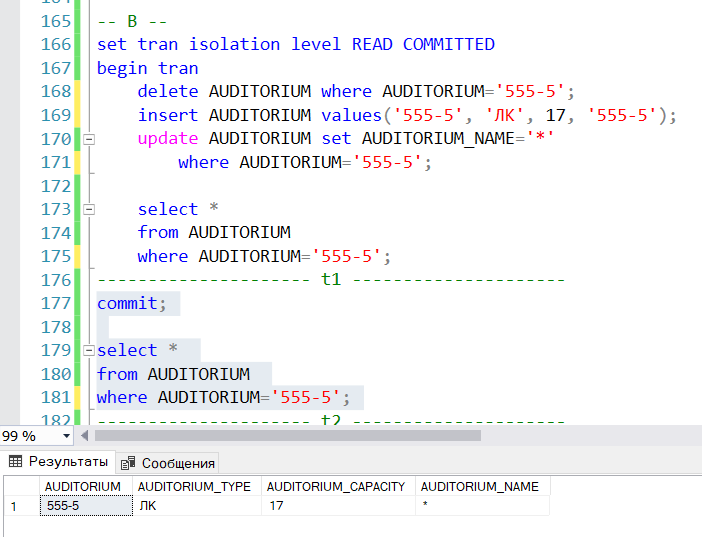
1. последовательно оба сценария до отметки t1;



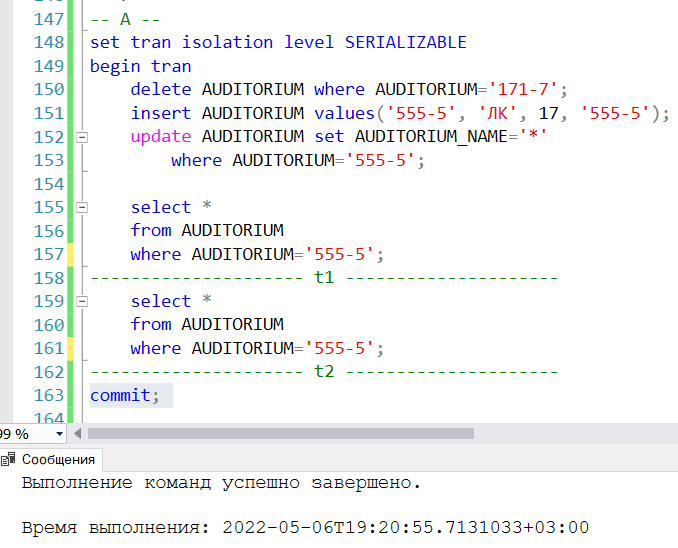


1. последовательно оба сценария c отметки t1 до отметки t2;

****

****

1. последовательно оба сценария c отметки t2 и до конца.



8.Разработать сценарий, демонстрирующий свойства *вложенных* транзакций, на примере базы данных X\_UNIVER.

***→ Теория (методичка)***

Транзакция, выполняющаяся в рамках другой транзакции, называется *вложенной*.

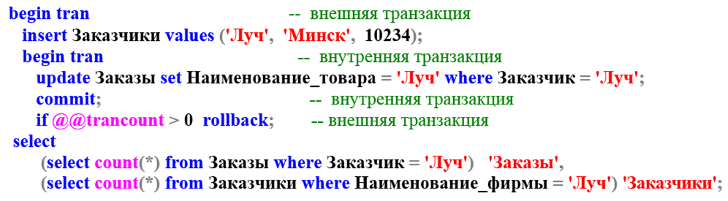
При работе с вложенными транзакциями нужно учитывать следующее:

− оператор COMMIT вложенной транзакции действует только на внутренние операции вложенной транзакции;

− оператор ROLLBACK внешней транзакции отменяет зафиксированные операции внутренней транзакции;

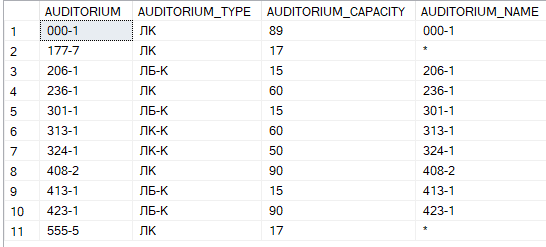
− оператор ROLLBACK вложенной транзакции действует на операции внешней и внутренней транзакции, а также завершает обе транзакции;

− уровень вложенности транзакции можно определить с помощью системной функции @@TRANCOUT.



***→ Выполнение заданий***

Таблица до транзакции:



Из-за отката во внешней транзакции, внутренняя тоже будет отменена, несмотря на то, что была зафиксирована:

set nocount on;

begin tran -- внешняя транзакция

insert AUDITORIUM values('!', 'ЛК',17, '!');

begin tran -- внутренняя транзакция

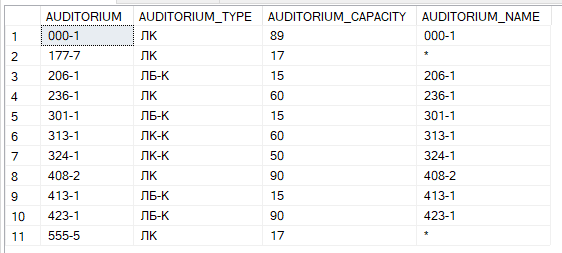
update AUDITORIUM set AUDITORIUM\_NAME='gtl-wereeeeeewqq'

where AUDITORIUM\_NAME='\*';

commit; -- внутренняя транзакция

rollback; -- внешняя транзакция

select \* from AUDITORIUM;



9. Разработать скрипты с использованием транзакций для базы данных **X\_MyBASE**.