

Транзакции

Понятие транзакции

Транзакция – это последовательность операций, проводимых над БД, выполняемых как единое целое и переводящих БД из одного непротиворечивого состояния в другое непротиворечивое состояние

Количество операций, входящих в транзакцию, может быть любым от одной до сотен, тысяч

Разработчик решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций.

При выполнении транзакции СУБД должна обеспечить обработку набора команд, входящих в транзакцию, так, чтобы гарантировать правильность и надежность работы системы.

Транзакция должна удовлетворять ACID – требованиям

ACID – требования

ACID – требования гарантируют правильность и надежность работы системы

Atomic
(атомарность)

Транзакция не может выполняться частично, либо все, либо ничего

Consistency
(согласованность)

После выполнения транзакции все данные должны находиться в согласованном состоянии

Isolation
(изолированность)

Транзакция должна быть автономной и воздействовать на другие транзакции или зависить от них

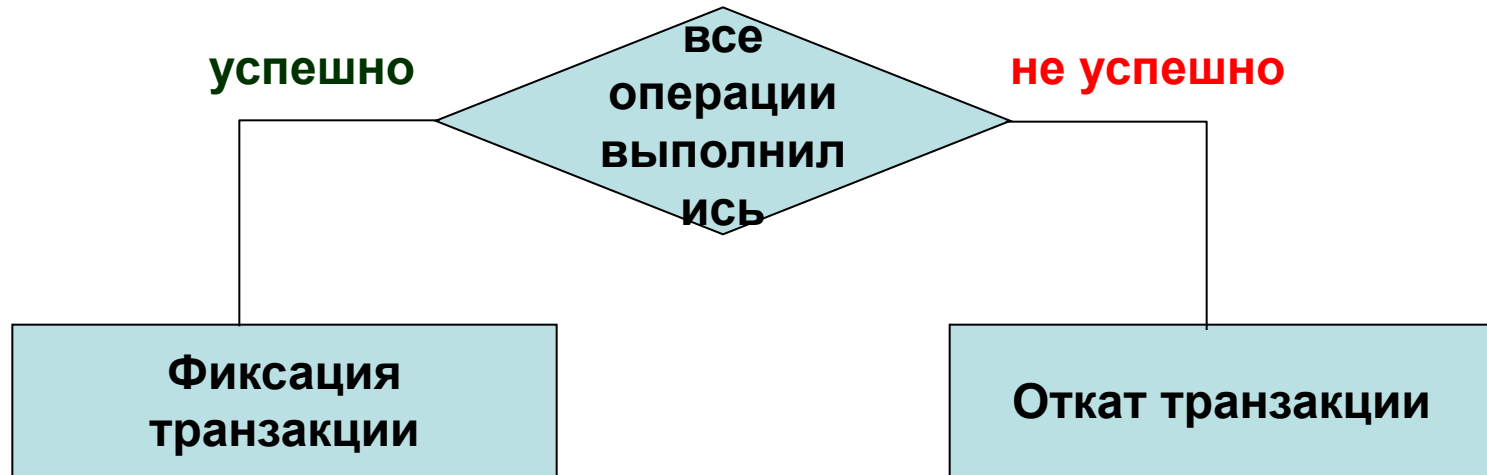
Durability
(устойчивость)

После завершения транзакции, внесенные изменения останутся неизменными

ACID - фундаментальные свойства систем обработки транзакций

Варианты завершения транзакций

2 варианта завершения транзакций



Фиксация транзакции – это действия, обеспечивающие сохранение на диске изменений БД, сделанные в процессе выполнения транзакции

Откат транзакции – это действия, обеспечивающие аннулирование всех изменений БД, сделанные в процессе выполнения транзакции

Виды определения транзакций

(определенные стандартом)

Явное

Требуется явно указать команды начала и конца транзакций

Автоматическое

Каждая команда рассматривается как отдельная транзакция

Неявное

Транзакция начинается с первого оператора SQL и заканчивается явным указанием конца транзакции

SQL Server поддерживает все три вида

Явные транзакции

Описываются командами T-SQL

Начало транзакции

BEGIN TRAN[SACTION] [transaction_name | var_transaction_name]

Имя транзакции используется для вложенных транзакций

Фиксация транзакции

COMMIT [TRAN[SACTION]] [transaction_name | var_transaction_name]

Сохранение точки отката транзакции

SAVE TRAN[SACTION]] [savepoint_name | var_savepoint_name]

Откат транзакции

ROLLBACK [TRAN[SACTION]] [transaction_name | var_transaction_name
savepoint_name | var_savepoint_name]

! При вложении транзакций *transaction_name* должно быть имя из самой внешней инструкции BEGIN TRANSACTION.

Пример 1 явной транзакции

```
CREATE PROC ДобавитьЗаказКолТовар
```

```
@КодЗак INT, @КодТов INT, @ДопКол INT
```

```
AS
```

```
DECLARE @Состояние VARCHAR(10), @Остаток INT, @Цена MONEY
```

```
SELECT @Состояние = Состояние FROM Заказы WHERE ЗаказID = @КодЗак
```

```
IF @Состояние IS NOT NULL AND @Состояние <> 'отгружен'
```

```
BEGIN
```

```
SELECT @Остаток = Остаток, @Цена = ЦенаОтпускная FROM Склад  
WHERE СкладID = @КодТов
```

```
IF @Остаток >= @ДопКол
```

```
BEGIN
```

```
BEGIN TRAN
```

```
UPDATE Склад SET Остаток = Остаток - @ДопКол WHERE СкладID = @КодТов
```

```
UPDATE ЗаказаноТоваров SET Количество = Количество + @ДопКол  
WHERE ЗаказID = @КодЗак AND СкладID = @КодТов
```

```
UPDATE Заказы SET ОбщаяСумма = ОбщаяСумма + @ДопКол * @Цена  
WHERE ЗаказID = @КодЗак
```

```
COMMIT
```

```
RETURN 0
```

```
END
```

```
ELSE
```

```
RETURN 1
```

```
ELSE
```

```
RETURN 2
```

Пример 2 явной транзакции

```
CREATE proc АннулированиеЗаказа
```

```
@КодЗаказа int
```

```
AS
```

```
if exists (select * from Заказы where ЗаказID=@КодЗаказа  
and Состояние = 'оформление')
```

```
begin
```

```
Begin tran
```

```
-- Возврат кол.товаров в табл. «Склад»
```

```
Update Склад set Остаток = Остаток + Количество
```

```
from ЗаказаноТоваров
```

```
where ЗаказаноТоваров.ЗаказID = @КодЗаказа and
```

```
ЗаказаноТоваров.СкладID = Склад.СкладID
```

```
-- Удаление заказанных товаров из табл. "ЗаказаноТоваров" для данного заказа
```

```
delete from ЗаказаноТоваров where ЗаказID=@КодЗаказа
```

```
-- Удаление заказа из табл. "Заказы"
```

```
delete from Заказы where ЗаказID=@КодЗаказа
```

```
commit tran
```

```
end
```


Неявные транзакции

Начало транзакции

начинается автоматически при исполнении любого из перечисленных операторов

ALTER TABLE, **CREATE** объекта БД, **DROP** объекта БД,
SELECT,
INSERT, **DELETE**, **UPDATE**,
OPEN, **FETCH**,
GRANT, **REVOKE**,
TRUNCATE TABLE.

Фиксация транзакции

COMMIT [TRAN[SACTION]] [transaction_name | var_transaction_name]

Откат транзакции

ROLLBACK [TRAN[SACTION]] [transaction_name | var_transaction_name
savepoint_name | var_savepoint_name]

Пример неявных транзакций

```
USE pubs
```

```
/* первая автоматическая транзакция
```

```
CREATE table t1
```

```
( A int Primary key,  
  B char(30) Not null )
```

```
/* вторая автоматическая транзакция
```

```
SELECT * FROM t1
```

```
SET IMPLICIT_TRANSACTIONS ON
```

```
/* первая неявная транзакция
```

```
INSERT INTO t1 VALUES (1, 'AAA')
```

```
INSERT INTO t2 VALUES (....)
```

```
COMMIT TRAN
```

```
/* вторая неявная транзакция
```

```
INSERT INTO t1 VALUES (2, 'BBB')
```

```
INSERT INTO t3 VALUES (....)
```

```
SELECT * FROM t4
```

```
COMMIT TRAN
```

```
SET IMPLICIT_TRANSACTIONS OFF
```

Переключение режима транзакций на неявные

Переключение режима транзакций на автоматические

Откат и фиксация транзакций в триггерах



Триггер работает так, как если бы при его выполнении имелась необработанная транзакция.

Поэтому **COMMIT** завершит внешнюю транзакцию

Если в триггере имеется **BEGIN TRANSACTION**, то создается вложенная транзакция и **COMMIT TRANSACTION** будет применяться только к вложенной транзакции.



ROLLBACK TRANSACTION в триггере:

- отменяет все изменения данных, уже выполненные в текущей транзакции, в том числе изменения, выполненные триггером;
- все оставшиеся инструкции после инструкции **ROLLBACK** продолжают выполняться;
- текущий пакет снимается с выполнения и, для версий 2005 и выше, сгенерируется ошибка 3609
- закрывает и освобождает все курсоры, которые были объявлены и открыты в пакете, содержащем инструкцию, приведшую к срабатыванию триггера.

Чтобы выполнить откат транзакций только в триггере, нужно использовать **SAVE TRANSACTION**.

Пример отката транзакций в триггере

```
CREATE TRIGGER Add_Клиенты
ON Клиенты
FOR INSERT
AS
PRINT 'Выполнение триггера';
DECLARE @КлиентID int, @ОрганизID int

SELECT @КлиентID=ОрганизацииID FROM INSERTED

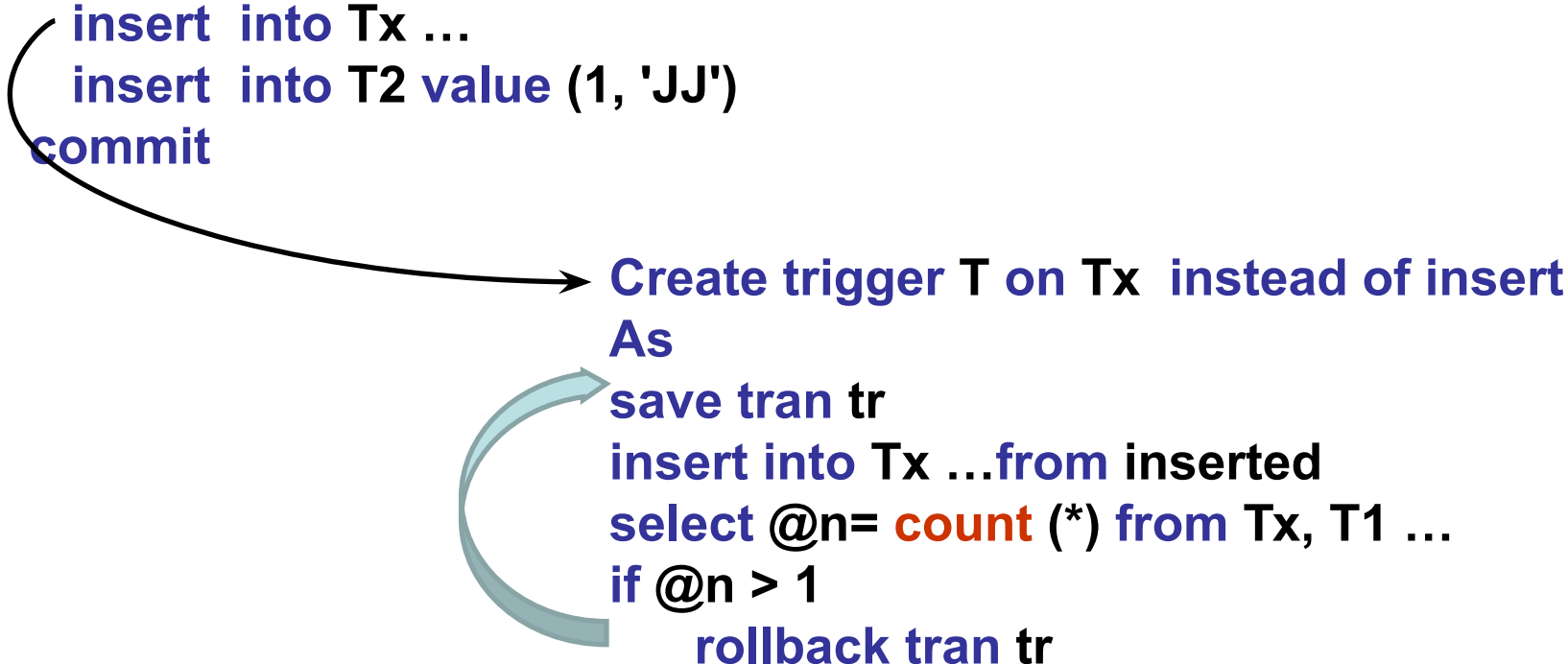
SELECT @ОрганизID=ОрганизацииID FROM Организации
WHERE ОрганизацииID=@КлиентID

IF @ОрганизID IS NULL
BEGIN
    PRINT 'нет организации'
    -- отменить вставку записи
    ROLLBACK TRAN
END
ELSE
BEGIN
    PRINT 'Клиент вставлен'
    COMMIT TRAN
END
```

Откат и фиксация транзакций в триггерах

Чтобы выполнить откат транзакций только в триггере, нужно использовать `SAVE TRANSACTION`.

```
begin tran
  insert into T1 value (1, 'A')
  insert into Tx ...
  insert into T2 value (1, 'JJ')
commit
```



```
Create trigger T on Tx instead of insert
As
save tran tr
insert into Tx ...from inserted
select @n= count (*) from Tx, T1 ...
if @n > 1
  rollback tran tr
```

Результат: будут вставлены строки в T1 и T2, в Tx будет вставлена, если условие в триггере не выполнится.

Журнал транзакций

Это системная структура, обеспечивающая восстановление состояния БД

Восстановление состояния БД требуется

1) при откатах транзакций

- явно (оператор ROLLBACK)
- при аварийном завершении клиентского приложения
- принудительный откат при взаимной блокировке

2) при внезапной потере данных в ОП

- при отключении электропитания
- сбои процессора

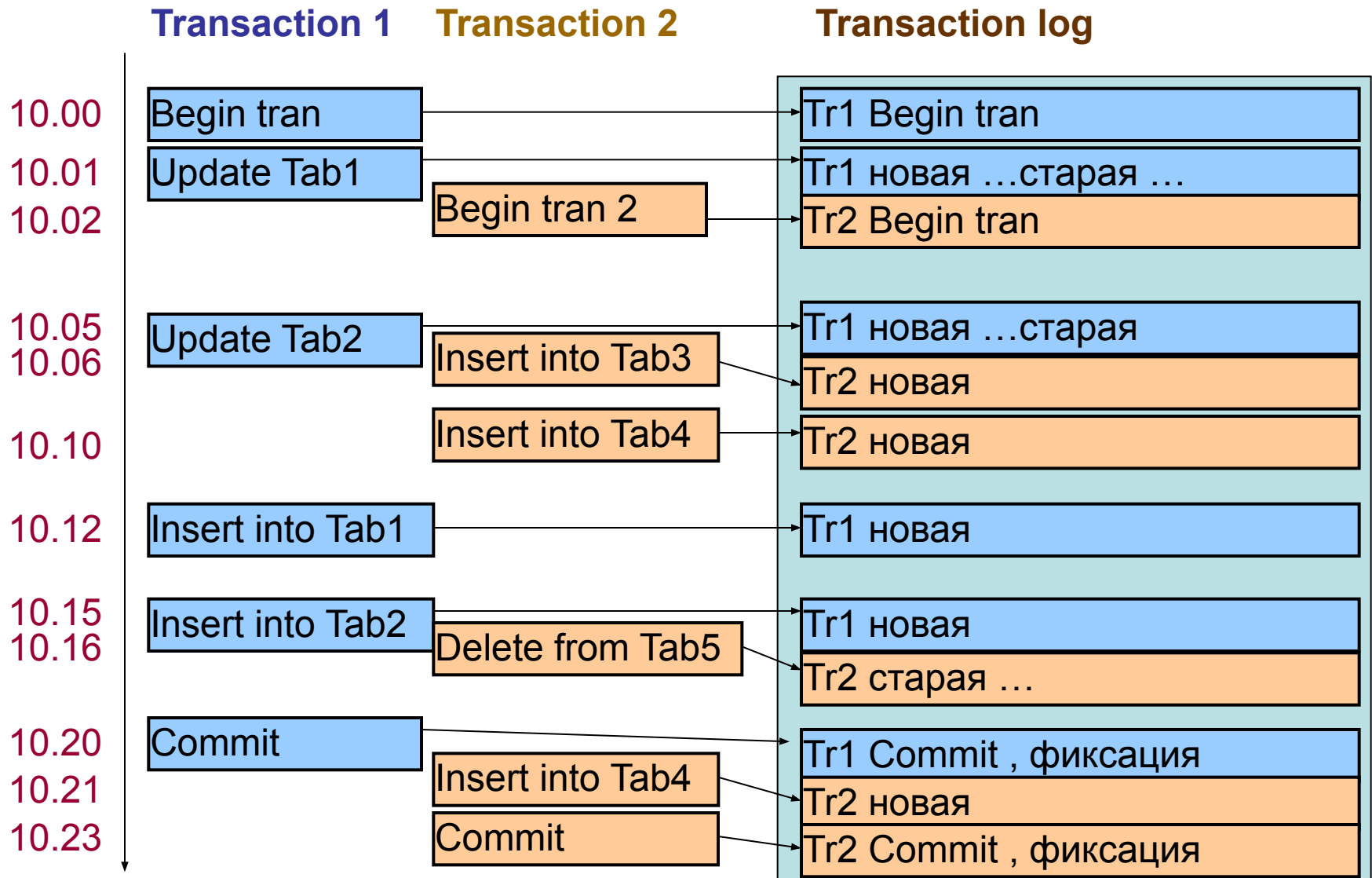
3) при отказе HD

Логическая структура журнала транзакций

- это последовательность записей, содержащих следующую информацию

Порядковый номер	Log Sequence Number (LSN)- последовательно увеличивающееся значение
Идентификатор транзакции	Маркирует транзакцию с учетом пользователя
Операция	Выполняемые команды, в т.ч. завершения транзакции
Атрибут	Имя таблицы, имя поля и т.п.
Новое значение атрибута	
Старое значение атрибута	

Пример ведения журнала транзакций



Журнал транзакций

Общими принципами восстановления состояния БД являются

Результаты **зафиксированных транзакция** должны быть **сохранены** в восстановленном состоянии БД

Результаты **незафиксированных транзакция** должны быть **отсутствовать** в восстановленном состоянии БД

Модели ведения журнала транзакций

- протокол с отложенными изменениями
- протокол с немедленными изменениями

Протокол с отложенными изменениями

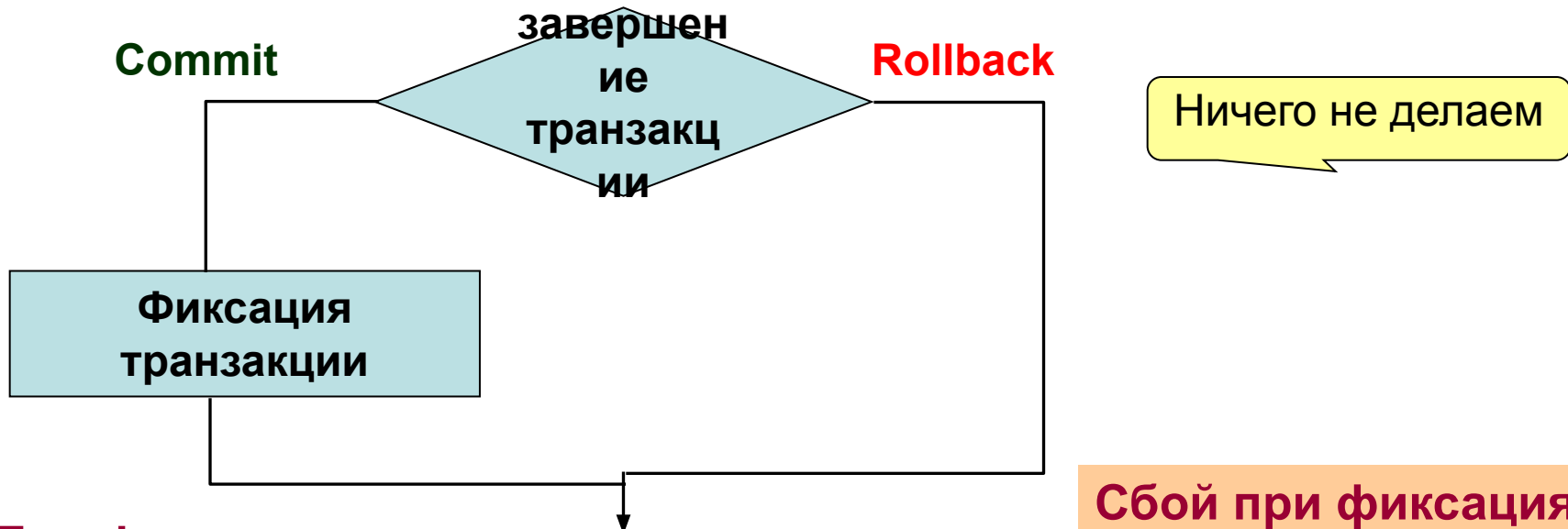
- предполагает внесение изменений, которые должны быть сделаны в БД, только в журнал транзакций

Окончание выполнения транзакции состоит из 2-х состояний:

- завершение
- фиксация

Протокол с отложенными изменениями

Завершение – это конец выполнения транзакции (с фиксированием или отменой)



При фиксации транзакции выполняется процедура **REDO()**.

Процедура **REDO()** переписывает результаты транзакции в БД, проходя по протоколу начиная с первой команды транзакции

Сбой при фиксации транзакции

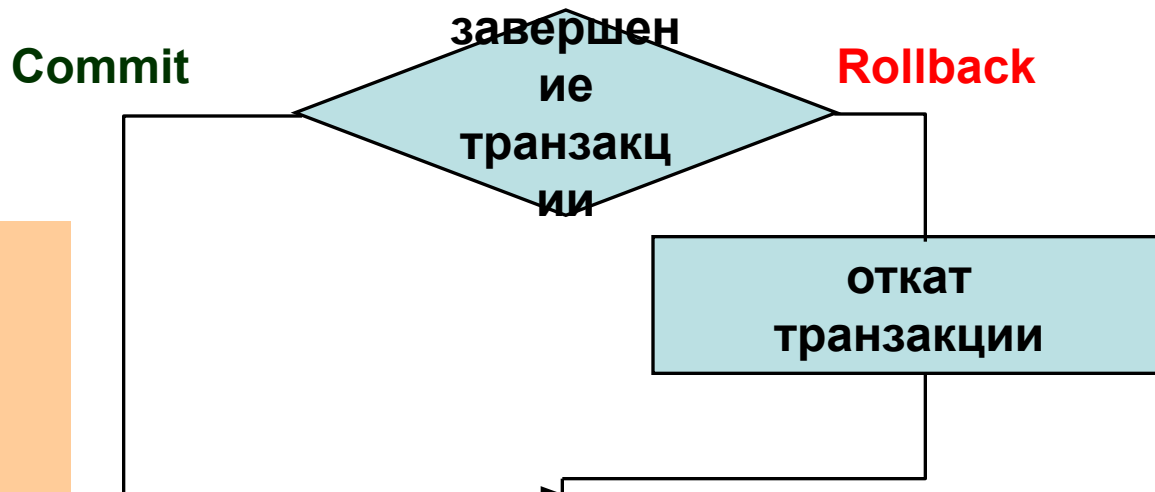
–восстанавливается работоспособность сервера

- выполняется процедура **REDO()**

Протокол с немедленными изменениями

- предполагает внесение изменений и БД и в журнал транзакций.

Ничего не делаем



Сбой при выполнении транзакции

–восстанавливается работоспособность сервера

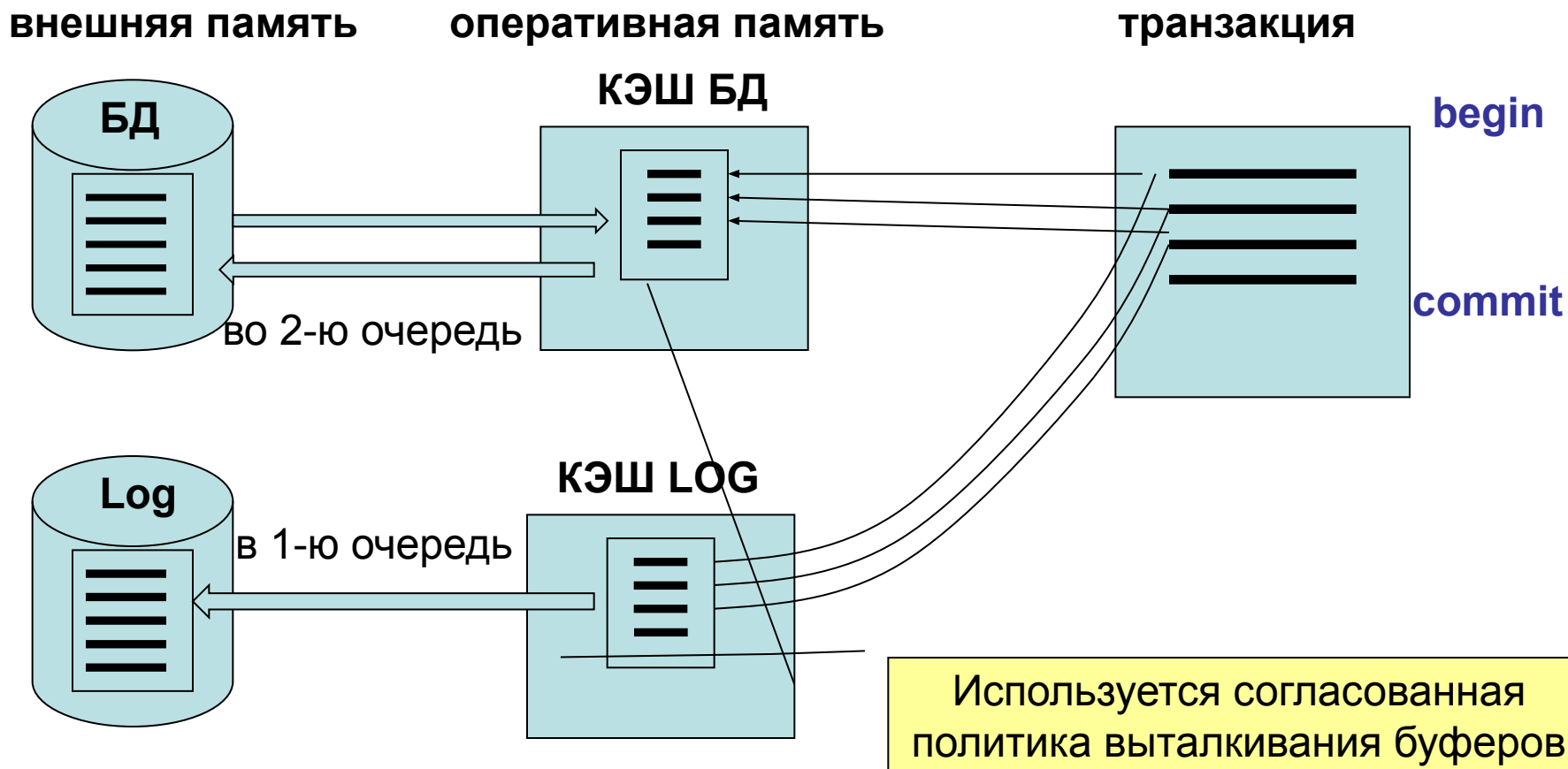
- если есть **Begin**, но нет **Commit**, выполняется **UNDO()**

- если есть **Begin Commit**, то выполняется фиксация в журнале

При откате транзакции выполняется процедура **UNDO()**

UNDO() возвращает все старые значения в БД, выполняя по журналу, начиная с последней команды отмененной транзакции, обратные команды.

Протокол с немедленными изменениями



В КЭШ БД считываются требуемые транзакцией страницы и все изменения происходят в КЭШ, а не на диске.

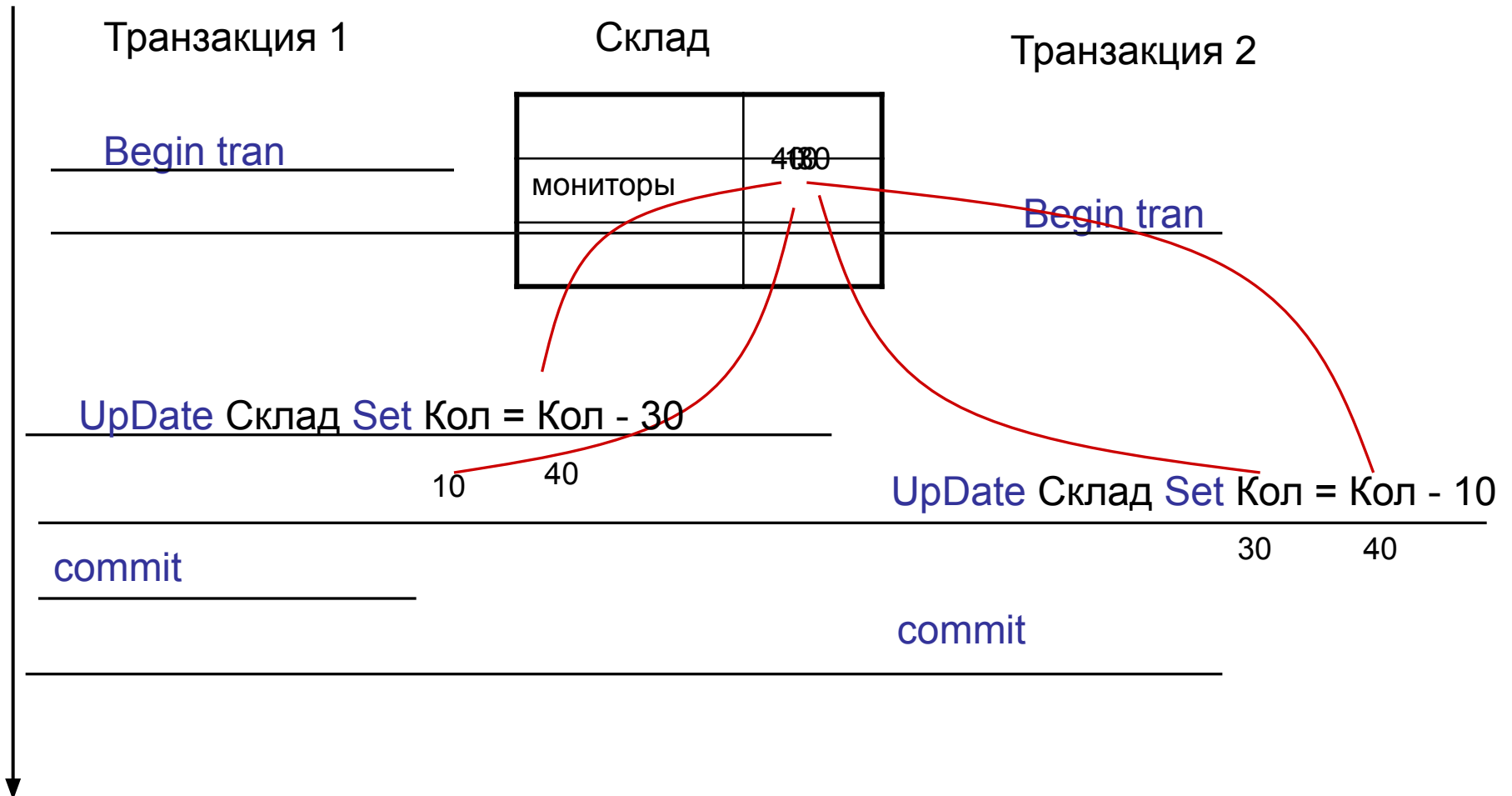
Параллельное выполнение транзакций

Параллельное выполнение нескольких транзакций может привести к следующим проблемам одновременного доступа к БД:

1. Проблемы пропавших обновлений
2. Проблемы промежуточного чтения
3. Проблемы несогласованных данных
4. Проблемы чтения фантомов

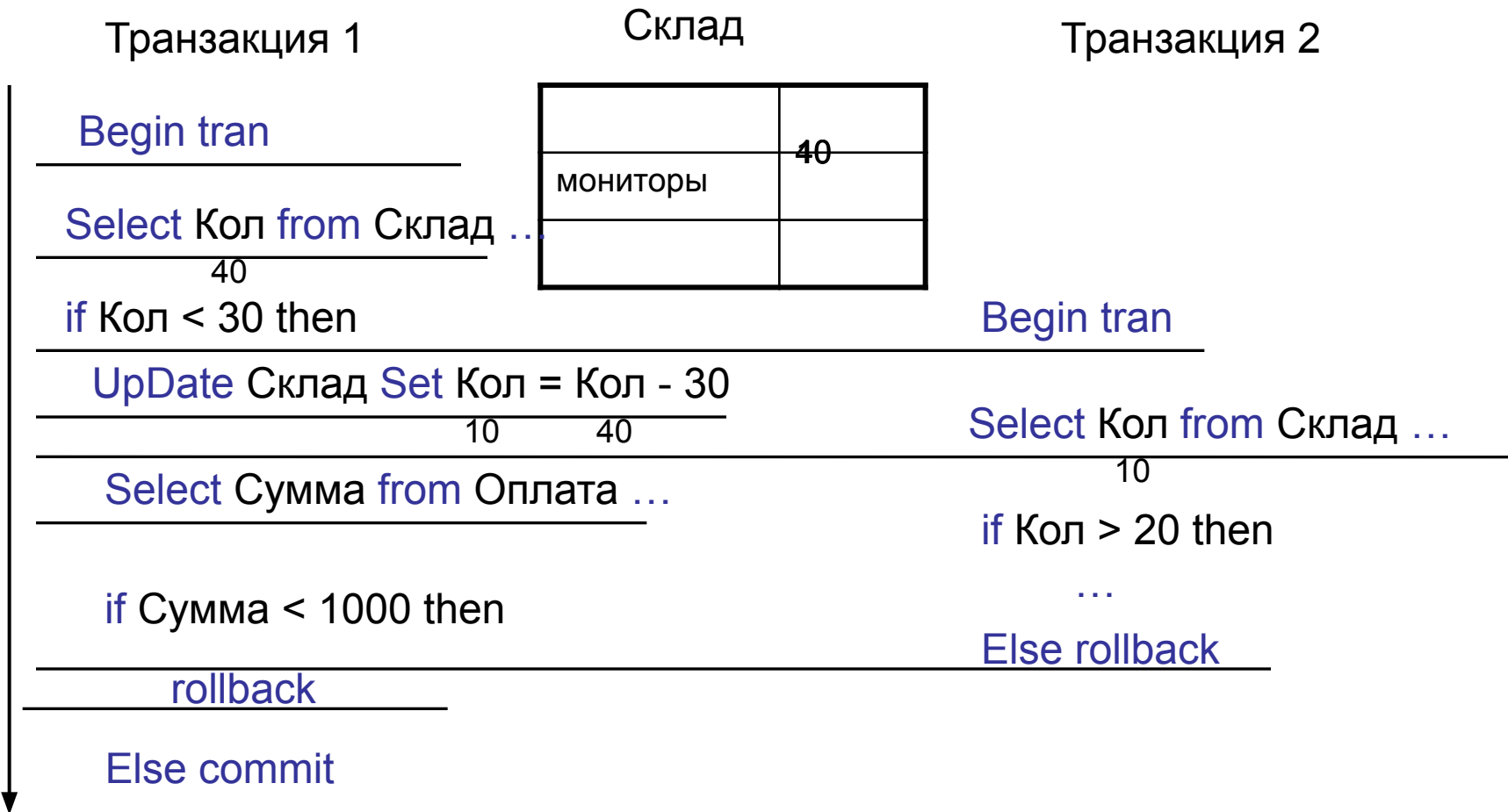
Проблема пропавших обновлений

Возникает когда несколько транзакций изменяют одну и тоже строку, основываясь на её начальном значении



Проблема промежуточного чтения

Возникает когда при выполнении одной транзакции другая использует её промежуточные данные



Проблема несогласованных данных

Возникает когда транзакция считывает одни и те же данные несколько раз, а другая транзакция вносит в эти данные изменения.

Транзакция 1

Склад

Транзакция 2

мониторы	40	200

Begin tran

Select Цена, Кол from Склад ...

200

if Кол > 30 then

Select Сумма from Оплата ...

if Сумма > 30*Цена then

Update Склад Set Кол = Кол - 30

10 40

Update Оплата Set Сумма = Сумма - 30*Цена

200

Select Кол, Цена from Склад ...

220

Insert into Накладные Values (Кол, Цена, Кол*Цена)

commit

Begin tran

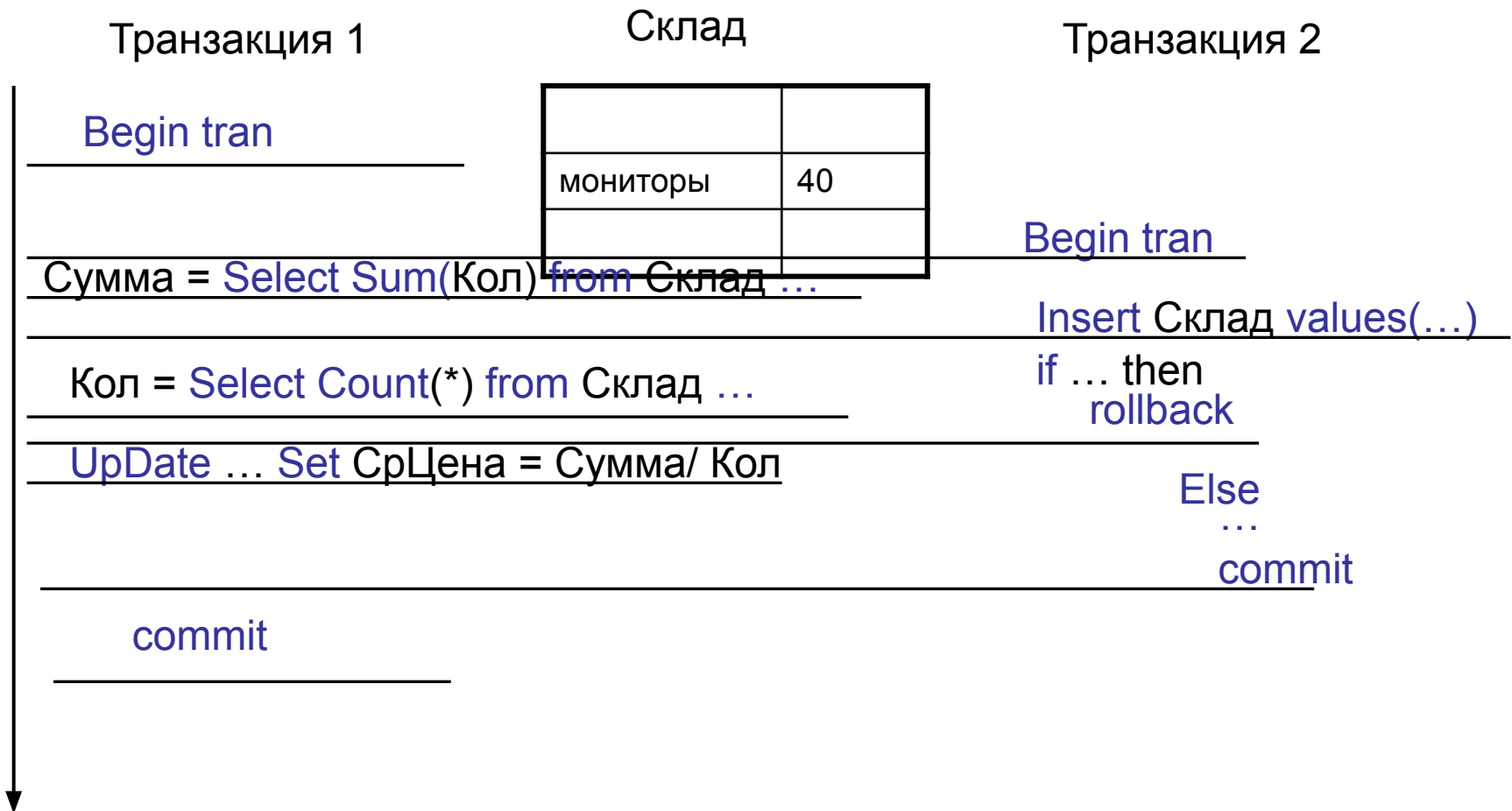
Update Склад Set Цена=220 ...

commit



Проблема чтения фантомов

Возникает когда одна транзакция выбирает данные из таблицы несколько раз, а другая транзакция вставляет новые строки до завершения первой.



Уровни изоляции транзакций

- определяют степень зависимости транзакций друг от друга.

Это способ решения проблем одновременного доступа
Стандартом ANSI SQL-92 определены 4 уровня изоляции транзакций:

0 –й уровень **READ UNCOMMITTED** (незавершенное чтение)

1 –й уровень **READ COMMITTED** (завершенное чтение)

2 –й уровень **REPEATABLE READ** (несогласованные данные)

3 –й уровень **SERIALIZABLE** (сериализуемость)

Каждый последующий уровень изоляции соответствует требованиям всех предыдущих уровней и обеспечивает дополнительную защиту транзакций.

В **SQL Server 2008** есть **SNAPSHOT** (Моментальный срез)— транзакция, в которой требуется чтение не ждёт завершения транзакции изменяющей данные, а считывает их версию, по состоянию на момент начала этой транзакции. Не входит в стандарта SQL 92.

Уровни изоляции транзакций

Уровни изоляции и решаемые им проблемы одновременного доступа

Уровни изоляции	Проблемы одновременного доступа			
	Пропавшие обновления	Промежуточное чтение	Несогласованных данных	Строки - признаки
SERIALIZABLE	X	X	X	X
REPEATABLE READ	X	X	X	
READ COMMITTED	X	X		
READ UNCOMMITTED	X			

В MS SQL SERVER 2008 по умолчанию установлен **READ COMMITTED**

В Oracle поддерживаются **READ COMMITTED** и **SERIALIZABLE**

Уровни изоляции транзакций

Уровни изоляции транзакций могут быть установлены с использованием **Transact-SQL** или через **API** доступа к СУБД:

- Transact-SQL

Используется инструкция **SET TRANSACTION ISOLATION LEVEL**

- OLE DB

Перед вызовом **ITransactionLocal::StartTransaction** устанавливается параметр **isoLevel** в значение **ISOLATIONLEVEL_READUNCOMMITTED, ISOLATIONLEVEL_READCOMMITTED, ISOLATIONLEVEL_REPEATABLE_READ, ISOLATIONLEVEL_SNAPSHOT, ISOLATIONLEVEL_SERIALIZABLE**

- ADO.NET

Перед вызовом метода **SqlConnection.BeginTransaction** устанавливается параметр **IsolationLevel** в значение **ReadUncommitted, ReadCommitted, RepeatableRead, Serializable** или **Snapshot**

- ODBC

вызывают функцию **SQLSetConnectAttr** с установленным параметром **Attribute** в значение **SQL_ATTR_TXN_ISOLATION** и параметром **ValuePtr** в значение **SQL_TXN_READ_UNCOMMITTED, SQL_TXN_READ_COMMITTED, ...**

Уровни изоляции транзакций

Команда Transact-SQL установки уровня изоляции транзакции

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED |  
    READ COMMITTED |  
    REPEATABLE READ |  
    SERIALIZABLE |  
                                     SNAPSHOT  
    }
```

Установленный уровень изоляции действует на протяжении всего сеанса подключения или до явной замены на другой уровень

Текущий уровень изоляции транзакции можно получить консольной командой

DBCC USEROPTIONS

Уровни изоляции транзакций

Механизмы обеспечения уровней изоляции транзакций:

- блокировки

Суть блокировки - синхронизационных захватах объектов БД запрещение доступа к объекту из других транзакций, пока текущая транзакция выполняется.

- временные метки (версионность)

Суть **временных меток** - одновременно разные транзакции могут видеть разные версии данных (в SQL Server копия первоначальных данных используемых текущей транзакцией с номером операции сохраняется в системной БД TempDB; в Oracle старая версия данных сохраняется в сегменте отката).

Многоверсионный подход уменьшает количество блокировок, но подход, основанный на блокировках, обеспечивает более согласованное представление данных.

Блокировки

- способ обеспечения уровней изоляции транзакций

Блокировка - это временно накладываемое ограничение на доступ к объектам БД во время выполнения транзакций

Самый простой вариант блокировки – это блокировка объекта на все время действия транзакции.

В момент начала работы с любым объектом (если он не заблокирован другой транзакцией) он блокируется до окончания заблокировавшей его транзакции.

После окончания транзакции все заблокированные ею объекты разблокируются и становятся доступными другим транзакциям.

Если транзакция обращается к заблокированному объекту, то она остается в состоянии ожидания до момента разблокировки этого объекта, после чего она может продолжать обработку данного объекта.

Блокировки

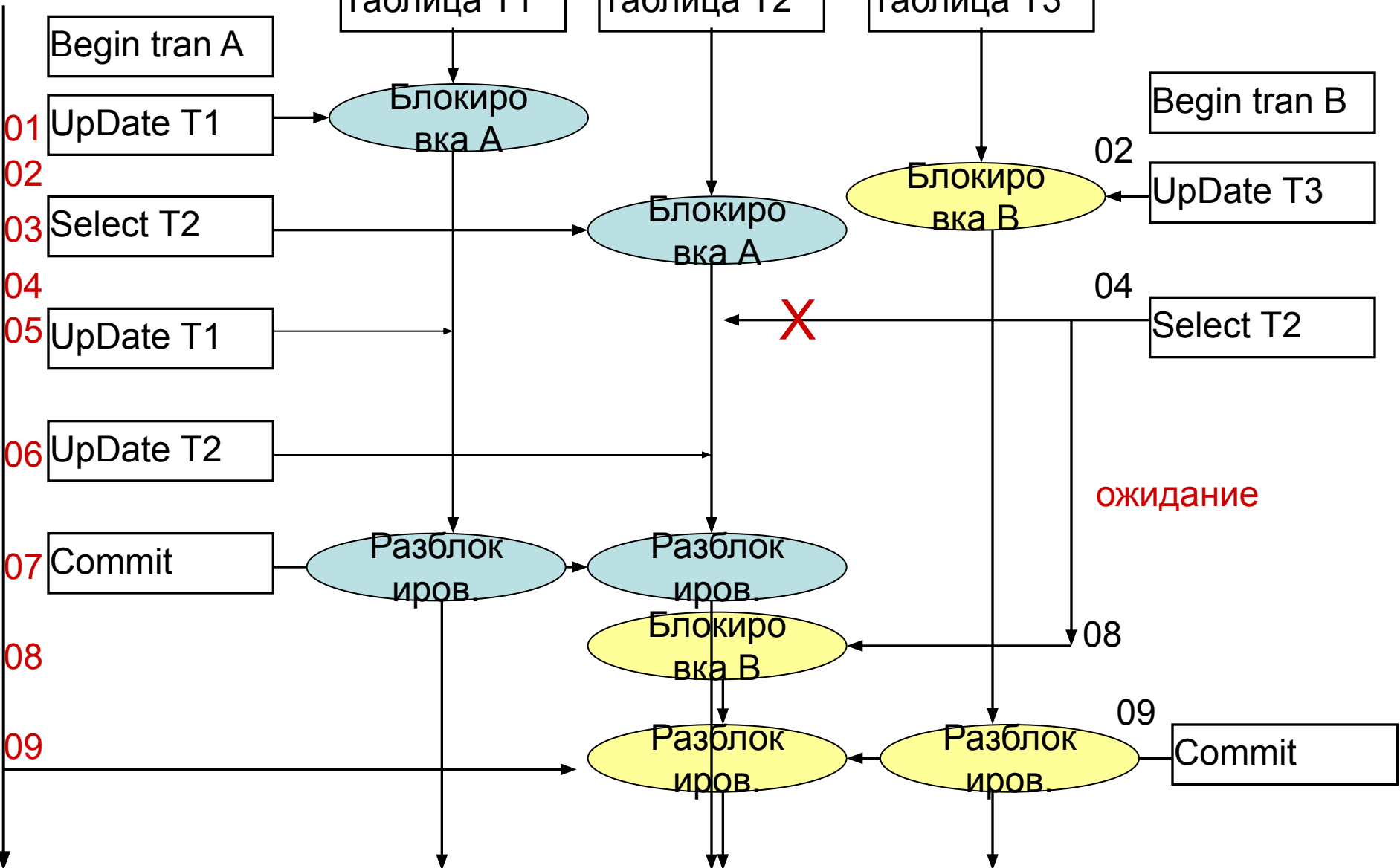
Транзакция А

Транзакция В

Таблица Т1

Таблица Т2

Таблица Т3



Типы и уровни блокировки

Для реализации различных уровней изоляции транзакций используются различные типы и уровни блокировок.

Тип блокировки определяет уровень зависимости соединения от заблокированного объекта

Уровень изоляции транзакций определяет тип блокировки и продолжительность при выполнении команд в транзакции

Различают основные и специальные типы блокировок

Типы и уровни блокировки

Основные типы блокировок

1. Коллективные блокировки (S)

(Shared)

Накладывается при выполнении операций чтения данных. Заблокированные объекты доступны другим транзакциям в режиме чтения

2. Монопольные блокировки (X)

(eXclusive)

Накладывается при выполнении операций изменения данных. Заблокированные объекты не доступны другим транзакциям ни в режиме чтения, ни в режиме изменения.

3. Блокировки обновления (U)

(Update)

Переходная блокировка. Накладывается при установленной коллективной блокировке на объект. Другие транзакции уже не могут установить никакие другие блокировки. После того как будет снята коллективная блокировка эта блокировка будет

Например, для одиночной команды UPDATE требуется сначала произвести чтение данных, а потом их замену. Тогда и подойдет блокировка U.

Блокировки

Транзакция А

Транзакция В

Таблица Т1

Таблица Т2

Таблица Т3

Begin tran A

01 UpDate T1

02 Select T2

03 UpDate T1

04 UpDate T2

05 Commit

Begin tran B

02 UpDate T3

04 Select T2

05 Commit

МОНОПОЛ
ьяная А

КОЛЛЕКТИ
ВН А

МОНОПОЛ
ьяная А

Разблoк
иров.

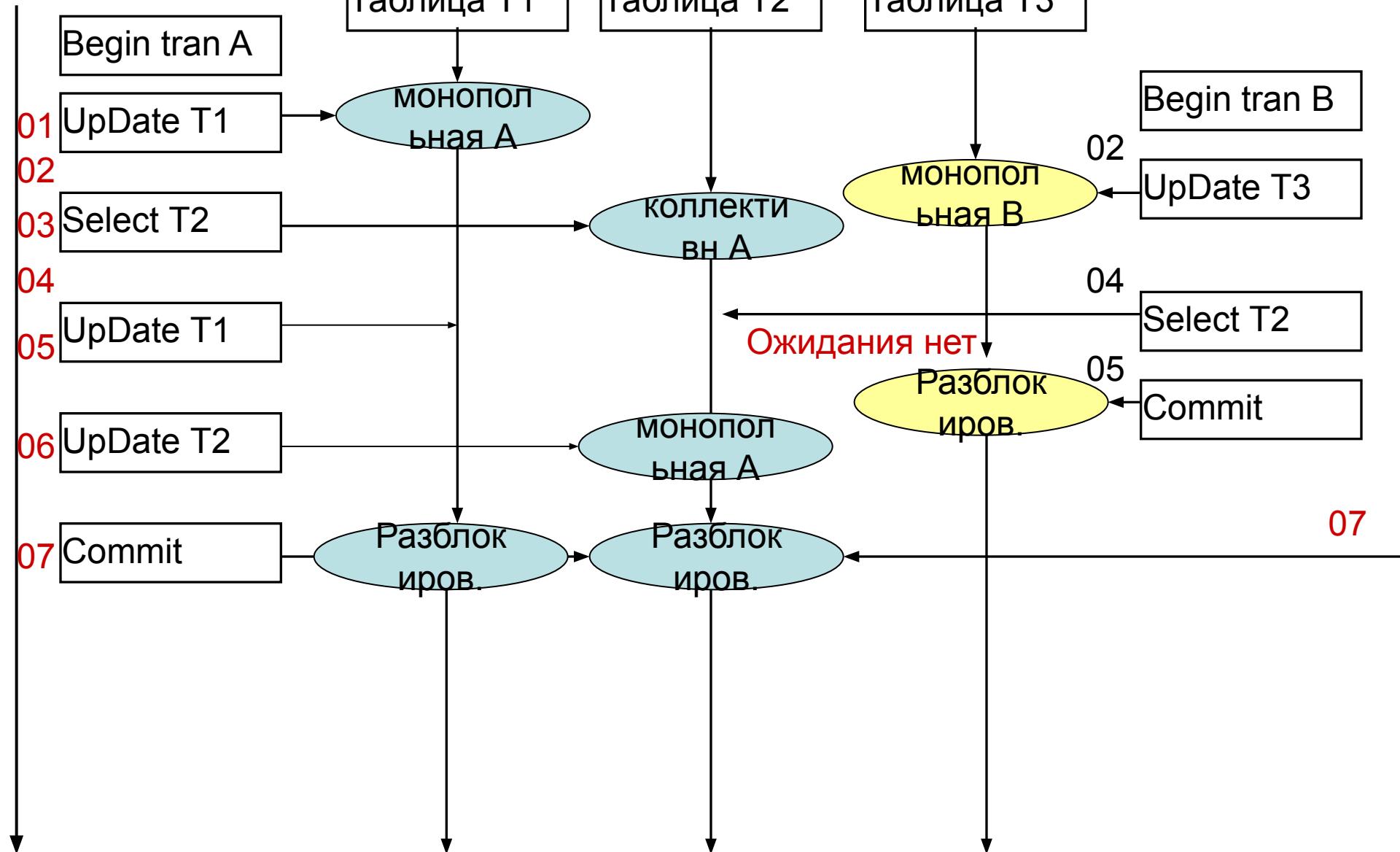
Разблoк
иров.

МОНОПОЛ
ьяная В

Разблoк
иров.

Ожидания нет

07



Типы и уровни блокировки

В SQL Server с версии 2008 имеется ещё другие основные блокировки

4. Блокировка диапазона ключа

Накладывается на диапазон строк, удовлетворяющих определенному условию. Решается проблема возникновения фантомов.

5. Блокировка стабильности схемы (Sch-S) (Stability Lock)

Накладывается на схему объекта, если любая транзакция установила блокировку любого типа (запрещается изменять схему, когда над таблицей производятся действия). Когда все блокировки снимаются, то автоматически снимается эта блокировка.

6. Блокировка изменения схемы (Sch-M) (Modification Lock)

Накладывается на объект, как только начинается изменение структуры объекта. Никакая другая блокировка не может быть наложена на объект, пока установлена эта блокировка.

Типы и уровни блокировки

Введение типов блокировок ликвидирует проблемы одновременного доступа, но создает другую проблему – задержки выполнения транзакций.

Для решения проблемы потери производительности введены уровни блокировок.

Уровни блокировок реализованы на уровнях иерархии объектов БД:

1. RID – блокировка уровня строки
2. KEY– блокировка уровня индекса (группа строк)
3. PAG– блокировка уровня страницы
4. EXT– блокировка уровня группы страницы
5. TAB– блокировка уровня таблицы
6. DB – блокировка уровня базы данных

Специальные блокировки

Специальные блокировки или блокировки намерения используются для разрешения конфликтов наложения блокировок на различных уровнях

1. Коллективные блокировки намерения (IS) (Intent Shared)

Накладывается при намерении транзакции читать данные вниз по иерархии объекта (например, таблица – группа страниц – страница – строка). Другим транзакциям запрещается устанавливать монопольные блокировки вниз по иерархии.

2. Монопольные блокировки намерения (IX) (Intent eXclusive)

Накладывается при намерении транзакции изменять данные вниз по иерархии объекта. Другим транзакциям запрещается устанавливать любые блокировки вниз по иерархии.

3. Коллективно - монопольная блокировка намерения (SIX) (Shared Intent with eXclusive)

Накладывается при намерении транзакции читать данные вниз по иерархии объекта и выполнять их частичное изменение, устанавливая монопольные блокировки.

Совместимость блокировок

Совместимость блокировок определяет возможности транзакций одновременно получить блокировку одного и того же ресурса.

Тип запрашиваемой блокировки	Тип наложенной блокировки					
	IS	S	U	IX	SIX	X
IS	+	+	+	+	+	
S	+	+	+			
U	+	+				
IX	+			+		
SIX	+					
X						

Если на страницу наложена коллективная блокировка, то на эту страницу нельзя наложить монопольную (X) блокировку, блокировки намерения коллективно – монопольную (SIX) и монопольную (IX).

Управление блокировками

SQL Server динамически управляет выбором типа и уровня блокировок.

Управление блокировками выполняет специальный компонент сервера – менеджер блокировок.

В функции менеджера блокировок входит установка, снятие и разрешение конфликтов блокировок.

Пользователю чаще всего не нужно предпринимать никаких действий по управлению блокировками.

Решение об уровне блокировки принимается автоматически во время оптимизации запроса.

Например, для операторов модификации данных и **SELECT** для небольшого количества данных блокировка задается на уровне строки или ключа,

а при большом объеме данных для оператора **SELECT**

(например, **SELECT * FROM tableX**)

устанавливается блокировка на уровне страницы или таблицы.

Блокировки

Решение о типе блокировки принимается автоматически по действующему уровню изоляции транзакции.

Уровни изоляции транзакций определяют:

- будут ли блокировки использоваться при чтении данных, и какого типа;
- как долго удерживать блокировки;
- как действовать, если операции чтения потребуется считать данные, на которые распространяется *монопольная блокировка* другой транзакции (ожидать *снятия блокировки*, прочитать незафиксированные данные, прочитать последнюю зафиксированную версию данных).

Т.е. задавая уровень изоляции транзакции, определяют типы и поведение блокировок при выполнении команд в транзакции

Например, при выполнении оператора **Select** для

READ UNCOMMITTED – не устанавливается ни каких блокировок на считываемые данные и игнорируются другие блокировки;

READ COMMITTED – устанавливается коллективная блокировка (S) на считываемые данные только на время выполнения команды;

REPEATABLE READ - устанавливается коллективная блокировка (S) на считываемые данные на время до конца выполнения транзакциию.

Управление блокировками

SQL Server 2008 просмотр

Состояние блокировки:

GRANT: блокировка получена

WAIT : ожидание блокировки

CNVRT: блокировка в конфликтном режиме

exec SP_LOCK 53

Результаты Сообщения

	spid	dbid	ObjId	IndId
1	53	10	0	0
2	53	10	1556200594	1
3	53	10	1556200594	0
4	53	10	1556200594	1

блокировки

Для получения имени идентификаторов можно воспользоваться функциями:
DB_NAME(spид)
OBJECT_NAME(ObjId)

Уровень блокировки

Идентификатор индекса, где удерживается блокировка (0 – собственно таблица)

Идентификатор объекта, где удерживается блокировка

Идентификатор БД в которой удерживается блокировка

Идентификатор сеанса

Блокировки

При разработке транзакции, важно не только определить её содержание и случаи, в которых должен быть выполнен её откат, но также и то, какие блокировки следует удерживать в процессе выполнения транзакции, и какую продолжительность они должны иметь.

Т.е. нужно определиться с уровнем изоляции транзакции

Уровни изоляции транзакций определяют:

- будут ли блокировки использоваться при чтении данных, и какого типа;
- как долго удерживать блокировки;
- как действовать, если операции чтения потребуется считать данные, на которые распространяется *монопольная блокировка* другой транзакции (ожидать *снятия блокировки*, прочитать незафиксированные данные, прочитать последнюю зафиксированную версию данных).

Блокировки

Но при необходимости в запросе можно явно указать какой тип блокировки необходимо использовать в том или ином случае.

Так же можно управлять временем ожидания разблокирования ресурса с помощью команды

```
SET LOCK_TIMEOUT <время_в_мс>
```

Управление блокировками в запросе

При выполнении запроса уровень изоляции действует тот, который был установлен командой,

SET TRANSACTION ISOLATION LEVEL

или по умолчанию)

Если в запросе необходимо установить блокировку или её продолжительность отличную от устанавливаемой по действующему уровню изоляции, то это можно сделать, указав в команде соответствующие специальные ключевые слова (хинты).

Форматы команд с хинтами

SELECT ... FROM table_name **WITH** (hint) **WHERE** ...

INSERT table_name (list_col) **WITH** (hint) **VALUES** ...

UPDATE table_name **WITH** (hint) **SET** ...

DELETE table_name **WITH** (hint) **WHERE** ...

Управление блокировками в запросе

Ключевые слова для явного указания типа блокировки (хинты)

NOLOCK (READUNCOMMITTED) – разрешает **чтение** незафиксированных данных, которые были изменены другими транзакциям.

HOLDLOCK (SERIALIZABLE)– устанавливает **совмещаемую** блокировку **до завершения транзакции**

UPDLOCK– определяет применение блокировки **обновления до завершения транзакции**

XLOCK – определяет применение **монопольной** блокировки на соответствующем уровне **до завершения транзакции**

PAGLOCK – устанавливает блокировку **страницы** вместо **таблицы**

ROWLOCK – устанавливает блокировку **на уровне строки**

TABLOCK – устанавливает соответствующую блокировку **на уровне таблицы**

TABLOCKX – устанавливает **монопольную** блокировку **на уровне таблицы до завершения транзакция**

Управление блокировками в запросе

READCOMMITTED – определяет правила для чтения, как для уровня изоляции **READ COMMITTED** (либо блокировка строк либо управление версиями, в зависимости, что установлено)

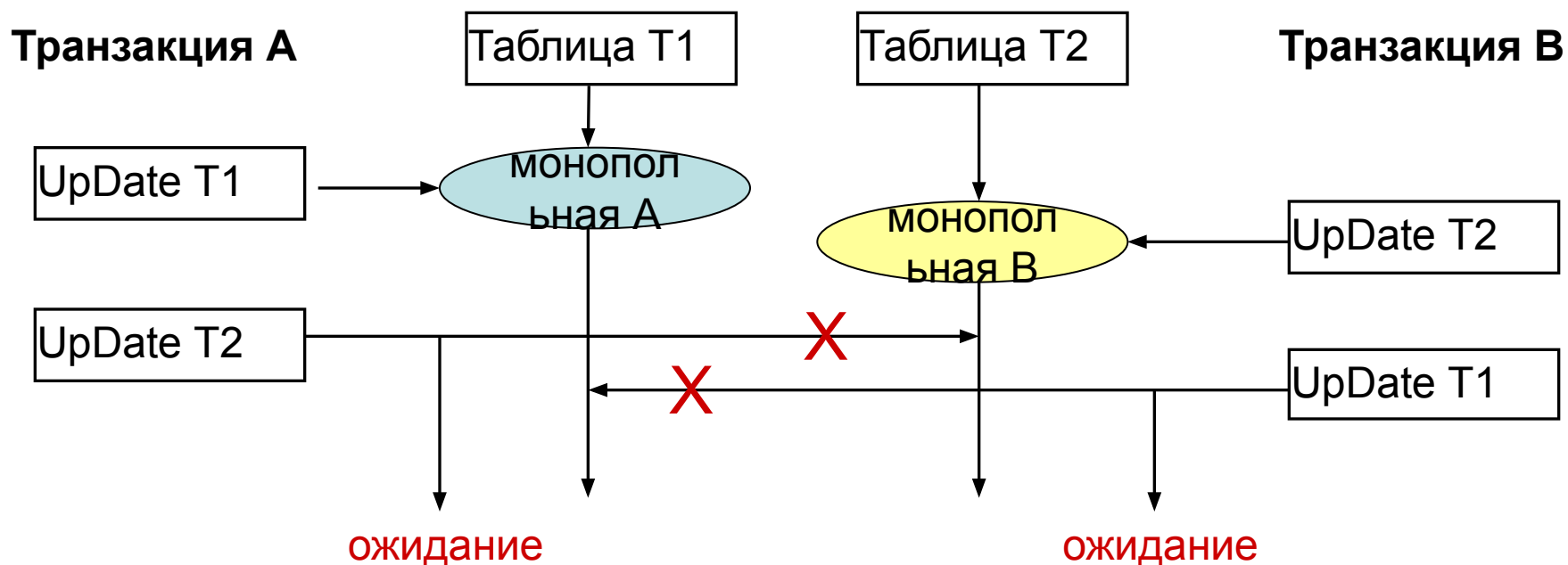
READCOMMITTEDLOCK – определяет правила для чтения, как для уровня изоляции **READ COMMITTED** с использованием блокировки

REPEATABLEREAD – определяет выполнение **просмотра** с семантикой блокировки, как для уровня изоляции **REPEATABLE READ**

и некоторые др.

Тупиковые блокировки

- возникают когда две транзакции блокируют два блока данных и для завершения работы каждой из них необходим доступ к данным, заблокированным ранее другой транзакцией.



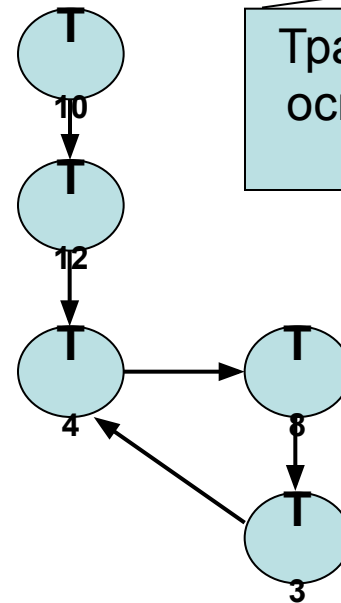
Тупиковые блокировки

Для разрешения конфликта тупиковых блокировок в современных СУБД имеются специальные механизмы обнаружения и разрешения тупиковых блокировок.

Один из алгоритмов обнаружения тупиковых блокировок.

1. Для каждого ресурса строится граф ожидания транзакций
2. Определяется в графе наличия цикла

Если цикл обнаружен, то в системе имеется тупиковая блокировка и далее должна выполняться процедура её разрешения



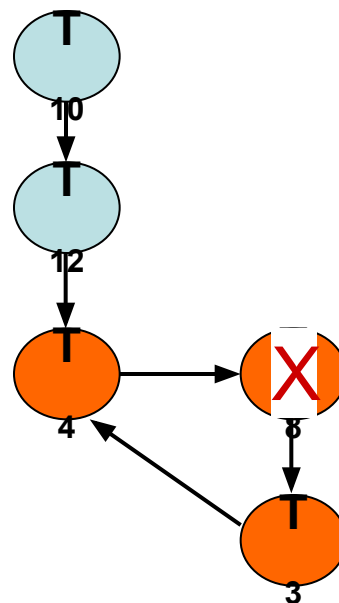
Транзакция T_{10} ожидает освобождения ресурса транзакцией T_{12}

Тупиковые блокировки

В основе стратегии разрешения тупиковых блокировок положен откат одной из транзакции, вызвавших её конфликт.

Выбор на откат транзакции выполняется по принципу:

1. минимальный приоритет;
2. любая из двух с одинаковым приоритетом.



Для установки приоритета блокировки используется команда `SET DEADLOCK_PRIORITY {LOW | NORMAL}`.

Тупиковые блокировки

Для минимизации возможностей возникновения тупиковых блокировок при разработке кода транзакций нужно придерживаться следующих правил:

1. Создание индексов для команд **UPDATE** и **DELETE**, содержащих предложение **WHERE**.

При выполнении этих команд без использования индексов осуществляется монопольная блокировка всей таблицы. При наличии индекса монопольная блокировка устанавливается на строку или страницу.

2. Вместо команды **INSERT** с большим количеством вставляемых строк использовать команду вставки по одной строке (в цикле, используя курсор).

При выполнении команды **INSERT** для вставки много строк осуществляется монопольная блокировка всей таблицы. При наличии **INSERT** для вставки одной строки монопольная блокировка устанавливается на строку.

Тупиковые блокировки

3. Избегать использования в запросах ключевого слова **HOLDLOCK**

При использовании **HOLDLOCK** в **SELECT** все коллективные блокировки будут оставаться в силе, пока вся транзакция не будет завершена. Без его использования – блокировка снимается, как это станет возможным, не дожидаясь окончания всей транзакции.

4. Использовать как можно более короткие транзакции.

- а) разбивать продолжительную транзакцию на короткую;
- б) минимизировать количество некластерных индексов (плотный);
- в) сокращать число столбцов в таблицах (увеличит количество строк на странице и, следовательно, время выполнения транзакций).

Тупиковые блокировки

5. Избегать использования вложенных транзакций

Во всех случаях использования вложенных транзакций все установленные в ней блокировки сохраняются до завершения внешней транзакции.

6. Исключать использования взаимодействия с пользователем во время выполнения транзакции.

7. Использовать как можно более низкий уровень изоляции

Например, `READ UNCOMMITTED` вместо `SERIALIZABLE` позволит нескольким транзакциям одновременно читать данные: каждая транзакция сможет установить коллективную блокировку не дожидаясь пока друга считывает данные и снимет блокировку.

8. Установить на сервере дополнительную оперативную память

Это увеличит КЭШ буферов и следовательно скорость выполнения транзакций и снизит конкуренцию за доступ к ресурсам.