Транзакции и блокировки

34. Определение транзакции. Свойства транзакций. Успешное завершение и откат транзакций. Операторы Transact SQL для работы с транзакциями. Контрольные точки.

Понятие транзакции

При работе с базами данных не исключены ошибки и сбои. Они могут быть вызваны ошибками пользователей, взаимодействующих с СУБД, или неустойчивой работой компьютеров. Поэтому в СУБД применяют специальные способы отмены действий, вызвавших такие ошибки.

С помощью транзакций и блокировок обеспечивается согласованность и целостность данных, несмотря на возникающие в системе ошибки.

Транзакцией называется последовательность операций, производимых над базой данных и переводящих ее из одного непротиворечивого состояния в другое непротиворечивое состояние.

Примером транзакции может быть перевод денег через банкомат.

Сумма 100 т.р. переводится с одного счета на другой. Программа вычитает сумму с первого счета, после чего прибавляет ее ко второму. Во время работы программы после выполнения первой модификации может произойти сбой питания, и увеличение второго счета не произойдёт.

Для того чтобы избежать подобной ситуации обе команды должны быть объединены в транзакцию. В случае, когда все команды транзакции не выполняются, происходит откат транзакции.

Пример транзакции по вводу данных о вновь поступивших в библиотеку книгах.

Эту операцию можно разбить на 2 последовательные: сначала ввод данных о книге – это новая строка в таблице **Книги.** Затем необходимо ввести данные обо всех экземплярах книги - это ввод набора новых строк в таблицу **Экземпляры.** Если эта последовательность действий будет прервана, то база данных не будет соответствовать реальному объекту, поэтому желательно выполнять ее как единую работу над базой данных.

Транзакция **– это неделимая, с точки зрения воздействия на базу данных, последовательность операций манипулирования данными.**

Транзакции позволяют обработать набор операторов Transact-SQL как единое целое.

Существуют различные модели транзакций, которые могут быть классифицированы на основе различных свойств, включающих структуру транзакции, параллельность внутри транзакции, продолжительность и т.д.

Классические транзакции характеризуются классическими свойствами:

- **Неразрывности (атомарности),**
- **согласованности,**
- **изолированности,**
- **долговечности.**

- Свойство **атомарности** выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе (“Все или ничего”).

- Свойство **согласованности** гарантирует, что по мере выполнения транзакции данные переходят из одного согласованного состояния в другое согласованное состояние - **транзакция не разрушает взаимной согласованности данных.**

Свойство **изолированности** означает, что конкурирующие за доступ к БД транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.

В многопользовательских системах с одной базой данных одновременно может работать несколько пользователей или прикладных программ.

Задачей системы является обеспечение изолированности пользователей, т.е. создание достоверной и надёжной иллюзии того, что каждый из пользователей работает с базой данных в одиночку.

Свойство **долговечности** означает, что если транзакция завершена успешно, то те изменения данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах, даже в случае последующих ошибок.

Эти свойства (атомарность, согласованность, изоляция и устойчивость) называются **ACID**-свойствами (от англ, atomicity, consistency, isolation, durability).

2 варианта завершения транзакции:

1. Если все операторы выполнены успешно и в процессе транзакции не произошло никаких сбоев программного или аппаратного обеспечения, **транзакция фиксируется**.

(**Фиксация** – это запись на диск изменений в БД, которые были сделаны в процессе выполнения транзакции). До тех пор, пока транзакция не зафиксирована, эти изменения могут быть аннулированы и база данных может быть возвращена в то состояние, в котором она была на момент начала транзакции.

Фиксация транзакции означает, что все результаты выполнения транзакции становятся постоянными и станут видимы другим транзакциям только после того, как текущая транзакция будет зафиксирована.

2. Если в процессе выполнения транзакции произошёл сбой, БД должна быть возвращена в исходное состояние.

Откат транзакции – это действие, обеспечивающее аннулирование всех изменений данных, которые были сделаны операторами SQL в теле текущей незавершённой транзакции.

Режимы транзакций. Операторы Transact SQL для работы с транзакциями. Явные транзакции

После завершения транзакции вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не будет выполнена одна из команд управления транзакциями.

Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.

Транзакция может начинаться в одном из трех режимов: автофиксация (autocommit), явный режим (explicit) или неявный режим (implicit). По умолчанию для SQL Server принят режим автофиксации. Рассмотрим, что означает каждый из этих режимов.

В соответствии со стандартом языка SQL:1999 транзакции могут образовываться **явным** образом с использованием оператора **START TRANSACTION**, либо неявно, когда выполняется оператор, для которого требуется контекст транзакции. Например, операторы SELECT, UPDATE или CREATE TABLE могут выполняться только в контексте транзакции.

Явная транзакция — это транзакция, начало и конец которой определены явно.

В стандарте SQL99 определены операторы

- **START TRANSACTION,**
- **COMMIT,**
- **ROLLBACK.**

Оператор **COMMIT** означает успешное завершение транзакции, результаты транзакции фиксируются во внешней памяти.

При завершении транзакции оператором **ROLLBACK** результаты транзакции отменяются.

в расширенной модели транзакций (например, в СУБД SQL SERVER) предусмотрены следующие операции:

- оператор **BEGIN TRANSACTION** сообщает о начале транзакции;

- оператор **COMMIT TRANSACTION** сообщает об успешном завершении транзакции. Этот оператор, также как и **COMMIT** в модели стандарта, фиксирует все изменения, которые производились в БД в процессе выполнения транзакции;

.

- оператор **SAVE TRANSACTION** создаёт внутри транзакции точку сохранения, которая соответствует промежуточному состоянию БД, сохранённому на момент выполнения этого оператора.

В операторе **SAVE TRANSACTION** может стоять имя точки сохранения, поэтому в ходе выполнения транзакции может быть запомнено несколько точек сохранения соответствующих нескольким промежуточным состояниям.

- Оператор **ROLLBACK** имеет 2 модификации.

Если он используется без дополнительного параметра, то он интерпретируется как оператор отката всей транзакции, если же он имеет параметр

ROLLBACK TO SAVEPOINT n,

то он интерпретируется как оператор частичного отката транзакции в точку сохранения n.

Точки сохранения целесообразно использовать в длинных и сложных транзакциях, чтобы обеспечить возможность отмены изменений, выполненных определёнными операторами.

Не рекомендуется организовывать работу так, чтобы транзакции содержали много команд, тем более не связанных между собой.
Это может привести к тому, что при отмене изменений будет выполнено слишком много действий, в том числе и тех, которые являются нужными и ошибки не вызвали.

Лучший вариант, когда транзакция состоит из одной команды или нескольких тесно связанных команд.

```
BEGIN TRANSACTION
INSERT INTO Студенты
VALUES(2005,'Иванов',NULL,NULL,NULL,NULL
,1,NULL)
UPDATE оценки SET код_студента=2005
WHERE код_студента =2009
DELETE FROM Студенты WHERE
код_студента=2009
IF @@ERROR = 0
 COMMIT
ELSE
 ROLLBACK
```

Триггер выполняется как неявно определённая транзакция, поэтому внутри триггера допускается применение команд управления транзакциями (***ROLLBACK TRANSACTION***, ***COMMIT TRANSACTION***).

Выполнение команды ***ROLLBACK TRANSACTION*** или ***COMMIT TRANSACTION*** не прерывает работу триггера.

```
DELETE FROM Товар WHERE КодТовара=2
      SAVE TRANSACTION  point2
```

В точке point2 сохраняется состояние таблицы Товар без товаров с кодом 2.

```
DELETE FROM Товар WHERE КодТовара=3
      SAVE TRANSACTION  point3
```

В точке point3 сохраняется состояние таблицы Товар без товаров с кодом 2 и с кодом 3.

```
DELETE FROM Товар WHERE КодТовара<>1
      ROLLBACK TRANSACTION point3
```

Происходит возврат в состояние таблицы без товаров с кодами 2 и 3, отменяется последнее удаление.

```
SELECT * FROM Товар
```

Оператор SELECT покажет таблицу Товар без товаров с кодами 2 и 3.


```
ROLLBACK TRANSACTION point1
```

Транзакциям могут быть присвоены имена. В этом случае команда сообщающая о начале транзакции имеет вид ***BEGIN TRANSACTION<имя транзакции>***

Неявный режим

В неявном режиме транзакция автоматически начинается при использовании определённых операторов T-SQL и продолжается, пока не появится оператор явного окончания COMMIT или ROLLBACK.

ALTER TABLE
CREATE TABLE
DELETE
DROP
FETCH
GRANT
INSERT
OPEN
REVOKE
SELECT
TRUNCATE TABLE
UPDATE

Чтобы задать неявный режим транзакций, используется оператор:

SET IMPLICIT_TRANSACTIONS {ON | OFF}

Значение ON активизирует неявный режим, и OFF отключает его. После отключения неявного режима используется режим автофиксации.

Поскольку здесь не используется явная инструкция **BEGIN TRANSACTION**, легко забыть о необходимости фиксации или отката транзакции, что может вызвать длительно работающие транзакции, нежелательные откаты при закрытии соединений и проблемы с блокировками для других соединений.

Транзакции с автоматической фиксацией

Режим автоматической фиксации транзакций является режимом управления транзакциями SQL Server по умолчанию.

В этом режиме каждый завершенный оператор Transact SQL либо фиксируется, либо откатывается. Если оператор выполнен успешно, он фиксируется, если при его исполнении возникла ошибка, выполняется откат.

Когда SQL Server использует транзакции с автофиксацией, каждая инструкция рассматривается как транзакция.

Если одна инструкция генерирует ошибку, соответствующая ей транзакция автоматически подвергается откату. Если инструкция успешно и без ошибок выполняется, то транзакция автоматически фиксируется. Например, инструкции 1 и 3 пакета могут быть зафиксированы, а инструкция 2, вызвавшая ошибку, отменена.

Пакетное выполнение не определяет, следует ли обрабатывать все инструкции в пакете как единую транзакцию.

Пример транзакции:

```
BEGIN TRAN
UPDATE account
SET balance= balance- 100
WHERE account_number=@s
If @@ error<>0
BEGIN
ROLLBACK TRAN
RETURN
END
UPDATE card_account
SET balance=balance+100
WHERE account_number=@s
If @@ error<>0
BEGIN
ROLLBACK TRAN
RETURN
END
COMMIT TRAN
```

Вложенные транзакции

Вложенными называются транзакции, выполнение которых инициируется из тела уже активной транзакции.
Для создания вложенной транзакции пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Завершение транзакции верхнего уровня откладывается до завершения вложенных транзакций.

Если транзакция самого нижнего (вложенного) уровня завершена неудачно и отменена, то все транзакции верхнего уровня, включая транзакцию первого уровня, будут отменены. Кроме того, если несколько транзакций нижнего уровня были завершены успешно (но не зафиксированы), однако на среднем уровне (не самая верхняя транзакция) неудачно завершилась другая транзакция, то в соответствии с требованиями ACID произойдет откат всех транзакций всех уровней, включая успешно завершенные.

Только когда все транзакции на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения транзакции верхнего уровня.

Если команда ROLLBACK TRANSACTION используется на любом уровне вложенности без указания имени транзакции, то откатываются все вложенные транзакции, включая транзакцию самого высокого (верхнего) уровня.

В команде ROLLBACK TRANSACTION разрешается указывать только имя самой верхней транзакции. Имена любых вложенных транзакций игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате транзакции любого уровня вложенности всегда происходит откат всех транзакций.

Если же требуется откатить лишь часть транзакций, можно использовать команду SAVE TRANSACTION , с помощью которой создается точка сохранения.

Транзакционность поддерживается следующими архитектурными компонентами SQL Server:

- **журналами транзакций,**
- **механизмом управления параллельным выполнением,**
- **блокировками.**

35. Журнал транзакций

Реализация принципа подтверждения или отката транзакций обеспечивается специальным механизмом, для поддержки которого создана системная структура, называемая **журналом транзакций.**

.

Журнал транзакций содержит последовательность записей об изменении БД. Он предназначен для обеспечения надёжного хранения данных в БД.

Информация, записывающаяся в журнал транзакций, включает:

- Время начала каждой транзакции;
- Изменения внутри каждой транзакции;
- Информация о фиксации или откате каждой транзакции.

Целью журнализации изменений баз данных является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя.

Общие принципы журнализации и восстановления:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии БД, т.е. должно поддерживаться свойство *долговечности* транзакций);

- результаты незафиксированных транзакций не должны присутствовать в восстановленном состоянии БД.

Это означает, что восстанавливается последнее по времени согласованное состояние БД.

Возможны следующие ситуации, при которых требуется производить восстановление состояния БД:

- Восстановление после внезапной потери содержимого оперативной памяти (**мягкий сбой**). Такая ситуация может возникнуть в следующих случаях: при аварийном выключении электропитания или при возникновении неустранимого сбоя процессора. Ситуация характеризуется потерей той части базы данных, которая находилась к моменту сбоя в буферах оперативной памяти.

- Восстановление после поломки основного внешнего носителя БД (**жесткий сбой**).

При **мягком сбое** необходимо восстановить содержимое БД по содержимому журналов транзакций, хранящихся на дисках.

При **жёстком сбое** необходимо восстановить содержимое БД по архивным копиям и журналам транзакций.

Для большей надежности журнал транзакций часто дублируется системными средствами СУБД, именно поэтому объем внешней памяти во много раз превышает реальный объем данных в базе.

Имеется 2 варианта ведения журнала транзакций:

- **протокол с отложенными обновлениями;**
- **протокол с немедленными обновлениями.**

Ведение журнала по принципу отложенных обновлений предполагает следующий механизм выполнения транзакций:

Когда транзакция T1 начинается, в протокол заносится запись ***T1 Begin Transaction***

1. На протяжении выполнения транзакции в протоколе для каждой изменяемой записи записывается новое значение
   T 1.ID_RECORD, атрибут, новое значение
3. Если все действия, из которых состоит транзакция, успешно выполнены, в протокол заносится:
   T1 COMMIT
4. Записи протокола, относящиеся к T1, используются для внесения изменений в БД.
5. Если происходит сбой, то СУБД просматривает протокол и выясняет, какие транзакции необходимо переделать. Транзакцию T1 необходимо переделать, если протокол содержит обе записи ***T1 Begin Transaction*** и ***T1 COMMIT***. БД может находиться в несогласованном состоянии, однако все новые значения измененных элементов данных содержатся в протоколе, и это требует повторного выполнения транзакции. Для этого используется системная процедура **REDO(),** которая заменяет все значения элементов данных на новые, просматривая протокол в прямом порядке.
6. Если в протоколе не содержится команда фиксации транзакции **COMMIT**, то никаких действий проводить не требуется, а транзакция запускается заново.

Альтернативный механизм с немедленным выполнением предусматривает внесение изменений сразу в БД, а в протокол заносятся не только новые, но и все старые значения изменяемых атрибутов, поэтому каждая запись выглядит так:

T1.ID_RECORD, атрибут, новое значение, старое значение

При этом запись в журнал предшествует непосредственному выполнению операции над БД.

Когда встречается команда **T1 COMMIT**, и она выполняется, то все изменения оказываются уже внесенными в БД и не требуется никаких дальнейших действий по отношению к этой транзакции.

При откате транзакции выполняется системная процедура **UNDO()**, которая возвращает все старые значения в отмененной транзакции, последовательно проходя по протоколу, начиная с команды **BEGIN TRANSACTION.**

Для восстановления при сбое используется следующий механизм:

Если транзакция содержит команду начала транзакции, но не содержит команду фиксации с подтверждением ее выполнения, то выполняется последовательность действий как при откате транзакции, то есть восстанавливаются старые значения.

На самом деле восстановление происходит по более сложным алгоритмам, т.к. изменения, как в журнал, так и в БД заносятся не сразу, а буферизуются.

Если бы запись об изменении БД, которая должна поступать в журнал при выполнении любой операции модификации БД, на самом деле немедленно записывалась во внешнюю память, это привело бы к существенному замедлению работы системы.

Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем изменённый объект оказывается во внешней памяти базы данных.

Соответствующий протокол журнализации (и управления буферизацией) называется **Write Ahead Log** (**WAL**) — "пиши сначала в журнал" и состоит в том, что если требуется записать во внешнюю память измененный объект базы данных, то перед этим нужно гарантировать запись во внешнюю память журнала транзакций записи о его изменении.

Если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции.

Обратное неверно, то есть если во внешней памяти журнале содержится запись о некоторой операции изменения объекта базы данных, то сам изменённый объект может отсутствовать во внешней памяти базы данных.

UPDATE Table SET c1 = 10 WHERE c2 LIKE '% Иванов%';

Выполняются следующие операции.

- Страницы данных из **Table** считываются с диска в память, поэтому можно выполнять поиск соответствующих строк. Например оказывается, что на трех страницах данных имеется пять строк, соответствующих предикату предложения WHERE.
- Автоматически запускается транзакция.
- Эти три страницы блокируются для выполнения обновлений.
- Изменения вносятся в пять записей данных на трех страницах данных, находящихся в памяти.
- **Изменения записываются в записи журнала транзакций на диске.**
- Автоматически фиксируется транзакция.

36. Параллельная работа пользователей.

Проблемы параллельного выполнения: потерянное обновление, доступ к промежуточным результатам транзакции, строки-фантомы.

Когда множество пользователей одновременно пытаются модифицировать данные в БД, необходимо создать систему управления, которая защитила бы модификации, выполненные одним пользователем, от негативного воздействия модификаций, сделанных другими. Этот процесс называется **управлением параллельным выполнением.**

Как правило, в среде OLTP (OnLine Transaction Processing – оперативная обработка транзакций) наиболее важным требованием к организации работы является обеспечение параллельного выполнения операций.

С другой стороны, в среде OLAP (Online Analytical Processing – оперативная аналитическая обработка данных) проблема обеспечения максимальной производительности за счет параллельной организации работы в основном теряет свою остроту, хотя во многих системах ее нельзя полностью переводить на второй план. В целом следует отметить, что без параллельной организации работы невозможно добиться приемлемой производительности системы. Основой параллельной организации работы в базах данных являются операции, называемые **блокировками**.

Управление блокировками

С помощью блокировок обеспечивается целостность и согласованность БД при параллельной работе.

Блокировки запрещают пользователям читать данные, изменяемые другими пользователями, и предотвращают одновременную модификацию одних и тех же данных несколькими пользователями.

Без использования блокировок логическая целостность данных БД может нарушиться, в результате при запросе выводятся искажённые данные.

Проблемы параллельного выполнения

Если несколько пользователей одновременно обращаются к БД при отсутствии блокировок, возможны ошибки при одновременном использовании в их транзакциях одних и тех же данных.

- потерянные обновления;
- чтение незафиксированных данных ;
- несогласованный анализ (неповторяемое чтение);
- чтение фантомов.

Потерянные обновления

Потерянные обновления появляются при выборе одной строки двумя или более транзакциями, которые обновляют эту строку на основе ее первоначального значения.

Ни у одной из транзакций нет сведений о действиях, выполненных другими транзакциями. Поэтому последнее обновление записывается поверх обновлений, сделанных другими транзакциями, что приводит к потере данных.

Например, два редактора сделали копии одного и того же документа. Затем внесли независимые изменения в свои копии и сохранили их поверх исходного документа. При этом редактор, сохранивший измененную копию последним, записал свои изменения без учета изменений первого редактора, в результате какие-то данные оказались потеряны.

Подобной проблемы удастся избежать, если запретить второму редактору изменять документ, пока с ним не закончит работу первый редактор.

Транзакция 1	Транзакция 2
SELECT f2 FROM tbl1 WHERE f1=1; **(f2=10)**	SELECT f2 FROM tbl1 WHERE f1=1; **(f2=10)**
UPDATE tbl1 SET f2=20 WHERE f1=1;	**UPDATE tbl1 SET f2=25 WHERE f1=1;**

Чтение незафиксированных данных (“грязное” чтение)

Чтение незафиксированных данных возникает, когда вторая транзакция читает строку, которую в это время обновляет первая транзакция. Таким образом, вторая транзакция читает ещё не зафиксированные данные, которые могут быть изменены первой транзакцией.

Например, редактор вносит изменения в электронный документ. В это время второй редактор делает копию документа со всеми изменениями, уже внесёнными первым редактором, и распространяет его в целевой аудитории.
После этого первый решает, что внесённые изменения не нужны, отменяет их и сохраняет документ. При этом распространённый документ содержит несуществующие, стало быть, недействительные правки.

Подобной проблемы можно избежать, если исключить возможность чтения изменяемого документа, пока первый редактор не признает внесенные изменения окончательными.

Транзакция 1	Транзакция 2
SELECT f2 FROM tbl1 WHERE f1=1	
UPDATE tbl1 SET f2=f2-2 WHERE f1=1	
	SELECT f2 FROM tbl1 WHERE f1=1
ROLLBACK WORK	

В транзакции 1 изменяется значение поля f2, а затем в транзакции 2 выбирается значение этого поля.
После этого происходит откат транзакции 1. В результате значение, полученное второй транзакцией, будет отличаться от значения, хранимого в базе данных

Несогласованный анализ (неповторяемое чтение)

Несогласованным анализом называется ситуация, когда вторая транзакция несколько раз обращается к той же строке, что и первая, однако от раза к разу данные меняются. Несогласованный анализ напоминает чтение незафиксированных данных тем, что первая транзакция изменяет данные, которые читает вторая транзакция. Однако при несогласованном анализе первая транзакция подтверждает изменение этих данных. Кроме того, при несогласованном анализе происходит многократное {двукратное или больше) чтение второй транзакцией одной и той же строки с разными данными.

Например, редактор два раза читает один и тот же документ, а в промежутке между обращениями автор документа переделывает его. Значит, во второй раз редактор читает уже измененный документ.

Этого можно избежать, если редактору разрешить чтение документа лишь в окончательной форме.

Транзакция 1	Транзакция 2
SELECT f2 FROM tbl1 WHERE f1=1	SELECT f2 FROM tbl1 WHERE f1=1
UPDATE tbl1 SET f2=f2+1 WHERE f1=1	
COMMIT	
	SELECT f2 FROM tbl1 WHERE f1=1

:

В транзакции 2 выбирается значение поля f2, затем в транзакции 1 изменяется значение поля f2. При повторной попытке выбора значения из поля f2 в транзакции 2 будет получен другой результат.

Чтение фантомов

Так называется ситуация, когда строки из диапазона строк, читаемого во время транзакции, добавляются или удаляются. Первая операция чтения, исполняемая во время транзакции, показывает наличие в прочитанном диапазоне некоторой строки. Если другая транзакция удалит ее, то эта строка не будет обнаружена, когда диапазон будет прочитан в следующий раз. Аналогично, если другая транзакция добавляет в читаемый диапазон строку после первого чтения, эта строка обнаружится только при повторном чтении.

Транзакция 1	Транзакция 2
	SELECT SUM(f2) FROM tbl1
INSERT INTO tbl1 (f1,f2) VALUES (15,20)	
COMMIT	
	SELECT SUM(f2) FROM tbl1;

В транзакции 2 выполняется SQL-оператор, использующий все значения поля f2. Затем в транзакции 1 выполняется вставка новой строки, приводящая к тому, что повторное выполнение SQL-оператора в транзакции 2 выдаст другой результат. Такая ситуация называется фантомным чтением. От неповторяющегося чтения оно отличается тем, что результат повторного обращения к данным изменился не из-за этих данных, а из-за появления новых (фантомных) данных.

```
UPDATE Employees
SET HourlyRate = 6.75
WHERE HourlyRate < 6.75
ALTER TABLE Employees
   ADD ckWage CHECK (HourlyRate >= 6.75)
```

Типы конфликтов между двумя параллельными транзакциями:

· **W-W** – транзакция 2 пытается изменить объект, измененный не закончившейся транзакцией 1;

· **R-W** – транзакция 2 пытается изменить объект, прочитанный не закончившейся транзакцией 1;

· **W-R** транзакция 2 пытается читать объект, измененный не закончившейся транзакцией 1.

Блокировки позволяют предотвратить применение к объекту базы данных таких операций, которые конфликтуют с операциями, уже выполняемыми над этим объектом. При этом учитывается то, какая из операций, связанных с доступом к объекту, началась раньше. Действия, которые могут и не могут быть выполнены в операциях, начавшихся позже, зависят от того, какие действия осуществляются в операции, начавшейся раньше.

37. Определение блокировки. Управление блокировками в SQL Server. Оптимистические и пессимистические стратегии блокировки. Блокируемые ресурсы.

Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных.

Управлением блокировками на сервере занимается **менеджер блокировок**, контролирующий их применение и разрешение конфликтов.

Транзакции и блокировки тесно связаны друг с другом.

Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID.

Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

SQL Server поддерживает как **оптимистическое,** так и **пессимистическое** управление параллельным выполнением.

По умолчанию в SQL Server применяется пессимистическое управление параллельным выполнением.

Оптимистическое параллельное выполнение

При оптимистическом управлении параллельным выполнением предполагается, что конфликты из-за ресурсов между несколькими пользователями маловероятны.

При оптимистическом управлении параллелизмом **данные на период чтения не блокируются.**

Когда пользователь обновляет данные, система проверяет, вносил ли другой пользователь в них изменение после считывания. Если другой пользователь изменял данные, возникает ошибка. Как правило, при получении сообщения об ошибке пользователь откатывает транзакцию и начинает ее заново.

В основном применяется в средах с небольшим количеством конфликтов данных, где затраты на периодический откат транзакции меньше затрат на блокировку данных при считывании.

Пессимистическое параллельное выполнение

Пессимистическое управление параллельным выполнением блокирует ресурсы, затребованные транзакцией, чтобы обеспечить ее устойчивость. Таким образом, гарантируется успешное завершение транзакции, если не возникнет взаимоблокировки.

В основном применяется в средах с большим количеством конфликтов данных, где затраты на защиту данных с помощью блокировок меньше затрат на откат транзакций в случае конфликтов параллелизма.

38. Уровни изоляции транзакций. Сериализуемые транзакции.

Уровни изоляции «повторяемое чтение», «завершенное чтение», «незавершенное чтение». Особенности обработки блокировок и изоляции транзакций в MS SQL Server.

Уровни изоляции

Проблемы, связанные с параллельным выполнением, разрешают, используя уровни изоляции (ISOLATION LEVEL).

Это позволяет полностью изолировать выполняемые транзакции друг от друга.

Уровни изоляции обеспечивают правила параллелизма и последовательности работы с данными.

Когда устанавливается уровень изоляции, множество пользователей, работающих с одними и теми же наборами данных, устанавливают блокировки или следуют основанным на установленном уровне изоляции правилам.

Уровни изоляции в стандарте SOL-99

Уровень изоляции определяет степень изоляции одной транзакции от остальных.

Более низкий уровень изоляции увеличивает возможность параллельного выполнения, но за это приходится расплачиваться согласованностью данных.

Напротив, более высокий уровень изоляции гарантирует согласованность данных, но при этом страдает параллельное выполнение.

Режим управления блокировками определяется уровнем изоляции, необходимым приложению.

Стандарт SQL-99 определяет следующие уровни изоляции

•неподтверждаемое чтение **READ UNCOMMITTED (**чтение незафиксированных данных)—самый низкий уровень;

READ UNCOMMITTED допускает **грязное чтение, неповторяемое чтение** и **фантомные записи.**

Уровень изоляции **READ UNCOMMITED** обычно является весьма нежелательным и должен быть использован, только когда не требуется точности чтения данных или когда данные изменяются редко.

- подтверждаемое чтение **Read committed** - чтение, при котором отсутствует черновое, "грязное" чтение (т.е. чтение пользователем данных, которые не были зафиксированы в БД командой **COMMIT**).

Но могут изменяться уже прочитанные данные, результатом чего будет **неповторяемое чтение или фантомные данные.**

В Oracle блокировки на чтение нет, вместо этого «читающая» транзакция получает ту версию данных, которая была актуальна в базе до начала «пишущей».

•повторяемое чтение (**Repeatable read**) - Уровень, при котором чтение одной и той же строки или строк в транзакции дает одинаковый результат.

Указывает на то, что транзакциями не могут считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями, а также на то, что другие транзакции не могут изменять данные, читаемые текущей транзакцией, до ее завершения.

Но другие транзакции могут вставлять новые строки, соответствующие условиям поиска, содержащихся в текущей транзакции. При повторном запуске инструкции текущей транзакцией будут извлечены новые строки, что приведет к **фантомному чтению.**

.

•упорядочение (**Serializable**) —наивысший уровень, на котором транзакции полностью изолированы друг от друга.

На этом уровне результаты параллельного выполнения транзакций для базы данных в большинстве случаев можно считать совпадающими с последовательным выполнением тех же транзакций.

- транзакции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.

- Другие транзакции не могут изменять данные, считываемые текущей транзакцией, до ее завершения.

- Другие транзакции не могут вставлять новые строки со значениями ключа, которые входят в диапазон ключей, считываемых инструкциями текущей транзакции, до ее завершения.

Блокировки диапазона сохраняются до завершения транзакции.

Это самый строгий уровень изоляции, поскольку он блокирует целые диапазоны ключей и сохраняет блокировку до завершения транзакции.

Из-за низкого параллелизма этот параметр рекомендуется использовать только при необходимости.

По умолчанию, устанавливается изоляция **Read Committed.**

Основной принцип состоит в том, что пишущая транзакция блокирует данные и читающие транзакции не могут их читать.

Когда установлен **Read Committed,** прочитать можно только сохранённые данные.

Тривиальным решением является действительно последовательное выполнение транзакций.

Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением свойства сериальности.

Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по объектам базы данных.

Блокировки, называемые также синхронизационными захватами объектов, могут быть применены к разному типу объектов.

Наибольшим объектом блокировки может быть вся БД, однако этот вид блокировки сделает БД недоступной для всех приложений, которые работают с данной БД.

Следующий тип объекта блокировки — это **таблицы.** Транзакция, которая работает с таблицей, блокирует ее на все время выполнения транзакции. Этот вид блокировки предпочтительнее предыдущего, потому что позволяет параллельно выполнять транзакции, которые работают с другими таблицами.

В ряде СУБД реализована **блокировка на уровне страниц.** В этом случае СУБД блокирует только отдельные страницы на диске, когда транзакция обращается к ним. Этот вид блокировки ещё более мягок и позволяет разным транзакциям работать даже с одной и той же таблицей, если они обращаются к разным страницам данных.

В некоторых СУБД возможна **блокировка на уровне строк**, однако такой механизм блокировки требует дополнительных затрат на поддержку этого вида блокировки.

В настоящее время проблема блокировок является предметом большого числа исследований.

Уровень изоляции	Фантомная вставка	Неповторяю щееся чтение	«Грязное» чтение	Потерянное обновление[3]
SERIALIZABLE	+	+	+	+
REPEATABLE READ	–	+	+	+
READ COMMITTED	–	–	+	+
READ UNCOMMITTED	–	–	–	+

Настройка блокировок

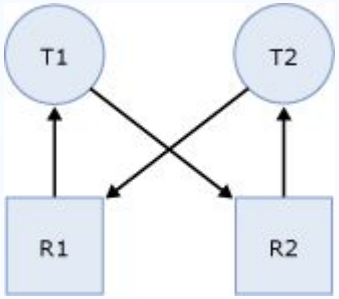
SQL Server реализует блокирование автоматически, но в приложениях эту функцию можно настроить следующими способами:

- задавая обработку взаимоблокировок и установку приоритетов при взаимоблокировках;
- задавая обработку тайм-аутов и определяя продолжительность тайм-аута блокировки;
- устанавливая уровень изоляции транзакции;
- назначая блокировки на уровне таблицы с операторами **SELECT, INSERT, UPDATE и DELETE.**

Обработка взаимоблокировок

Взаимоблокировка возникает, когда два или более потоков, конкурирующих за ресурс, становятся взаимозависимыми.

Взаимоблокировка возникает, когда у каждой транзакции заблокирован ресурс, который пытаются заблокировать другие транзакции.



Транзакция T1 блокирует ресурс R1 и запросила блокировку ресурса R2.

Транзакция T2 блокирует ресурс R2 и запросила блокировку ресурса R1 .

Так как ни одна из задач не может продолжиться до тех пор, пока не будет доступен ресурс, а ни один из ресурсов не может быть освобожден до тех пор, пока задание не продолжится, наступает состояние взаимоблокировки.

Обе транзакции находятся в состоянии взаимоблокировки и будут всегда находиться в состоянии ожидания, если взаимоблокировка не будет разрушена внешним процессом.

Монитор взаимоблокировок периодически проверяет задачи на состояние взаимоблокировки. Если монитор обнаруживает цикличную зависимость, то выбирается одна задача, для которой транзакция будет завершена с ошибкой.

Это позволяет другой задаче завершить свою транзакцию. Позднее приложение может повторно выполнить транзакцию, которая завершилась с ошибкой, обычно после того как другая транзакция (бывшая в состоянии взаимоблокировки) завершится.

Минимизация числа взаимоблокировок

Хотя полностью избежать взаимоблокировок не удаётся, их число можно существенно уменьшить, чтобы увеличить пропускную способность при обработке транзакций и снизить системные издержки, поскольку меньше транзакций будут подвергаться откату с отменой всей выполненной ими работы. Кроме того, уменьшится число транзакций, повторно переданных приложениями на сервер из-за отката в результате взаимоблокировок.

Чтобы минимизировать число взаимоблокировок, необходимо придерживаться следующих правил:

- следить, чтобы транзакции были короткими и не выходили за пределы одного пакета;
- использовать низкие уровни изоляции;

Оператор **SET LOCK_TIMEOUT** позволяет задать максимальный период ожидания оператором освобождения ресурса.

Если ожидание оператора продлилось дольше установленного LOCK_TIMEOUT, оператор автоматически отменяется и приложению возвращается сообщение об ошибке 1222 Lock request time-out period exceeded (Превышение тайм-аута при ожидании блокировки).

В приложении должен быть обработчик ошибок, перехватывающий сообщение об ошибке 1222.

Реализация обработчика ошибок, перехватывающего сообщение об ошибке 1222, позволяет приложению обработать тайм-аут и предпринять меры для решения проблемы (например, выполнить откат всей транзакции).

Основными режимами блокировок являются следующие:

- **совместный режим** – S (Shared), означающий совместную (**по чтению**) блокировку объекта и требуемый для выполнения операции чтения объекта;

- **монопольный режим** – X (eXclusive), означающий монопольную (**по записи**) блокировку объекта и требуемый для выполнения операций вставки, удаления и модификации объекта.

Настройка уровней изоляции транзакции

По умолчанию SQL Server работает на уровне изоляции **READ COMMITTED** (подтвержденное чтение).

Однако для работы приложения иногда требуется другой уровень изоляции. Чтобы использовать в приложениях более или менее строгие уровни изоляции, следует настроить блокировку для текущего сеанса, установив уровень изоляции сеанса с помощью оператора
SET TRANSACTION ISOLATION LEVEL
Например:
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

В некоторых СУБД возможна блокировка на уровне строк. MS SQL Server стремится установить блокировку на уровне записей, чтобы обеспечить максимальную параллельность в работе.

С увеличением количества блокировок строк сервер может перейти к блокировке страниц, если количество записей превышает пороговое значения.

Реализация указаний блокирования на уровне таблицы

Для операторов **SELECT, INSERT, UPDATE и DELETE** разрешается задавать ряд указаний блокирования таблицы, определяющих тип используемой блокировки.

Указания блокирования на уровне таблицы служат для тонкой настройки типов блокировок, необходимых для объекта.

Указания блокирования подменяют текущий уровень изоляции транзакций сеанса.

NOLOCK - можно читать неподтверждённые транзакции.

PAGLOCK - блокировка страницы.

ROWLOCK - блокировка на уровне строк вместо более грубых блокировок уровня страницы и таблицы.

TABLOCK - блокировка таблиц вместо более тонких блокировок уровня строк и страниц.

В следующем примере установлен уровень изоляции транзакций **SERIALIZABLE**, а также используется указание блокировки на уровне таблицы с оператором **SELECT**:

```
USE PROPERTY
GO
SET TRANSACTION ISOLATION LEVEL READ COMMITED
GO
BEGIN TRANSACTION
SELECT PROPERTY_NO FROM PROPERTY (tabLOCK)
GO
...
```

- При одновременном обращении нескольких пользователей к БД, когда блокирование не применяется, весьма вероятно возникновение ошибки, если их транзакции одновременно используют одни и те же данные.

- Блокировки запрещают пользователям читать данные, изменяемые другими пользователями, а также не дают изменять одни и те же данные нескольким пользователям одновременно.

- Проблемы параллельного выполнения возникают из-за **потерянных обновлений, несогласованного анализа (неповторяемые данные) и чтения фантомов.**

- Возможно **оптимистическое и пессимистическое** управление параллельным выполнением. При оптимистическом управлении параллельным выполнением предполагается, что конфликты нескольких пользователей из-за ресурсов маловероятны. Этот режим позволяет исполнять транзакции без блокирования ресурсов.

- Пессимистическое управление параллельным выполнением блокирует ресурсы на время исполнения транзакции. Таким образом, обеспечивается успешное завершение транзакции, если не возникает взаимоблокировка.

- Уровень изоляции —это **степень**, в которой одна транзакция должна быть изолирована от других.

- SQL Server поддерживает следующие уровни изоляции: **неподтверждаемое чтение, подтверждаемое чтение, повторяемое чтение и упорядочение.**
- Хотя SQL Server реализует блокирование автоматически, его можно настроить, используя обработку взаимоблокировок, установку приоритета при взаимоблокировке, обработку тайм-аутов и определение длины тайм-аута блокировки, а также устанавливая уровни изоляции с помощью указаний блокирования на уровне таблицы и настройки блокирования.

Программирование эффективных транзакций

- Следует создавать как можно более короткие транзакции.

- При модификации данных изменяемые строки следует защитить монопольными блокировками, не допускающими чтения этих строк другими транзакциями, причем необходимо, чтобы блокировки удерживались до фиксации транзакции или ее отката.

Медленные, малоэффективные транзакции не создают особых проблем в системах с небольшим числом пользователей, но в системах с тысячами пользователей они недопустимы.