

БАЗЫ ДАННЫХ

Лекция 14 Транзакции

Транзакция

- Одна или несколько команд SQL, которые либо успешно выполняются как единое целое, либо отменяются как единое целое

Транзакция

- Логическая единица работы, обеспечивающая переход базы данных из одного согласованного состояния в другое согласованное состояние

Транзакции

- Неявные
- Явные

Неявные транзакции

- Неявная транзакция — задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции

Явные транзакции

- Явная транзакция — группа инструкций, начало и конец которой обозначаются инструкциями:
 - BEGIN TRANSACTION
 - COMMIT
 - ROLLBACK

ACID

- Atomicity - Атомарность
- Consistency - Согласованность
- Isolation - Изолированность
- Durability - Долговечность

Atomicity - Атомарность

- Выполняются или все изменения данных в транзакции или ни одна

Consistency - Согласованность

- Выполняемые транзакцией трансформации данных переводят базу данных из одного согласованного состояния в другое

Isolation - Изолированность

- Все параллельные транзакции отделяются друг от друга.
- Активная транзакция не может видеть модификации данных в параллельной или незавершенной транзакции

Durability - Долговечность

- Транзакцию после фиксации нельзя отменить, кроме как другой транзакцией

Инструкции

- BEGIN TRANSACTION
- BEGIN DISTRIBUTED TRANSACTION
- COMMIT [WORK]
- ROLLBACK [WORK]
- SAVE TRANSACTION
- SET IMPLICIT_TRANSACTIONS

BEGIN TRANSACTION

```
 BEGIN TRANSACTION;

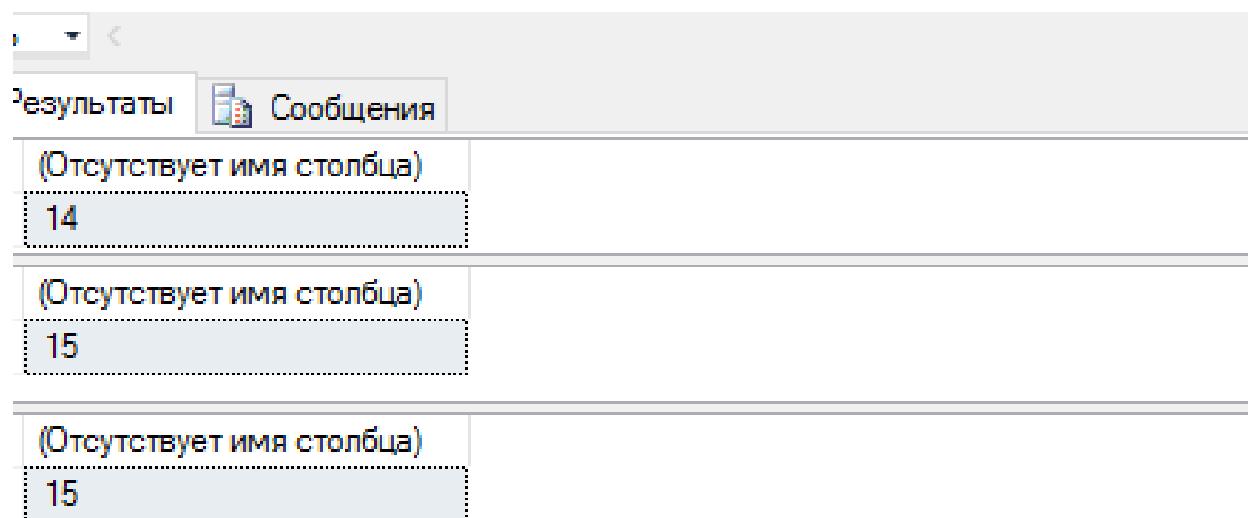
SELECT COUNT(*) FROM AUDITORIUM;

INSERT INTO AUDITORIUM VALUES ('123-1', 'ЛК', 60, '123-1');

SELECT COUNT(*) FROM AUDITORIUM;

COMMIT;

SELECT COUNT(*) FROM AUDITORIUM;
```



The screenshot shows the results of the SQL query execution. The interface includes tabs for 'Результаты' (Results) and 'Сообщения' (Messages). The Results tab displays three rows of data:

1	2	3
(Отсутствует имя столбца)		
14		
(Отсутствует имя столбца)		
15		
(Отсутствует имя столбца)		
15		

ROLLBACK

```
BEGIN TRANSACTION;

SELECT COUNT(*) FROM AUDITORIUM;

INSERT INTO AUDITORIUM VALUES ('128-1', 'ЛК-К', 60, '128-1');

SELECT COUNT(*) FROM AUDITORIUM;

ROLLBACK;

SELECT COUNT(*) FROM AUDITORIUM;
```

The screenshot shows a database query results window with three rows of data. The first row has a header '(Отсутствует имя столбца)' and contains the value '15'. The second row has a header '(Отсутствует имя столбца)' and contains the value '16'. The third row has a header '(Отсутствует имя столбца)' and contains the value '15'.

(Отсутствует имя столбца)
15
(Отсутствует имя столбца)
16
(Отсутствует имя столбца)
15

SAVE TRANSACTION

```
BEGIN TRANSACTION;

INSERT INTO AUDITORIUM VALUES ('127-1', 'ЛК', 60, '127-1');
SAVE TRANSACTION A;

INSERT INTO AUDITORIUM VALUES ('128-1', 'ЛК-К', 60, '128-1');
SAVE TRANSACTION B;

INSERT INTO AUDITORIUM VALUES ('129-1', 'ЛК', 60, '129-1');
ROLLBACK TRANSACTION B;

INSERT INTO AUDITORIUM VALUES ('130-1', 'ЛК-К', 60, '130-1');
ROLLBACK TRANSACTION A;

COMMIT TRANSACTION;
```

|

```
SELECT * FROM AUDITORIUM
WHERE AUDITORIUM IN('127-1', '128-1', '129-1', '130-1');
```

% <

Результаты Сообщения

AUDITORIUM	AUDITORIUM_TYPE	AUDITORIUM_CAPACITY	AUDITORIUM_NAME
127-1	ЛК	60	127-1

SAVE TRANSACTION

- Точка сохранения определяет точку в транзакции, такую что все последующие изменения данных могут быть отменены без отмены всей транзакции
- **SAVE TRANSACTION** создает метку для последующей инструкции **ROLLBACK**, имеющей такую же метку, как и данная инструкция **SAVE TRANSACTION**

BEGIN DISTRIBUTED TRANSACTION

- Запускается распределенная транзакция
- Управляется Microsoft Distributed Transaction Coordinator
- Распределенная транзакция — это транзакция, которая используется на нескольких базах данных и на нескольких серверах
- Координатор - сервер, запустивший инструкцию BEGIN DISTRIBUTED TRANSACTION

SET IMPLICIT_TRANSACTIONS ON

- Режим неявных транзакций
- Если транзакцию явно не зафиксировать, то все изменения, выполненные в ней, откатываются при отключении пользователя
- Любая из следующих инструкций запускает транзакцию:
- CREATE (ALTER, DROP, TRUNCATE) TABLE
- OPEN FETCH
- GRANT REVOKE
- INSERT DELETE UPDATE
- SELECT

SET IMPLICIT_TRANSACTIONS OFF

- BEGIN TRANSACTION
- COMMIT или ROLLBACK
- Явные транзакции можно вкладывать друг в друга
- Вложенные транзакции используются в хранимых процедурах, которые сами содержат транзакции и вызываются внутри другой транзакции
- @@TRANCOUNT содержит число активных транзакций для текущего пользователя

@@TRANCOUNT

```
----- @@TRANCOUNT
```

```
SELECT COUNT(*) '1', @@TRANCOUNT 'TRANCOUNT' FROM AUDITORIUM;  
BEGIN TRANSACTION A;
```

```
INSERT INTO AUDITORIUM VALUES ('128-1', 'ЛК', 60, '128-1');  
SELECT COUNT(*) '2', @@TRANCOUNT 'TRANCOUNT' FROM AUDITORIUM;
```

```
BEGIN TRANSACTION B;
```

```
DELETE AUDITORIUM where Auditorium = '128-1';
```

```
SELECT COUNT(*) '3', @@TRANCOUNT 'TRANCOUNT' FROM AUDITORIUM;
```

```
COMMIT TRANSACTION B;
```

```
SELECT COUNT(*) '4', @@TRANCOUNT 'TRANCOUNT' FROM AUDITORIUM;
```

```
INSERT INTO AUDITORIUM VALUES ('128-1', 'ЛК', 60, '128-1');
```

```
SELECT COUNT(*) '5', @@TRANCOUNT 'TRANCOUNT' FROM AUDITORIUM;
```

```
COMMIT TRANSACTION A;
```

```
SELECT COUNT(*) '6', @@TRANCOUNT 'TRANCOUNT' FROM AUDITORIUM;
```

1	TRANCOUNT
16	0
2	TRANCOUNT
17	1
3	TRANCOUNT
16	2
4	TRANCOUNT
16	1
5	TRANCOUNT
17	1
6	TRANCOUNT
17	0

Журнал транзакций

- Журнал транзакций применяется для отката или восстановления транзакции
- Для каждой базы данных собственный журнал транзакций
- Database Engine сохраняет значения до и после транзакции в журналах транзакций (transaction log)
 - Исходные образы записей (before image)
 - Преобразованные образы записей (after image)
 - LSN – порядковый номер для каждой записи
- Процессы:
 - Процесс отмены записей (undo)
 - Процесс повторного выполнения действий (redo)

Блокировки

- Блокировки – механизм обеспечения согласованности данных в случае одновременного обращения к данным нескольких пользователей
- Свойства:
 - Длительность блокировки
 - Режим блокировки
 - Гранулярность блокировки

Длительность блокировки

- Длительность блокировки — это период времени, в течение которого ресурс удерживает определенную блокировку

Режим блокировки

- Разделяемая (shared lock)
 - Монопольная (exclusive lock)
 - Обновления (update lock)
-
- СУБД автоматически выбирает соответствующий режим блокировки, в зависимости от типа операции (чтение или запись)

Разделяемая блокировка

- Разделяемая блокировка резервирует ресурс только для чтения
- Другие процессы не могут изменять заблокированный ресурс
- Может быть несколько разделяемых блокировок

Монопольная блокировка

- Монопольная блокировка резервирует страницу или строку для монопольного использования одной транзакции
- Применяется при INSERT, UPDATE и DELETE
- Монопольную блокировку нельзя установить, если на ресурс уже установлена какая-либо блокировка

Блокировка обновления

- Можно устанавливать на объекты с разделяемой блокировкой, накладывается еще одна разделяемая блокировка
- Нельзя устанавливать при наличии на нем другой блокировки обновления или монопольной блокировки
- При COMMIT транзакции обновления, блокировка обновления преобразовывается в монопольную блокировку
- У объекта может быть только одна блокировка обновления

Гранулярность блокировки

- Гранулярность блокировки определяет, какой объект блокируется:
 - строки
 - страницы
 - индексный ключ или диапазон индексных ключей
 - таблицы
 - экстент
 - база данных
- СУБД выбирает гранулярность блокировки автоматически

Гранулярность блокировки

- Процесс преобразования большого числа блокировок уровня строки, страницы или индекса в одну блокировку уровня таблицы называется эскалацией блокировок (lock escalation)
- ALTER TABLE
- SET (LOCK_ESCALATION = {TABLE | AUTO | DISABLE})
- Подсказки блокировок (locking hints)
- SET LOCK_TIMEOUT - период в миллисекундах, в течение которого транзакция будет ожидать снятия блокировки с объекта (-1 по умолчанию, не установлен)

БЛОКИРОВКИ

- sys.dm_tran_locks

```
- SELECT resource_type,
DB_NAME(resource_database_id) as database_name,
request_session_id, request_mode,
request_status
FROM sys.dm_tran_locks;
```

The screenshot shows a SQL Server Management Studio window with the 'Результаты' (Results) tab selected. The results of the query are displayed in a table:

resource_type	database_name	request_session_id	request_mode	request_status
DATABASE	BSTU	55	S	GRANT
DATABASE	BSTU	51	S	GRANT
DATABASE	BSTU	54	S	GRANT
DATABASE	BSTU	53	S	GRANT
DATABASE	BSTU	52	S	GRANT

Взаимоблокировки

- Взаимоблокировка (deadlock) — это особая проблема одновременного конкурентного доступа, в которой две транзакции блокируют друг друга

Взаимоблокировки

```
-----deadlock1
BEGIN TRANSACTION
---  
UPDATE AUDITORIUM
    SET AUDITORIUM_TYPE = 'ЛБ-К'
    WHERE AUDITORIUM_NAME = '128-1';
WAITFOR DELAY '00:00:10';
UPDATE AUDITORIUM
    SET AUDITORIUM_CAPACITY = 70
    WHERE AUDITORIUM_NAME = '127-1';
COMMIT;
```

```
-----deadlock2
BEGIN TRANSACTION
---  
UPDATE AUDITORIUM
    SET AUDITORIUM_TYPE = 'ЛК'
    WHERE AUDITORIUM_NAME = '127-1';
WAITFOR DELAY '00:00:10';
UPDATE AUDITORIUM
    SET AUDITORIUM_CAPACITY = 66
    WHERE AUDITORIUM_NAME = '128-1';
COMMIT;
```

Messages

Msg 1205, Level 13, State 51, Line 4
Transaction (Process ID 57) was deadlocked on lock resources

with another process and has been chosen as the deadlock victim. Rerun the transaction.

Уровни изоляции

- Уровень изоляции задает степень защищенности данных в транзакции от возможности изменения другими транзакциями

Проблемы

- Потеря обновлений
- Грязное чтение
- Неповторяющееся чтение
- Фантомное чтение

Потеря обновлений

- Несколько транзакций одновременно могут считывать и обновлять одни и те же данные
- Теряются все обновления данных, за исключением обновлений, выполненных последней транзакцией

Грязное чтение

- Происходит чтение несуществующих данных или потеря модифицированных данных

Неповторяющее чтение

- Один процесс считывает данные несколько раз, а другой процесс изменяет эти данные между двумя операциями чтения первого процесса
- Значения двух чтений будут разными

Фантомное чтение

- Последовательные операции чтения могут возвратить разные значения
- Считывание разного числа строк при каждом чтении
- Возникают дополнительные фантомные строки, которые вставляются другими транзакциями

УРОВНИ ИЗОЛЯЦИИ

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE
- SNAPSHOT

Уровни изоляции

- Пессимистическая модель:
 - READ UNCOMMITTED
 - REPEATABLE READ
 - SERIALIZABLE
- Оптимистическая модель:
 - SNAPSHOT
- Обе модели:
 - READ COMMITTED

READ UNCOMMITTED

- Не изолирует операции чтения других транзакций
- Транзакция не задает и не признает блокировок
- Допускает проблемы:
 - Грязное чтение
 - Неповторяющее чтение
 - Фантомное чтение

READ UNCOMMITTED

```
-- 1  
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED  
BEGIN TRAN  
SELECT COUNT(*) FROM AUDITORIUM -- запускаем транзакцию, Результат: 17
```

Results	
	Messages
(No column name)	
17	

```
-- 2  
BEGIN TRAN -- открываем параллельную транзакцию  
DELETE FROM AUDITORIUM WHERE AUDITORIUM='128-1' -- удаляем строку из таблицы  
-- 3  
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 16, налицо неподтвержденное чтение
```

Results	
	Messages
(No column name)	
16	

READ UNCOMMITTED

```
-- 4  
ROLLBACK TRAN -- откатываем транзакцию
```

```
-- 5  
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 17, после отката транзакции В  
COMMIT TRAN
```

Results	Messages
(No column name)	
17	

READ COMMITTED

- Транзакция выполняет проверку только на наличие монопольной блокировки для данной строки
- Является уровнем изоляции по умолчанию
- Проблемы:
 - Неповторяющее чтение
 - Фантомное чтение

READ COMMITTED

]----- Покажем, что уровень изолированности READ COMMITTED не допускает неподтвержденное чтение

-- 6

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  
BEGIN TRAN  
]SELECT COUNT(*) FROM AUDITORIUM -- запускаем транзакцию, Результат: 17
```

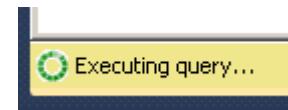
Results	Messages
(No column name)	
17	

-- 7

```
BEGIN TRAN -- открываем параллельную транзакцию  
]DELETE FROM AUDITORIUM WHERE AUDITORIUM='128-1' -- удаляем строку из таблицы
```

-- 8

```
]SELECT COUNT(*) FROM AUDITORIUM -- Результат: ожидание, неподтвержденного чтения нет
```



-- 9

```
]ROLLBACK TRAN -- откатываем транзакцию |
```

-- 10

```
SELECT COUNT(*) FROM AUDITORIUM -- сразу после отката транзакции В Результат: 17,  
]COMMIT TRAN
```

Results	Messages
(No column name)	
1	17

READ COMMITTED

----- Покажем, что уровень изолированности READ COMMITTED допускает неповторяющееся чтение

-- 11

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  
BEGIN TRAN  
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 17
```

Results	
(No column name)	
1	17

-- 12

```
BEGIN TRAN -- открываем параллельную транзакцию  
DELETE FROM AUDITORIUM WHERE AUDITORIUM = '128-1' -- удаляем строку из таблицы  
COMMIT TRAN
```

-- 13

```
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 16  
-- пока вторая транзакция удаляла запись, данные дважды прочитались по-разному.  
COMMIT TRAN
```

Results	
(No column name)	
16	

REPEATABLE READ

- Устанавливает разделяемые блокировки на все считываемые данные и удерживает эти блокировки до тех пор, пока транзакция не будет подтверждена или отменена
- Не препятствует другим инструкциям вставлять новые строки
- Проблема:
 - Фантомное чтение

REPEATABLE READ

----- Покажем, что уровень изолированности REPEATABLE READ не допускает неповторяющееся чтение

```
INSERT INTO AUDITORIUM VALUES ('128-1', 'ЛК', 60, '128-1'); -- вернем запись
```

-- 14

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

```
BEGIN TRAN
```

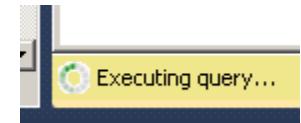
```
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 17
```

Results	Messages
(No column name)	
17	

-- 15

```
BEGIN TRAN -- открываем параллельную транзакцию
```

```
DELETE FROM AUDITORIUM WHERE AUDITORIUM = '128-1' -- удаляем строку из таблицы, результат - ожидание
```



-- 16

```
COMMIT TRAN -- сразу после фиксации транзакции А в окне В
```

-- Строк обработано:1 - прошло выполнение оператора удаления

-- 17

```
COMMIT TRAN -- завершаем транзакцию
```

REPEATABLE READ

----- Покажем, что уровень изолированности REPEATABLE READ допускает проблему фантомных записей

```
INSERT AUDITORIUM VALUES ('128-1', 'ЛК', 60, '128-1'); -- вернем запись
```

-- 18

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

```
BEGIN TRAN
```

```
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 17
```

Results	
	(No column name)
1	17

-- 19

```
BEGIN TRAN
```

```
INSERT INTO AUDITORIUM VALUES ('437-1', 'ЛК', 20, '437-1') -- Строк обработано:1
```

```
COMMIT TRAN -- завершаем транзакцию
```

-- 20

```
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 18
```

--- в рамках одной транзакции А два результата

```
COMMIT TRAN
```

Results	
	(No column name)
	18

SERIALIZABLE

- Устанавливает блокировку на всю область данных, считываемых соответствующей транзакцией
- Предотвращает вставку новых строк другой транзакцией до тех пор, пока первая транзакция не будет подтверждена или отменена

SERIALIZABLE

- Реализуется с использованием метода блокировки диапазона ключа
- Блокировка диапазона ключа блокирует элементы индексов

SERIALIZABLE

----- Покажем, что уровень изолированности SERIALIZABLE не допускает проблему фантомных записей

-- 21

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

```
BEGIN TRAN
```

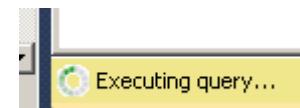
```
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 18
```

Results	
(No column name)	
1	18

-- 22

```
BEGIN TRAN
```

```
INSERT INTO AUDITORIUM VALUES ('446-1', 'ЛК', 20, '446-1') -- ожидание
```



-- 23

```
COMMIT TRAN -- после выполнения этой команды в сценарии В - Строк обработано:1
```

Установка уровня изоляции

- SET TRANSACTION ISOLATION LEVEL:
 - READ UNCOMMITTED
 - READ COMMITTED
 - REPEATABLE READ
 - SERIALIZABLE

Уровень изоляции

DBCC USEROPTIONS	
Set Option	Value
textsize	2147483647
language	us_english
dateformat	mdy
datefirst	7
lock_timeout	-1
quoted_identifier	SET
arithabort	SET
ansi_null_dflt_on	SET
ansi_warnings	SET
ansi_padding	SET
ansi_nulls	SET
concat_null_yields_null	SET
isolation level	read committed

Управление версиями строк

Механизм управления оптимистическим одновременным конкурентным доступом основан на управлении версиями строк

Для всех изменений данных создаются и поддерживаются логические копии

При каждом изменении строки СУБД сохраняет в tempdb исходный вид записи

Управление версиями строк

Каждая версия строки помечается порядковым номером транзакции (XSN — transaction sequence number)

Самая последняя версия строки сохраняется в базе данных и соединяется в связанном списке с версией, сохраненной в tempdb

Управление версиями строк

Поддержка уровней изоляции READ COMMITTED
SNAPSHOT и SNAPSHOT

Создание в триггерах таблиц inserted и deleted

READ COMMITTED SNAPSHOT

- Любая другая транзакция будет читать значения зафиксированные на момент начала этой транзакции
- ALTER DATABASE
- SET ISOLATION LEVEL
- READ COMMITTED SNAPSHOT

SNAPSHOT

- Уровень изоляции на уровне транзакций
- Любая другая транзакция будет читать подтвержденные значения в том виде, в каком они существовали непосредственно перед началом выполнения этой транзакции

SNAPSHOT

- На уровне базы данных включается
- ALLOW_SNAPSHOT_ISOLATION
- SET TRANSACTION ISOLATION LEVEL SNAPSHOT

SNAPSHOT

```
-- Установим ALLOW_SNAPSHOT_ISOLATION
USE master
GO
ALTER DATABASE B_BSTU SET ALLOW_SNAPSHOT_ISOLATION ON
GO
USE B_BSTU
GO

-- 24
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRAN
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 19

--- 25
BEGIN TRAN
INSERT INTO AUDITORIUM VALUES ('447-1','ЛК', 20, '447-1') -- выполняем вставку - Строк обработано:1

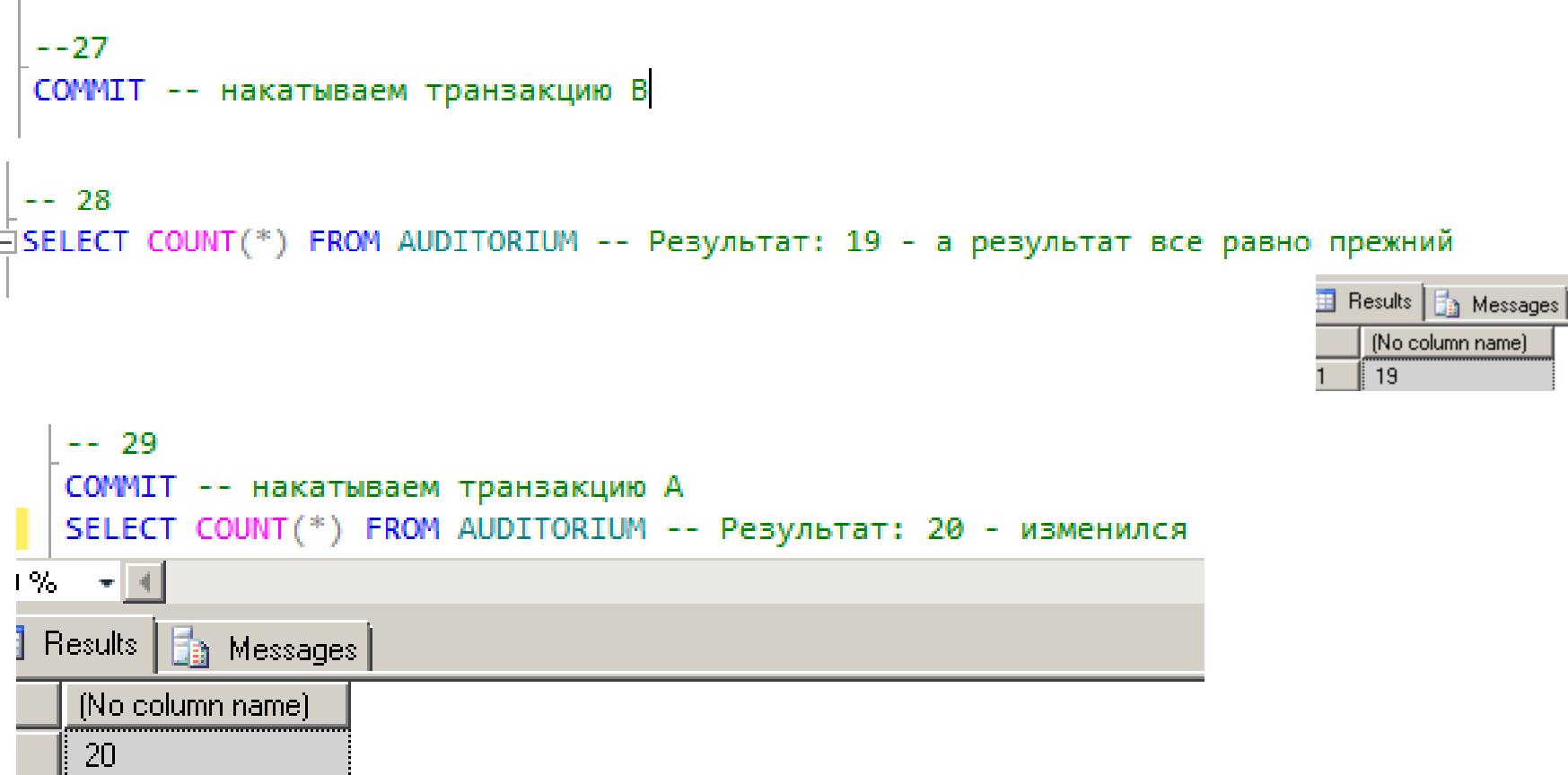
-- 26
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 19 - результат прежний
```

Results	Messages
(No column name)	
19	

Results	Messages
(No column name)	
19	

SNAPSHOT

```
--27  
COMMIT -- накатываем транзакцию В  
  
-- 28  
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 19 - а результат все равно прежний  
  
-- 29  
COMMIT -- накатываем транзакцию А  
SELECT COUNT(*) FROM AUDITORIUM -- Результат: 20 - изменился
```



The screenshot shows the execution of a T-SQL script in SQL Server Management Studio. The script demonstrates the behavior of snapshot isolation. It includes several comments (--) and two COMMIT statements. The first COMMIT statement (transaction B) is followed by a SELECT COUNT(*) query which returns 19. The second COMMIT statement (transaction A) is followed by the same SELECT query, which now returns 20, indicating that the count has changed since transaction B began. On the right, there are two results panes: 'Results' and 'Messages'. The 'Results' pane for transaction B shows a single row with value 19. The 'Results' pane for transaction A shows a single row with value 20.

Вопросы?