Лабораторная работа 9:

Управление Транзакциями и Блокировками

a. Определение транзакции

В этом задании вы на примере простой транзакции посмотрите, как работают инструкции **COMMIT TRAN** и **ROLLBACK TRAN**.

1. Запустите **Query Editor** (Редактор запросов) и в окне запроса введите следующий код T-SQL:

```
USE ApressFinancial
GO
SELECT 'Before',ShareId,ShareDesc,CurrentPrice
    FROM ShareDetails.Shares
    WHERE ShareId = 3
BEGIN TRAN ShareUpd
    UPDATE ShareDetails.Shares
    SET CurrentPrice = CurrentPrice * 1.1
    WHERE ShareId = 3
COMMIT TRAN
SELECT 'After',ShareId,ShareDesc,CurrentPrice
    FROM ShareDetails.Shares
    WHERE ShareId = 3
```

Примечание. *Обратите внимание, что инструкция **COMMIT TRAN** не использует имя, связанное с **BEGIN TRAN.***

2. Выполните код. На рис 9.1 приведены результаты, которые отображают таблицу до и после транзакции:

	(No column name)	ShareId	ShareDesc	CurrentPrice
1	Before	3	NetRadio Inc	29.79000

	(No column name)	ShareId	ShareDesc	CurrentPrice
1	After	3	NetRadio Inc	32.76900

Рис. 9.1 Обновление с меткой транзакции и инструкцией COMMIT TRAN.

3. Теперь поработаем с инструкцией **ROLLBACK TRAN**. Введите следующий код и выполните его *весь одновременно*:

```
SELECT 'Before',ShareId,ShareDesc,CurrentPrice
    FROM ShareDetails.Shares
    WHERE ShareId <= 3
BEGIN TRAN ShareUpd
    UPDATE ShareDetails.Shares
          SET CurrentPrice = CurrentPrice * 2.0
          WHERE ShareId <= 3
    SELECT 'Within the transaction',ShareId,ShareDesc,CurrentPrice
          FROM ShareDetails.Shares WHERE ShareId <= 3
ROLLBACK TRAN
    SELECT 'After',ShareId,ShareDesc,CurrentPrice
    FROM ShareDetails.Shares
    WHERE ShareId <= 3
```

4. Результаты на рис 9.2 в точности показывают, что произошло. Изучите их внимательно.

	(No column name)	ShareId	ShareDesc	CurrentPrice
1	Before	1	ACME'S HOMEBAKE COOKIES INC	2.34125
2	Before	2	FAT-BELLY.COM	45.20000
3	Before	3	NetRadio Inc	32.76900

	(No column name)	ShareId	ShareDesc	CurrentPrice
1	Within the transaction	1	ACME'S HOMEBAKE COOKIES INC	4.68250
2	Within the transaction	2	FAT-BELLY.COM	90.40000
3	Within the transaction	3	NetRadio Inc	65.53800

	(No column name)	ShareId	ShareDesc	CurrentPrice
1	After	1	ACME'S HOMEBAKE COOKIES INC	2.34125
2	After	2	FAT-BELLY.COM	45.20000
3	After	3	NetRadio Inc	32.76900

Рис. 9.2. Обновление с меткой транзакции и инструкцией ROLLBACK TRAN.

5. Теперь введите и выполните следующий код, чтобы понять, как работают вложенные транзакции:

```
BEGIN TRAN ShareUpd
    SELECT '1st TranCount',@@TRANCOUNT
BEGIN TRAN ShareUpd2
    SELECT '2nd TranCount',@@TRANCOUNT
COMMIT TRAN ShareUpd2
    SELECT '3rd TranCount',@@TRANCOUNT
COMMIT TRAN -- It is at this point that data modifications will be committed
    SELECT 'Last TranCount',@@TRANCOUNT
```

Примечание. *Глобальная переменная* ***@@TRANCOUNT*** *при выполнении каждой инструкции* ***BEGIN TRAN*** *увеличивается на 1, а при выполнении каждой инструкции* ***COMMIT TRAN*** *уменьшается на 1. При выполнении* ***ROLLBACK TRAN*** *производится откат транзакции, а значение переменной* ***@@TRANCOUNT*** *устанавливается в 0.*

b. Поиск и обнаружение блокирования

Для изоляции транзакций SQL Server применяет блокировки. В этом задании вы научитесь получать сведения о блокировках.

1. Выполните код из файла **SQLQuery_CREATE_DATABASE_exmp_block.sql**. У вас появится БД **[test_block]**.
2. Запустите **Query Editor** (Редактор запросов) и откройте три отдельных окна запросов в SSMS. Далее эти окна будут называться **Connection 1** (подключение), **Connection 2** и **Connection 3.** Все окна подключите к созданной БД. Для этого во всех окнах выполните следующий код:

```
USE [test_block];
```

3. В окне **Connection 1** выполните следующий код для обновления строки в таблице **dbo.[t1]**, которое заключается в увеличении текущей цены на 1.00 для товара 2:

```
-- Connection 1
BEGIN TRAN;
  UPDATE dbo.[t1]
    SET [Price] = [Price] + 1.00
  WHERE [ID] = 2;
SELECT [ID],[Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *Для обновления строки сеанс должен был установить монопольную блокировку, и если обновление прошло успешно, значит, блокировка была установлена. Монопольные блокировки сохраняются до конца транзакции, а поскольку транзакция остается открытой, блокировка продолжает удерживаться***.**

4. В окне **Connection 2** выполните следующий код, чтобы попытаться запросить ту же самую строку:

```
-- Connection 2
SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *Этот сеанс вынужден ждать, т.к. ему нужна совместная блокировка для чтения данных, но совместная блокировка несовместима с монопольной блокировкой***.**

5. Чтобы прояснить подобную ситуацию в окне **Connection 3** выполните запросы к объектам динамического управления для обнаружения ситуации блокирования:

```
-- Connection 3
-- Lock info
SELECT -- use * to explore
  request_session_id            AS spid,
  resource_type                 AS restype,
  resource_database_id          AS dbid,
  DB_NAME(resource_database_id) AS dbname,
  resource_description          AS res,
  resource_associated_entity_id AS resid,
  request_mode                  AS mode,
  request_status                AS status
FROM sys.dm_tran_locks;
```

6. Должен быть получен следующий результат:

spid	restype	dbid	dbname	res	resid	mode	status
52	DATABASE	9	test_block		0	S	GRANT
55	DATABASE	9	test_block		0	S	GRANT
54	DATABASE	9	test_block		0	S	GRANT
54	PAGE	9	test_block	1:41	72057594038779904	IS	GRANT
52	PAGE	9	test_block	1:41	72057594038779904	IX	GRANT
54	PAGE	9	test_block	1:90	72057594038845440	IS	GRANT
54	OBJECT	9	test_block		85575343	IS	GRANT
52	OBJECT	9	test_block		85575343	IX	GRANT
52	RID	9	test_block	1:41:1	72057594038779904	X	GRANT
54	RID	9	test_block	1:41:1	72057594038779904	S	WAIT
54	KEY	9	test_block	(020068e8b274)	72057594038845440	S	GRANT

Примечание. *Каждый сеанс обозначен уникальным ID серверного процесса (spid), а определить SPID вашего процессам можно с помощью* ***@@SPID*** *(в SSMS это значение можно найти внизу экрана в строке состояния в круглых скобках справа от регистрационного имени). Обратите внимание на то, что оба процесса (52 и 54) блокируют строку с одинаковыми значениями атрибутов* ***res*** *и* ***resid****.*

7. Для получения сведений о подключениях, связанных с процессами, входящими в цепочку блокировок, в окне **Connection 3** запросите представление **sys.dm_exec_connections**:

```
-- Connection info
SELECT -- use * to explore
  session_id AS spid,
  connect_time,
  last_read,
  last_write,
  most_recent_sql_handle
FROM sys.dm_exec_connections
WHERE session_id IN(52, 54);
```

Примечание. *Правильно укажите процессы в предложении WHERE.*

8. Можно получить результат, отображающий последний пакет или фрагмент программного кода, запускаемый каждым подключением, входящим в цепочку блокировок:

```
-- SQL text
SELECT session_id, text
FROM sys.dm_exec_connections
  CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) AS ST
WHERE session_id IN(52, 54);
```

9. Должен получиться следующий результат:

```
session_id  text
----------- ------------------------------------------------------------------
52          BEGIN TRAN;
  UPDATE dbo.[t1]
    SET [Price] = [Price] + 1.00
  WHERE [ID] = 2;
54          (@1 tinyint)SELECT [ID],[Price] FROM [dbo].[t1] WHERE [ID]=@1

(2 row(s) affected)
```

10. Еще одно DMV-представление отображает активные процессы, включая заблокированные:

```
-- Blocking
SELECT -- use * to explore
  session_id AS spid,
  blocking_session_id,
  command,
  sql_handle,
  database_id,
  wait_type,
  wait_time,
  wait_resource
FROM sys.dm_exec_requests
WHERE blocking_session_id > 0;
```

Примечание. *У заблокированных процессов значение атрибута* ***blocking_session_id > 0.***

11. В окне **Connection 2** остановите запрос, выполнив команду из меню **Query** (Запрос) → **Cancel Executing Query** (Отменить выполняющийся запрос).
После отмены заблокированного запроса в окне **Connection 2** задайте время ожидания блокировки 5 сек и снова выполните запрос:

```
-- Connection 2
-- Stop, then set the LOCK_TIMEOUT, then retry
```

```
SET LOCK_TIMEOUT 5000;

SELECT [ID], [Price]
FROM dbo.[t1]
WHERE [ID] = 2;
```

12. Через пять секунд появится сообщение:
```
Msg 1222, Level 16, State 45, Line 3
Lock request time out period exceeded.
```
13. Удалите значение время ожидания блокировки, вернув его к значению по умолчанию (неопределенному) и повторно запустите запрос:
```
-- Connection 2
-- Remove timeout
SET LOCK_TIMEOUT -1;

SELECT [ID], [Price]
FROM dbo.[t1]
WHERE [ID] = 2;
```
А в окне **Connection 3** завершите транзакцию, выполнив код:
```
-- Connection 3
KILL 52;
```
Эта инструкция вызовет откат транзакции в сеансе **Connection 1**, т.е. изменение цены товара 2 отменяется и монопольная блокировка снимается.

14. Перейдите в окно **Connection 2:** будут получены результаты запроса, данные будут такими же, как до изменения цены.

c. Уровни изоляции

Уровни изоляции определяют поведение параллельно работающих пользователей, читающих и записывающих данные. Читающий процесс – это любая инструкция, извлекающая данные и применяющая по умолчанию совместную блокировку. Пишущая инструкция - это любая инструкция, модифицирующая таблицу и запрашивающая монопольную блокировку. В этом задании вы научитесь задавать уровни изоляции транзакций и посмотрите, как они работают.

Уровень изоляции *READ UNCOMMITTED*

1. Запустите **Query Editor** (Редактор запросов) и откройте два отдельных окна запросов в SSMS. Далее эти окна будут называться **Connection 1** и **Connection 2.** Все окна подключите к созданной БД. Для этого во всех окнах выполните следующий код:
```
USE [test_block];
```
2. В окне **Connection 1** запустите транзакцию и выполните код, аналогичный коду предыдущего задания для обновления строки в таблице **dbo.[t1]**, которое заключается в увеличении текущей цены на 1.00 для товара 2:
```
-- Connection 1
BEGIN TRAN;
  UPDATE dbo.[t1]
    SET [Price] = [Price] + 1.00
  WHERE [ID] = 2;
SELECT [ID],[Price] FROM dbo.[t1] WHERE [ID] = 2;
```
Примечание. *Вы видите, что обновление прошло успешно, но транзакция остается открытой. Это означает монопольную блокировку строки в подключении* ***Connection 1.***
3. В окне **Connection 2** установите уровень изоляции *READ UNCOMMITTED,* для чего введите и выполните следующий код:
```
-- Connection 2
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

BEGIN TRAN;
SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```
Примечание. *Поскольку сеансу не требовалась совместная блокировка, конфликт не возник, запрос выполнился и вернул состояние строки после обновления, несмотря на то, что изменение не было зафиксировано.*

4. Теперь в окне **Connection 1** выполните откат транзакции:

```
-- Connection 1
ROLLBACK TRAN;
SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *Обновление данных отменено.* ***Это пример «грязного» чтения.***

Уровень изоляции *READ COMMITTED*

1. Запустите **Query Editor** (Редактор запросов) и откройте два отдельных окна запросов в SSMS. Далее эти окна будут называться **Connection 1** и **Connection 2.** Все окна подключите к созданной БД. Для этого во всех окнах выполните следующий код:

```
USE [test_block];
```

2. В окне **Connection 1** запустите транзакцию:

```
-- Connection 1
BEGIN TRAN;
  UPDATE dbo.[t1]
    SET [Price] = [Price] + 1.00
  WHERE [ID] = 2;
SELECT [ID],[Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *Обновление прошло успешно, подключение* ***Connection 1*** *монопольно блокирует строку 2 в таблице.*

3. В окне **Connection 2** установите уровень изоляции *READ COMMITTED* и выполните *SELECT*:

```
-- Connection 2
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *В настоящий момент инструкция SELECT блокируется. Не забывайте о том, что данный уровень изоляции устанавливается в SQL Server по умолчанию.*

4. Теперь в окне **Connection 1** выполните откат транзакции:

```
-- Connection 1
ROLLBACK TRAN;
```

Примечание. *На уровне изоляции* ***READ COMMITTED*** *«грязного» чтения вы не получили, но может быть* ***несогласованное (неповторяющееся) чтение****.*

Уровень изоляции *REPEATABLE READ*

1. Запустите **Query Editor** (Редактор запросов) и откройте два отдельных окна запросов в SSMS. Далее эти окна будут называться **Connection 1** и **Connection 2.** Все окна подключите к БД:

```
USE [test_block];
```

2. В окне **Connection 1** установите уровень изоляции REPEATABLE READ и запустите транзакцию:

```
-- Connection 1
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRAN;
SELECT [ID],[Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *Код вернет текущую цену товара, но подключение* ***Connection 1*** *все еще удерживает совместную блокировку строки 2 в таблице.*

3. В окне **Connection 2** выполните следующий код, чтобы попытаться изменить цену товара 2:

```
-- Connection 2
UPDATE dbo.[t1]
  SET [Price] = [Price] + 1.00
WHERE [ID] = 2;
```

Примечание. *Попытка будет блокирована. Но если бы вы читающий процесс выполняли на уровне изоляции* ***READ UNCOMMITTED*** *или* ***READ COMMITTED****, то попытка модификации строки завершилась бы успешно, а при повторном считывании строки 2 транзакцией (в подключении* ***Connection 1****) ей вернулись бы модифицированные данные после обновления.*

4. В окне **Connection 1** еще раз считайте данные (они не изменились!) и завершите транзакцию:

```
-- Connection 1
SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
COMMIT TRAN;
```

5. Зайдите в окно **Connection 2**. Вы увидите сообщение «**(1 row(s) affected)**»
   Проверьте цену товара 2 – она должна измениться.

Примечание. *На уровне изоляции* ***REPEATABLE READ*** *читающие процессы сохраняют совместные блокировки до конца транзакции. Это гарантирует повторяемость результатов при считывании строк. Однако транзакция блокирует ресурсы (например, строки), которые запрос нашел в процессе выполнения в первый раз, а не строки, которых там не было во время выполнения запроса.* ***Этот уровень изоляции не предотвращает «фантомного» чтения.***

Уровень изоляции *SERIALIZABLE*

1. Запустите **Query Editor** (Редактор запросов) и откройте два отдельных окна запросов в SSMS. Далее эти окна будут называться **Connection 1** и **Connection 2.** Все окна подключите к БД:
```
USE [test_block];
```
2. В окне **Connection 1** установите уровень изоляции SERIALIZABLE и запустите транзакцию:
```
-- Connection 1
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRAN;
  SELECT [ID], [Name], [Price], [Discount]
  FROM dbo.[t1]
  WHERE [Discount] < 0.20;
```
Примечание. *Вы должны получить 6 строк с информацией о товарах со скидкой до 20%.*
3. В окне **Connection 2** попытайтесь добавить новый товар со скидкой 5%:
```
-- Connection 2
INSERT INTO dbo.[t1]
    ( [Name], [Number], [Price], [Discount])
  VALUES('Product ABCDE', 1, 1.00, 0.050);
```
Примечание. *На всех уровнях изоляции более низких, чем SERIALIZABLE, такая попытка была бы успешной. Но на уровне SERIALIZABLE она блокируется.*
4. Вернитесь в окно **Connection 1**, еще раз извлеките данные и зафиксируйте транзакцию:
```
-- Connection 1
  SELECT [ID], [Name], [Price], [Discount]
  FROM dbo.[t1]
  WHERE [Discount] < 0.20;
COMMIT TRAN;
```
Примечание. *Вы должны получить тот же результат, без строк-«фантомов».*
5. Теперь, когда транзакция читающего процесса зафиксирована, и совместная блокировка с диапазона строк снята, модифицирующий процесс в окне **Connection 2** получит монопольную блокировку, которую он ожидал, и добавит строку.
6. Для возврата уровня изоляции к значению, принятому по умолчанию, во всех открытых подключениях выполните следующий код:
```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

Уровень изоляции *SNAPSHOT*

1. Запустите **Query Editor** (Редактор запросов) и откройте два отдельных окна запросов в SSMS: **Connection 1** и **Connection 2.** Для разрешения работы на уровне изоляции SNAPSHOT сначала необходимо установить параметр на уровне БД. В любом открытом окне запроса выполните код:
```
ALTER DATABASE [test_block] SET ALLOW_SNAPSHOT_ISOLATION ON;
```
2. В окне **Connection 1** запустите транзакцию, обновляющую цену товара 2:
```
-- Connection 1
BEGIN TRAN;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
UPDATE dbo.[t1]
  SET [Price] = [Price] + 1.00
WHERE [ID] = 2;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```
Примечание. *Вы должны увидеть, что цена товара обновилась.*
3. В окне **Connection 2** выполните следующий код:
```
-- Connection 2
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
BEGIN TRAN;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Примечание. *Имейте ввиду: несмотря на то, что транзакция в окне* ***Connection 1*** *выполнялась на уровне изоляции READ COMMITTED, принятом по умолчанию, SQL Server должен был перед обновлением скопировать версию строки со старой ценой в БД tempdb. Поэтому в окне* ***Connection 2*** *при выполнении транзакции на уровне изоляции SNAPSHOT была получена последняя зафиксированная версия строки (на уровне изоляции SERIALIZABLE запрос был бы заблокирован).*

4. Вернитесь в окно **Connection 1** и зафиксируйте транзакцию, модифицирующую строку:

```
-- Connection 1
COMMIT TRAN;
```

5. Если вы еще раз в окне **Connection 2** прочитаете данные, то опять получите версию строки, которая имелась в момент запуска транзакции. Считайте данные и зафиксируйте транзакцию:

```
-- Connection 2
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
COMMIT TRAN;
```

6. В окне **Connection 2** выполните следующий код:

```
-- Connection 2
BEGIN TRAN;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
COMMIT TRAN;
```

Примечание. *Вы увидите обновленную цену товара. Объясните, почему?*

Уровень изоляции SNAPSHOT предотвращает конфликты обновления, но в отличие от уровней изоляции REPEATABLE READ и SERIALIZABLE, делающих это генерацией взаимоблокировки, он аварийно завершает транзакцию, указывая, что обнаружен конфликт обновления.

7. В окне **Connection 1** выполните следующий код:

```
-- Connection 1, Step 1
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
BEGIN TRAN;
    SELECT [ID],[Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Вы получите результат. Предполагая, что еще выполнены некие вычисления, основанные на чтении, в том же окне **Connection 1** выполните следующий код для изменения значения цены и зафиксируйте транзакцию:

```
-- Connection 1, Step 2
  UPDATE dbo.[t1]
    SET [Price] = 20.00
  WHERE [ID] = 2;
COMMIT TRAN;
```

Никакая другая транзакция между чтением, вычислением и записью не модифицировала строку, поэтому конфликта обновления не было.

8. *Пример сценария с конфликтом обновления*. В окне **Connection 1** выполните следующий код:

```
-- Connection 1, Step 1
BEGIN TRAN;
    SELECT [ID],[Price] FROM dbo.[t1] WHERE [ID] = 2;
```

Вы получите результат, показывающий, что цена товара 20,00.

Следующий код выполните в другом окне **Connection 2** (чтобы изменить цену):

```
-- Connection 2, Step 1
  UPDATE dbo.[t1]
    SET [Price] = 25.00
  WHERE [ID] = 2;
```

В окне **Connection 1** выполните следующий код, опять изменяющий значение цены:

```
-- Connection 1, Step 2
  UPDATE dbo.[t1]
    SET [Price] = 20.00
  WHERE [ID] = 2;
```

SQL Server обнаружил, что на этот раз другая транзакция модифицировала данные между чтением и записью, поэтому он аварийно завершит транзакцию с такой ошибкой:

```
Msg 3960, Level 16, State 2, Line 1
Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation
to access table 'dbo.t1' directly or indirectly in database 'test_block' to update, delete, or
insert the row that has been modified or deleted by another transaction. Retry the transaction
or change the isolation level for the update/delete statement.
```

Закройте все подключения.

Уровень изоляции *READ COMMITTED SNAPSHOT*

Уровень изоляции READ COMMITTED SNAPSHOT также основан на версиях строк. Он отличается от уровня изоляции SNAPSHOT тем, что вместо последней зафиксированной версии строки, имевшейся при старте транзакции, читающий процесс получает последнюю зафиксированную версию строки, имевшуюся в момент старта инструкции. Кроме того он не обнаруживает конфликты обновления.

1. Запустите **Query Editor** (Редактор запросов) и откройте два отдельных окна запросов в SSMS: **Connection 1** и **Connection 2.** Для разрешения применения в базе данных уровня изоляции READ COMMITTED SNAPSHOT необходимо установить флаг базы данных. В любом открытом окне запроса выполните код:

```
ALTER DATABASE [test_block] SET READ_COMMITTED_SNAPSHOT ON;
```

   Примечание. *Это подключение должно быть единственным открытым подключением к БД test_block! Интересная особенность этого флага базы данных заключается в том, что в отличие от уровня изоляции SNAPSHOT этот флаг на самом деле изменяет значение уровня изоляции по умолчанию READ COMMITTED на уровень изоляции READ COMMITTED SNAPSHOT.*

2. В окне **Connection 1** запустите транзакцию, обновляющую цену товара 2:

```
-- Connection 1
BEGIN TRAN;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
UPDATE dbo.[t1]
  SET [Price] = [Price] + 1.00
WHERE [ID] = 2;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

   Примечание. *Вы получите результат, показывающий, что цена товара обновилась.*

3. В окне **Connection 2** запустите транзакцию и прочтите строку с товаром 2:

```
-- Connection 2
BEGIN TRAN;
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

   Вы получите последнюю зафиксированную версию строки, имевшуюся на тот момент, когда инструкция стартовала, т.е. еще неизмененную цену.

4. В окне **Connection 1** зафиксируйте транзакцию:

```
-- Connection 1
COMMIT TRAN;
```

5. Теперь в окне **Connection 2** снова прочтите строку с товаром 2:

```
-- Connection 2
    SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;
```

   Вы получите последнюю зафиксированную версию строки, т.е. уже изменившуюся цену.

   Примечание. *Эта ситуация именуется «**неповторяемое чтение»!***

   Зафиксируйте транзакцию:

```
COMMIT TRAN;
```

Закройте все подключения. В окне *нового* подключения выполните следующий программный код, запрещающий применение в БД уровней изоляции, основанных на моментальных снимках:

```
ALTER DATABASE [test_block] SET ALLOW_SNAPSHOT_ISOLATION OFF;

ALTER DATABASE [test_block] SET READ_COMMITTED_SNAPSHOT OFF;
```