Лекция N11 Транзакции и блокировки

Транзакции

- Транзакция это последовательность операций, объединенных в единый логический рабочий модуль.
- Механизм транзакций позволяет контролировать выполнение операций в этом логическом модуле и производить откаты (отмену уже сделанной операции), если этого требует логика приложения.
- Рабочий модуль должен соответствовать основным требованиям к транзакциям, сокращенно называемые ACID (Atomicity, Consistency, Isolation, Durability)

- Atomicity (атомарность)Логика приложения должна предполагать, что должны быть проделаны либо все изменения данных, входящие в транзакцию либо ни одного;
- Consistensy(постоянство) После завершения транзакции не должна быть нарушена целостность данных,система не может оказаться в неком промежуточном состоянии;
- Isolation (изолированность)Изменения, производимые в рамках одной транзакции, изолируются от других (конкурирующих) транзакций;(4-уровня изоляции:0-двум процессам запрещается изменять одни и те же данные; 1-запрещено считывание пока идут изменения; 2-в промежутках чтения в одной TRAN не допускаются изменения в другой; 3-запрещаются в это время вставки и удаления)
- Durability (устойчивость)После завершения транзакции все сделанные изменения будут сделаны в любом случае, даже если во время этого процесса произошел сбой системы или потеря связи – после восстановления работоспособности SQL сервер обращается к журналу транзакций и производит изменения.

Запуск транзакции

SQL сервер позволяет запустить явную, автоматически совершаемую или неявную транзакцию

- Explicit (явная) транзакция предваряется выражением BEGIN TRANSACTION
- Autocommit (автоматически совершаемая) транзакция – режим, в котором работает SQL сервер по умолчанию, каждая отдельная инструкция T-SQL совершается (изменения в данные вносятся физически) после отрабатывания инструкции. Не нужно указывать никаких ключевых слов, чтобы начать такую транзакцию
- Implicit (неявная) транзакция. Такой режим транзакции устанавливается инструкцией SET IMPLICIT_TRANSACTIONS ON, следующая за этой инструкцией конструкция T-SQL автоматически начинает новую транзакцию. Когда эта транзакция завершается, следующее выражение начинает новую транзакцию.

Завершение транзакции.

- Для завершения транзакции используется конструкция COMMIT
- Если все прошло успешно, конструкция COMMIT гарантирует, что все изменения будут сделаны на физическом уровне.
- Если же во время выполнения транзакции произошла ошибка, используется конструкция ROLLBACK – данные возвращаются к первоначальному состоянию, или к некоторой точке сохранения, системные ресурсы освобождаются.

Синтаксис

- SAVE TRAN [SACTION] { *savepoint_name* | *@savepoint_variable* } – объявить savepoint
- BEGIN TRAN [SACTION] [*transaction_name* | *@tran_name_variable*
  [WITH MARK ['*description*']]]
- ROLLBACK [TRAN [SACTION]
  [*transaction_name* | *@tran_name_variable*
  | *savepoint_name* | *@savepoint_variable*]]
- COMMIT [TRAN [SACTION] [*transaction_name* | *@tran_name_variable*]]

Примеры:

```
CREATE TABLE ImplicitTran (Cola int PRIMARY KEY,
                           Colb char(3) NOT NULL)
SET IMPLICIT_TRANSACTIONS ON
/* Первая implicit транзакция начнется конструкцией INSERT*/
INSERT INTO ImplicitTran VALUES (1, 'aaa')
INSERT INTO ImplicitTran VALUES (2, 'bbb')
/* Commit first transaction */
COMMIT TRANSACTION
/* Вторая implicit транзакция начнется конструкцией SELECT */
SELECT COUNT(*) FROM ImplicitTran
INSERT INTO ImplicitTran VALUES (3, 'ccc')
SELECT * FROM ImplicitTran
/* Commit second transaction */
COMMIT TRANSACTION
SET IMPLICIT_TRANSACTIONS OFF
```

В autocommit режиме в случае возникновения ошибки компиляции SQL сервер делает rollback всему пакету инструкций. При возникновении runtime error откат не производится, не выполняется лишь инструкция, в которой произошел runtime error:

--example 1(autocommit ошибка компиляции)

```
CREATE TABLE TestBatch (Cola INT PRIMARY KEY, Colb   CHAR(3))
INSERT INTO TestBatch VALUES (1, 'aaa')
INSERT INTO TestBatch VALUES (2, 'bbb')
INSERT INTO TestBatch VALUSE (3, 'ccc')  /* Syntax error */
SELECT * FROM TestBatch   /* Returns no rows */
```

--example 2(autocommit ошибка runtime error)

```
CREATE TABLE TestBatch (Cola INT PRIMARY KEY, Colb CHAR(3))
INSERT INTO TestBatch VALUES (1, 'aaa')
INSERT INTO TestBatch VALUES (2, 'bbb')
INSERT INTO TestBatch VALUES (1, 'ccc')  /* Duplicate key error */
SELECT * FROM TestBatch   /* Returns rows 1 and 2 */
```

В SQL Server контроль имени объекта производится в

(во время выполнения) execution time. Поэтому в следующем примере будет runtime error, а не compile error:

```
USE pubs
GO
CREATE TABLE TestBatch (Cola INT PRIMARY KEY,
Colb CHAR(3))
GO
INSERT INTO TestBatch VALUES (1, 'aaa')
INSERT INTO TestBatch VALUES (2, 'bbb')
INSERT INTO TestBch VALUES (3, 'ccc')  /* Table name error */
GO
SELECT * FROM TestBatch   /* Returns rows 1 and 2 */
GO
```

Обработка исключений

- @@ERROR возвращает номер ошибки, возникшей при выполнении предыдущего SQL-выражения или 0, если ошибок не было. Сообщения, соответствующие каждому коду ошибки хранятся в таблице sysmessages
- @@ROWCOUNT возвращает количество строк, затронутых предыдущим запросом
- RAISEERROR – генерирует исключение

RAISERROR **(** { *msg_id* | *msg_str* } { **,** *severity* **,** *state* }
[**,** *argument* [**,**...*n*]] **)**
[WITH *option* [**,**...*n*]]

severity - тип проблемы (использовать 11-16)

state – произвольное число от 1 до 127

Схема обработки ошибок в explicit транзакции:

```
begin tran
Update Authors
    set contact=1 where au_id=1000
Save transaction Authors_done
Update AU_titles
    set au_num=au_num+3 where au_id=1000
if @@error <> 0 or @@rowcount >1
begin
    raiserror('Couldn''t update ',16,1)
    print ‘error_update’
    rollback tran Authors_done
    return
end
commit tran
Print ‘tran’’s ok!’
```

Блокировки

SQL-сервер позволяет одновременную работу большого числа пользователей, каждый из которых запускает свои запросы в параллельных сессиях. Это приводит к возможности одновременного доступа нескольких сессий к одним и тем же данным (доступ может быть как чтением, так и изменением данных). Одновременная работа таких конкурирующих запросов может привести к следующим проблемам:

- «проблема потери последнего изменения» (lost update problem) или проблема «грязной» записи
- проблема «грязного чтения» (dirty read)
- проблема «повторного чтения» (repeatable read)
- проблема «фантомов» (phantom)

Проблема «грязной» записи заключается в том, что при одновременном выполнении транзакций, в которых производится изменение данных, невозможно сказать заранее, какое конечное значение примут данные после фиксирования обеих транзакций. В случае «грязной» записи только одна из всех параллельно выполняющихся транзакций будет работать с действительными данными, остальные – нет. Другими словами, хотя данные и будут находиться в согласованном состоянии, логическая их целостность будет нарушена.

Проблема «грязного чтения» возникает, когда одна транзакция пытается прочитать данные, с которыми работает другая параллельная транзакция. В таком случае временные, неподтвержденные данные могут не удовлетворять ограничениям целостности или правилам. И, хотя к моменту фиксации транзакции они могут быть приведены в «порядок», другая транзакция уже может воспользоваться этими неверными данными, что приведет к нарушению ее работы.

Проблема повторного чтения состоит в том, что между операциями чтения в одной транзакции другие транзакции могут беспрепятственно вносить любые изменения, так что повторное чтение тех же данные приведет к другому результату.

Проблема фантомов возникает возникает, когда выборка данных сделанная в одной транзакции изменяется другой транзакцией. Например мы устанавливаем для поля F1 значение 5000 для всех записей таблицы, затем накладываем constraint, ограничивающий поле F1 сверху числом 5001, а в промежуток между этими 2мя операциями другая транзакция вносит в таблицу запись с F1 = 5500

- Для разрешения этих проблем необходимо **изолировать** транзакции друг от друга.

- Для реализации различных уровней изоляции в SQL сервере используются блокировки (LOCKs)

- Блокировки – чрезвычайно важный и неотъемлемый механизм функционирования сервера. Они применяются для каждого запроса на чтение или обновления данных, а также во многих других случаях (например, при создании новой сессии). Работой с блокировками занимается специальный модуль SQL Server'а – менеджер блокировок (Lock Manager).

Уровни определения изоляции транзакций(Каждый уровень включает в себя предыдущий с предъявлением более жестких требований к изоляции)

- No trashing of data (запрещение «загрязнения» данных). Запрещается изменение одних их тех же данных двумя и более параллельными транзакциями. Изменять данные может только одна транзакция, если какая-то другая транзакция попытается сделать это, она должна быть заблокирована до окончания работы первой транзакции.

- No dirty read (запрещение «грязного» чтения). Если данная транзакция изменяет данные, другим транзакциям запрещается читать эти данные до тех пор, пока первая транзакция не завершится.

- No nonrepeatable read (запрещение неповторяемого чтения). Если данная транзакция читает данные, запрещается изменять эти данные до тех пор, пока первая транзакция не завершит работу. При этом другие транзакции могут получать доступ на чтение данных.

- No phantom (запрещение фантомов). Если данная транзакция производит выборку данных, соответствующих какому-либо логическому условию, другие транзакции не могут ни изменять эти данные, ни вставлять новые данные, которые удовлетворяют тому же логическому условию.

- Блокировки в MS SQL Server 2000 (– это механизм реализации требования изолированности транзакций.
- Существует три основных типа блокировок и множество специфичных. Сервер устанавливает блокировки автоматически в зависимости от текущего уровня изоляции транзакции, однако при желании вы можете изменить тип с помощью специальных подсказок – хинтов.
- При открытии новой сессии по умолчанию выбирается уровень изоляции READ COMMITTED.
- Вы можете изменить этот уровень для данного соединения с помощью команды:

SET TRANSACTION ISOLATION LEVEL

Уровни изоляции	Загрязнение данных	Грязное чтение	Неповторяемое чтение	Фантомы
READ UNCOMMITTED	-	+	+	+
READ COMMITTED	-	-	+	+
REPEATABLE READ	-	-	-	+
SERIALIZABLE	-	-	-	-

Блокировки применяются для защиты совместно используемых ресурсов сервера. В качестве объектов блокировок могут выступать следующие сущности:

- База данных (обозначается DB). При наложении блокировки на базу данных блокируются все входящие в нее таблицы.
- Таблица (обозначается TAB). При наложении блокировки на таблицу блокируются все экстенты данной таблицы, а также все ее индексы.
- Экстент (обозначается EXT). При наложении блокировки на экстент блокируются все страницы, входящие в данный экстент.
- Страница (обозначается PAG). При наложении блокировки на страницу блокируются все строки данной страницы.
- Строка (обозначается RID).
- Диапазон индекса (обозначается KEY). Блокируются данные, соответствующие диапазону индекса, на обновление, вставку и удаление.

- SQL Server сам выбирает наиболее оптимальный объект для блокировки, однако пользователь может изменить это поведение с помощью тех же хинтов.
- При автоматическом определении объекта блокировки сервер должен выбрать наиболее подходящий с точки зрения производительности и параллельной работы пользователей.
- Чем меньше детализация блокировки (строка – самая высокая степень детализации), тем ниже ее стоимость, но ниже и возможность параллельной работы пользователей.
- Если выбирать минимальную степень детализации, запросы на выборку и обновление данных будут исполняться очень быстро, но другие пользователи при этом должны будут ожидать завершения транзакции.
- Степень параллелизма можно увеличить путем повышения уровня детализации, однако блокировка – вполне конкретный ресурс SQL Server’а, для ее создания, поддержания и удаления требуется время и память.

- SQL Server может принимать решение об уменьшении степени детализации, когда количество блокированных ресурсов увеличивается. Этот процесс называется **эскалацией блокировок.**

- Вообще говоря, существует два метода управления конкуренцией для обеспечения параллельной работы множества пользователей – оптимистический и пессимистический. SQL Server использует оптимистическую конкуренцию только при использовании курсоров (cursors). Для обычных запросов на выборку и обновление используется пессимистическая конкуренция.

- Оптимистический метод управления характеризуется тем, что вместо непосредственного чтения данных берется значение из буфера. Никаких блокировок при этом не накладывается. Другие транзакции могут спокойно читать или даже изменять данные. В момент фиксирования транзакции система сравнивает предыдущее (заранее сохраненное) значение данных с текущим. Если они совпадают, выполняются операции блокировки, обновления и разблокировки данных. Если же значения отличаются, то система генерирует ошибку и откатывает транзакцию.

- Пессимистический метод. В этом случае сервер всегда блокирует ресурсы в соответствии с текущим уровнем изоляции.

Основные задачи менеджера блокировок:

- создание и установка блокировок;
- снятие блокировок;
- эскалация блокировок;
- определение совместимости блокировок;
- устранение взаимоблокировок (deadlocks)

- Когда пользователь делает запрос на обновление или чтение данных, менеджер транзакций передает управление менеджеру блокировок для того, чтобы выяснить были ли блокированы запрашиваемые ресурсы, и, если да, совместима ли запрашиваемая блокировка с текущей.

- Если блокировки несовместимы, выполнение текущей транзакции откладывается до тех пор, пока данные не будут разблокированы.

- Как только данные становятся доступны, менеджер блокировок накладывает запрашиваемую блокировку, и возвращает управление менеджеру транзакций.

Простые блокировки

- **Разделяемая блокировка (Shared Lock)**, обозначается латинской буквой S. Эта самый распространенный тип блокировки, который используется при выполнении операции чтения данных. Гарантируется что данные, на которые она наложена, не будут изменены другой транзакцией. Однако чтение данных возможно.

- **Монопольная блокировка (Exclusive Lock)**, обозначается латинской буквой X. Этот тип применяется при изменении данных. Если на ресурс установлена монопольная блокировка, гарантируется, что другие транзакции не могут не только изменять данные, но даже читать их.

- **Блокировка обновления (Update Lock)**, обозначается латинской буквой U. Эта блокировка является промежуточной между разделяемой и монопольной блокировкой.

- Так как монопольная блокировка не совместима ни с одним видом других блокировок ее установка приводит к полному блокированию ресурса.

- Если транзакция хочет обновить данные в какой-то ближайший момент времени, но не сейчас, и, когда этот момент придет, не хочет ожидать другой транзакции, она может запросить блокировку обновления.

- В этом случае другим транзакциям разрешается устанавливать разделяемые блокировки, но не позволяет устанавливать монопольные.

- Если данная транзакция установила на ресурс блокировку обновления, никакая другая транзакция не сможет получить на этот же ресурс монопольную блокировку или блокировку обновления до тех пор, пока установившая блокировку транзакция не будет завершена.

- Для просмотра текущих блокировок существует системная хранимая функция sp_lock.

- Эта процедура возвращает данные о блокировках из системной таблицы syslockinfo, которая находится в базе данных master.

Пример:

```
create table test(i int, n varchar(20))
insert into test  values(1,'alex')
insert into test values(2,'rosa')
insert into test values(3,'dima')
```

убедимся, что при чтении данных с уровнем изоляции ниже REPEATABLE READ разделяемые блокировки снимаются сразу же после извлечения данных:

```
print @@spid
begin tran select * from test
```

в другой сессии запустим:

```
sp_lock 63
```

Мы видим стандартную блокировку, которая создается для каждого соединения с базой данных. Никакой дополнительной блокировки установлено не было.

В первой сессии зафиксируем транзакцию:

```
--print @@spid
--begin tran select * from test
commit
```

Повторный вызов sp_lock приводит к тем же результатам. Это подтверждает, что предыдущим запросом никаких блокировок не устанавливалось. Теперь попробуем наложить блокировку обновления. Делается это с помощью хинта updlock (хинты подробно будут рассмотрены далее):

```
begin tran select * from test with (updlock)
```

spdi	dbid	ObjId	IndId	Type	Resource	Mode
54	8	0	0	DB	S	GRANT
54	8	1993058136	0	RID	1:29:2	U	GRANT
54	8	1993058136	0	RID	1:29:0	U	GRANT
54	8	1993058136	0	PAG	1:29	IU	GRANT
54	8	1993058136	0	TAB	IX	GRANT
54	8	1993058136	0	RID	1:29:1	U	GRANT

Как видно, на три строки была наложена блокировка обновления, что означает невозможность обновления этих строк другими транзакциями. Кроме этого, были наложены еще две блокировки, которые относятся к типу блокировок намерения (intent locks) – блокировка на страницу и на таблицу.

- Блокировки намерений всегда устанавливаются на таблицу или страницу, но никогда – на строку.
- Блокировки намерений относятся к специальным типам блокировок и предназначены для повышения производительности работы менеджера блокировок.
- Предположим, некая транзакция пытается изменить какую-либо строку в таблице test. Чтобы определить, что эту транзакцию необходимо заблокировать, менеджеру транзакций (в отсутствие блокировок намерения) пришлось бы сканировать всю таблицу syslockinfo для проверки всех строк таблицы test. Чтобы избежать этой неблагодарной работы, менеджер блокировок сразу устанавливает на страницу и таблицу блокировку намерения обновления (Intent Update) и монопольную блокировку намерения (Intent Exclusive) соответственно, и проверяет уже только их.

Типы блокировок намерений

- Разделяемая блокировка намерений (обозначается IS).
- Монопольная блокировка намерений (обозначается IX).
- Разделяемо-монопольная блокировка намерений (обозначается SIX)

Продолжим пример. Создадим новую сессию и выполним следующий скрипт:

```
begin tran
insert into test values(4,'other')
```

pid	dbid	ObjId	IndId	Type	Resource	Mode
54	8	0	0	DB	S	GRANT
54	8	1993058136	0	RID	1:29:02	U	GRANT
54	8	1993058136	0	RID	1:29:00	U	GRANT
54	8	1993058136	0	PAG	1:29	IU	GRANT
54	8	1993058136	0	TAB	IX	GRANT
54	8	1993058136	0	RID	1:29:01	U	GRANT
55	8	0	0	DB	S	GRANT
55	8	1993058136	0	PAG	1:29	IX	GRANT
55	8	1993058136	0	TAB	IX	GRANT
55	8	1993058136	0	RID	1:29:03	X	GRANT

Как видно, предыдущие блокировки остались (так как мы не зафиксировали транзакцию), и добавились четыре новых: одна блокировка базы, не имеющая никакого значения, две блокировки намерений (на таблицу и страницу) и монопольная блокировка на новую строку (идентификатор 1:29:03).