Транзакции

Транзакция - это

последовательность операций, производимых над базой данных и переводящих базу данных из одного согласованного состояния в другое согласованное состояние.

Транзакция

- некоторое неделимое действие над базой данных, осмысленное с точки зрения пользователя
- логическая единица работы системы

Транзакция- требования ACID

- Atomicity — Атомарность
- Consistency — Согласованность
- Isolation — Изолированность
- Durability — Надежность

Atomicity

- Каждая транзакция представляет собой единицу работы.
- Она не может быть разбита на меньшие части.
- Выполняются либо все действия, определенные в данной транзакции, либо не выполняется ни одно из них.

Consistency

- Свойство согласованности гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.
- Для поддержания согласованности данных в процессе транзакции применяются все правила, проверки, ограничения и триггеры.

Isolation

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

Durability

- Свойство долговечности трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Согласованность в базах данных

- *База данных находится в **согласованном состоянии**, если для этого состояния выполнены все **ограничения целостности**.*
- ***Ограничение целостности** - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.*
- Конкурентное использование ресурсов корректными программами может привести к некорректным результатам

Согласованность

- Отказы при выполнении программ-клиентов, СУБД, операционной системы и оборудования могут привести к нарушениям согласованности
- Функция СУБД –гарантировать согласованность при конкурентном выполнении и восстановление согласованности после всех видов отказов

Пример 1
обрыв транзакции

T1

Read (A);

Read (B);

A:=A-100;

B:=B+100;

Write (A)

…

Write (B).

Пример 2
конкурирующие транзакции

T1

Read (A);

A:=A+1;

Write (A).

T2

Read (A);

A:=A+1;

Write (A).

Пример 3
конкурирующие транзакции

T1

Read (A)

A:=A+100

Write (A)

Read (B)

B:=B+100

Write (B)

T2

Read (B)

B:=B*2;

Write (B)

Read (A)

A:=A*2

Write (A)

Транзакции, истории и расписания

- База данных: x, y, z, …
- Операции: r(x), w(x)
- Транзакции: конечная последовательность операций
  – $r_1(x) r_1(y)\ w_1(z)$
- История: упорядоченная совокупность операций нескольких транзакций
  – $r_1(x) r_2(x) w_1(x)\ w_2(x) c_1\ c_2$
- Расписание: префикс истории

Проблемы, вызванные конкурентностью выполнения

- Потеря обновлений
  - $r_1(x)\ r_2(x)\ w_1(x)\ w_2(x)$
- Несогласованное чтение
  - $r_1(x)\ r_2(y)\ w_2(y)\ r_1(y)$
- Грязное чтение
  - $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1$

Серийное расписание

- Расписание, в котором все операции любой транзакции либо предшествуют, либо следуют за операциями любой другой транзакции
- Серийное расписание всегда корректно

Конфликты

- Пара операций из расписания, такая, что
  - Операции принадлежат разным транзакциям
  - Работают с одним элементом данных
  - По крайней мере одна из двух –операция записи
- Конфликты присутствуют в любом нетривиальном расписании

Эквивалентность и серийность

- Множество конфликтов расписания содержит пары, в которых первая операция предшествует второй (и пары находятся в конфликте)
- Расписания эквивалентны, если их множества конфликтов совпадают
- Расписание называется серийным, если оно эквивалентно по конфликтам серийному

- **Серийные расписания корректны**

Критерий серийности

- Граф конфликтов (граф серийности)
  - Вершины соответствуют транзакциям
  - Дуги проводятся для каждого конфликта в направлении конфликта
- Расписание серийно по конфликтам тогда и только тогда, когда граф серийности не содержит контуров

План доказательства

- Граф конфликтов серийного расписания не может иметь контуров, потому что транзакции упорядочены и все дуги направлены от начала к концу расписания
- Если граф не имеет контуров, эквивалентное серийное расписание можно построить с помощью топологической сортировки графа

Сериализуемость по коммутативности

- Любые операции чтения коммутируют
- Любые операции над разными элементами коммутируют
- Расписание сериализуемо по коммутативности, если его можно преобразовать в серийное перестановками соседних операций
- Сериализуемость по коммутативности эквивалентна сериализуемости по конфликтам

Использование замков

- Операции установки

lr(x) – блокировка на чтение, совмещаемая

lw(x) – блокировка на запись, монопольная

- Операции снятия замка ur(x), uw(x)

Решение

T1

Lw(A);

Read (A);

A:=A+1;

Write (A);

Uw(A).

T2

Lw(A);

Read (A);

A:=A+1;

Write (A);

Uw(A).

Совместимость замков

- Замки для одного элемента данных несовместимы, если они устанавливаются разными транзакциями и по крайней мере один из них - на запись.
- Попытка установки несовместимого замка переводит транзакцию в состояние ожидания
- Проблемы, связанные с использованием замков: корректность, тупики, производительность

Протокол блокирования 2PL

- Для каждой операции необходимо предварительно установить замок, все замки должны быть сняты до завершения транзакции
- Транзакция не может устанавливать новые замки после того, как она сняла какой-либо из замков
- Двухфазный протокол генерирует расписания, сериализуемые по конфликтам

Проблемы:

- Бесконечные ожидания
- Тупики

Решения:

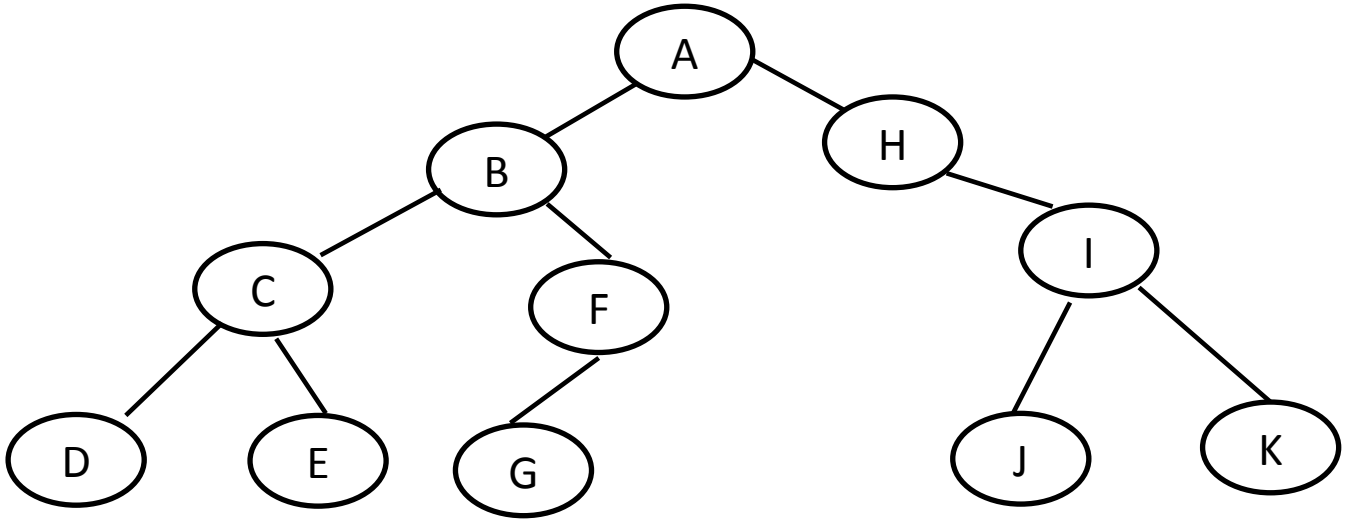
- Установить очередь, возможно с приоритетами
- Две фазы: установка и снятие блокировок
- Установить порядок
- Периодически проверять на тупики и рестарт

Тупики

- $W_2(x)\ w_2(y)\ r_1(y)\ r_2(x)$
- Транзакции, попавшие в тупик, должны быть оборваны
- Граф ожиданий
  - Вершины –активные транзакции
  - Дуги проводится из ожидающей транзакции в транзакцию, установившую несовместимые замки
- Тупик имеет место тогда и только тогда, когда в графе ожиданий имеется контур

Протокол для деревьев WTL

- База данных структурирована как дерево
- Протокол:
  - Все замки – на запись
  - Для установки замка необходимо иметь установленный замок на родительскую вершину (кроме корня дерева)
  - Ни один элемент не блокируется дважды одной и то же транзакцией.
  - Снятие замков возможно в любое время и не препятствует установке новых замков


A
B
H
C
F
I
D
E
G
J
K

Корректность WTL

- Граф сериализуемости ацикличен
- Пусть y –родитель x, и $w_1(x) < w_2(x)$, тогда $w_1(y) < w_2(y)$
- Транзакции сериализуются в том порядке, в котором они устанавливают замки на корень
- Протокол WTL свободен от тупиков
  - Последовательность установки замков определяется последовательностью их установки на корень

Диспетчер транзакций

- Модель СУБД: диспетчер (транзакций) и исполнитель запросов
- Требования к диспетчеру транзакций: корректность и производительность
- Пессимистические и оптимистические протоколы

Виды транзакций в SQL Server

- Явные транзакции
- Автоматическая фиксация транзакций
- Неявные транзакции

Фазы транзакций

- Начало
  BEGIN TRANSACTION
- Конец
  COMMIT
  ROLLBACK

Автоматическая фиксация транзакций

- Режим по умолчанию.
- Каждая отдельная инструкция языка Transact-SQL фиксируется после завершения.
- Нет необходимости указывать какие-либо инструкции для управления транзакциями.

Неявные транзакции

- Установка неявного режима через инструкцию языка Transact-SQL SET IMPLICIT_TRANSACTIONS ON.
- Новая транзакция неявно начинается, когда предыдущая транзакция завершена,
- Но каждая транзакция явно завершается инструкцией COMMIT или ROLLBACK.

Явные транзакции

- Каждая транзакция явно начинается с инструкции BEGIN TRANSACTION и явно заканчивается инструкцией COMMIT или ROLLBACK.

Проблемы параллельного доступа с использованием транзакций

- потерянное обновление (lost update);
- «грязное» чтение (dirty read) — чтение данных, которые были записаны откаченной транзакцией;
- неповторяющееся чтение (non-repeatable read);
- фантомное чтение (phantom reads).

Потерянное обновление

Транзакция 1	Транзакция 2
SELECT f2 FROM tbl1 WHERE f1=1;	SELECT f2 FROM tbl1 WHERE f1=1;
UPDATE tbl1 SET f2=20 WHERE f1=1;	
	UPDATE tbl1 SET f2=25 WHERE f1=1;

«Грязное» чтение

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

Неповторяющееся чтение

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;

Фантомное чтение

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

Размер транзакций

- Логически *транзакция* должна объединять только выполнение взаимосвязанных операций.
- Если делать *транзакции* "очень большими", состоящими из последовательности не связанных между собой операторов, то любой сбой, автоматически выполняющий откат *транзакции*, повлияет на отмену действий, которые могли бы быть успешно завершены при более "коротких" *транзакциях*.

Фазы транзакций

- Установить уровень изоляции
- Начало
  BEGIN TRANSACTION
- Конец
  COMMIT
  ROLLBACK

Уровень изоляции

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

READ UNCOMMITTED

- Не устанавливаются блокировки на чтение
- Транзакции могут считывать измененные другими транзакциями, но не зафиксированные строки. Это позволяет считывать незафиксированные изменения, которые называются чтением«грязных» данных.
- Значения в данных могут быть изменены и до окончания транзакции строки могут появляться и исчезать в наборе данных. Это наименьшее ограничение уровней изоляции.

READ COMMITTED

- Нельзя считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы - «грязные» данных.
- Считанные данные могут быть изменены другими транзакциями во время работы текущей транзакции, результатом чего будет неповторяемое чтение или недействительные данные.
- Этот режим в SQL Server установлен по умолчанию.

REPEATABLE READ

- Нельзя считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями.
- Совмещаемые блокировки применяются ко всем считываемым данным и сохраняются до завершения. Это запрещает другим транзакциям изменять строки, считанные текущей транзакцией.
- Другие транзакции могут вставлять новые строки, соответствующие условиям поиска текущей транзакции. При повторном чтении диапазона могут появиться новые строки «фантомы».

SNAPSHOT

- Копирует БД на момент начала транзакции. Текущая транзакция не видит изменений данных, произведенных другими транзакциями после запуска текущей транзакции.
- Считывание данных транзакциями моментальных снимков не блокирует запись данных другими транзакциями.
- Если транзакция пытается записать данные, измененные кем-то после ее начала, то она обрывается.

SNAPSHOT

- Перед запуском транзакции, использующей уровень изоляции моментальных снимков, необходимо установить параметр базы данных ALLOW_SNAPSHOT_ISOLATION в ON.

- Транзакция, работающая с уровнем изоляции моментального снимка, может просматривать внесенные ею изменения.

SNAPSHOT

```
Select name, snapshot_isolation_state,
    snapshot_isolation_state_desc from
    sys.databases

ALTER DATABASE Database SET
    ALLOW_SNAPSHOT_ISOLATION ON;
```

SNAPSHOT

- После включения изоляции моментального снимка обновленные версии строк для каждой транзакции содержатся в tempdb.
- Применение такого подхода, предусматривающего отказ от блокировок, способствует значительному снижению вероятности взаимоблокировок в сложных транзакциях.

SERIALIZABLE

- Нельзя считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.
- Другие транзакции не могут изменять данные, считываемые текущей транзакцией.
- Другие транзакции не могут вставлять новые строки со значениями ключа, считываемых текущей транзакции, до ее завершения. При повторном считывании будет тот же самый набор строк.

SERIALIZABLE

- Блокировки считанного диапазона сохраняются до завершения транзакции.
- Из-за низкого параллелизма этот параметр рекомендуется использовать только при необходимости.

Уровень изоляции	потерянное обновление	«Грязное» чтение	Неповторяющееся чтение	Фантомное чтение
Не зафиксированные операции чтения	Нет	Да	Да	Да
Зафиксированные операции чтения	Нет	Нет	Да	Да
Повторяющиеся операции чтения	Нет	Нет	Нет	Да
Моментальный снимок	Нет	Нет	Нет	Нет
Сериализуемый	Нет	Нет	Нет	Нет

TRANSACTION NAME

- Строка до 32 символов
- @tran_name_variable
  Имя определенной пользователем переменной, содержащей допустимое имя транзакции.

BEGIN TRANSACTION

```
BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable
    }
]
```

Начинает транзакцию.

COMMIT

BEGIN { TRAN | TRANSACTION } [
transaction_name | @tran_name_variable]]

[;]

…

COMMIT

Фиксирует транзакцию.

Вложенные транзакции

- При использовании вложенных транзакций фиксация внутренних транзакций не освобождает ресурсы и не делает их изменения постоянными. Изменения данных становятся постоянными и освобождаются ресурсы только при фиксации внешней транзакции.
- Вызов каждой инструкции COMMIT TRANSACTION приводит к тому, что значение параметра @@TRANCOUNT уменьшается на 1. Когда значение параметра @@TRANCOUNT уменьшится до нуля, вся внешняя транзакция фиксируется.
- Функция @@NESTLEVEL возвращает уровень вложенности транзакций.

Вложенные транзакции

- Так как аргумент transaction_name не учитывается компонентом Database Engine, то вызов инструкции COMMIT TRANSACTION, содержащей ссылку на имя внешней транзакции, приводит к тому, что если существуют невыполненные внутренние транзакции, то производится только уменьшение значения параметра @@TRANCOUNT на 1.
- При обрыве (откате) вложенной транзакции откатываются все внешние.

Вложенные транзакции

```
create table tbl(id int identity,
field varchar(50))
declare @i bit

begin tran
  insert tbl(field) select 'test1'
  select @i=1 --or 0
  begin tran
    insert tbl(field) select 'test2'
    if @i=1
  rollback tran
    else
  commit tran
  insert tbl(field) select 'test3'
commit tran

select * from tbl
drop table tbl
```

SAVE

SAVE { TRAN | TRANSACTION } {
savepoint_name | @savepoint_variable }

Устанавливает точку сохранения внутри транзакции.

ROLLBACK

ROLLBACK { TRAN | TRANSACTION }
[transaction_name | @tran_name_variable
| savepoint_name | @savepoint_variable]

Откатывает транзакцию до ее начала или до заданной точки сохранения

ROLLBACK

- Инструкция ROLLBACK TRANSACTION без аргумента savepoint_name или transaction_name откатывает изменения на начало транзакции.
- При наличии вложенных транзакций эта инструкция откатывает все внутренние транзакции к началу самой внешней инструкции BEGIN TRANSACTION. В обоих случаях инструкция ROLLBACK TRANSACTION уменьшает @@TRANCOUNT до 0.

ROLLBACK

- Инструкция ROLLBACK TRANSACTION savepoint_name не уменьшает @@TRANCOUNT.
- Инструкция ROLLBACK TRANSACTION в хранимой процедуре не влияет на последующие инструкции в пакете, вызвавшем процедуру; последующие инструкции в пакете выполняются.

```
CREATE TABLE ValueTable ([value] int)
DECLARE @TrName varchar(20) = 'Tr1';
BEGIN TRAN @TrName
    INSERT INTO ValueTable VALUES(1)
ROLLBACK TRAN @TrName
INSERT INTO ValueTable VALUES(2)
SELECT * FROM ValueTable
DROP TABLE ValueTable
```

```
CREATE TABLE ValueTable ([value] int)
DECLARE @TrName varchar(20) = 'Tr1';
BEGIN TRAN @TrName
    INSERT INTO ValueTable VALUES(1)
ROLLBACK TRAN @TrName
INSERT INTO ValueTable VALUES(2)
SELECT * FROM ValueTable
DROP TABLE ValueTable
```

- 2

Пример отката с точкой

```
CREATE TABLE ValueTable ([value] int)
BEGIN TRAN
    INSERT INTO ValueTable VALUES(1)
BEGIN TRAN
    INSERT INTO ValueTable VALUES(2)
SAVE TRAN SavePoint1
BEGIN TRAN
    INSERT INTO ValueTable VALUES(3)
ROLLBACK TRAN SavePoint1
INSERT INTO ValueTable VALUES(4)
SELECT * FROM ValueTable
DROP TABLE ValueTable
```

USE basa_user2	USE basa_user2
BEGIN TRANSACTION TRA	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
	BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4	4.SELECT * FROM Товар (читает измененные неподтвержденные данные)
5.DELETE FROM Товар WHERE КодТовара=4	6.SELECT * FROM Товар (читает измененные неподтвержденные данные)
7.INSERT Товар (Название, остаток) VALUES ('SS',999)	8.SELECT * FROM Товар (читает измененные неподтвержденные данные)
	9.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
10.DELETE FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)	11.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
12.ROLLBACK TRANSACTION TRA	13.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Пользователь user1 Конкурирующая транзакция	Пользователь user2 Текущая транзакция
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL READ COMMITTED BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар
3.UPDATE Товар SET остаток=остаток+10 (захватывает данные)	4.SELECT * FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
5.DELETE FROM Товар WHERE КодТовара=4	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
7.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется той транзакцией, которая первой захватила данные на изменение или удаление)	8.DELETE FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
9.INSERT Товар (Название, остаток) VALUES ('SS',999)	10.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
11.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	12.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

USE basa_user2	USE basa_user2
BEGIN TRANSACTION TRA	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар (захватывает данные)
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется)	4.SELECT * FROM Товар (выполняется)
5.DELETE FROM Товар WHERE КодТовара=4 (блокируется)	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей транзакцией)
7.INSERT Товар (наименование, остаток) VALUES ('SS',999) (блокируется)	8.DELETE FROM Товар WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей транзакцией)
10.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	9.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
	11.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Обработка ошибок

```
CREATE TABLE table1 (
i int NOT NULL PRIMARY KEY,
col1 varchar(20) NOT NULL,
col2 varchar(20) NULL);

INSERT INTO table1 (i,col1,col2)
VALUES (1,'First row','First row');
INSERT INTO table1 (i,col1,col2)
VALUES (2,NULL,'Second row');
INSERT INTO table1 (i,col1,col2)
VALUES (3,'Third row','Third row');
```

Обработка ошибок

```
BEGIN TRAN
INSERT INTO table1 (i,col1,col2)
VALUES (1,'First row','First row');
INSERT INTO table1 (i,col1,col2)
VALUES (2,NULL,'Second row');
INSERT INTO table1 (i,col1,col2)
VALUES (3,'Third row','Third row');
COMMIT TRAN;
```

SET XACT_ABORT { ON | OFF }

- Если выполнена инструкция SET XACT_ABORT ON и инструкция языка Transact-SQL вызывает ошибку, вся транзакция завершается и выполняется ее откат.
- Если выполнена инструкция SET XACT_ABORT OFF, в некоторых случаях выполняется откат только вызвавшей ошибку инструкции языка Transact-SQL, а обработка транзакции продолжается. В зависимости от серьезности ошибки возможен откат всей транзакции при выполненной инструкции SET XACT_ABORT OFF.
  OFF — установка по умолчанию.

XACT_STATE

- Является скалярной функцией, которая сообщает о состоянии пользовательской транзакции текущего выполняемого запроса. Функция XACT_STATE указывает, имеет ли запрос активную пользовательскую транзакцию и может ли она быть зафиксирована.
- SELECT XACT_STATE()
- Тип возвращаемых данных **smallint**

XACT_STATE возвращаемые значения

1	Текущий запрос содержит активную пользовательскую транзакцию. Запрос может выполнять любые действия, включая запись данных и фиксирование транзакции.
0	У текущего запроса нет активной пользовательской транзакции.
-1	В текущем запросе есть активная транзакция, однако произошла ошибка, из-за которой транзакция классифицируется как нефиксируемая. Запросу не удается зафиксировать транзакцию или выполнить откат до точки сохранения; можно только запросить полный откат транзакции.

Обработка ошибок

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

Обработка ошибок

```
BEGIN TRY
 BEGIN TRAN
  INSERT INTO table1 (i,col1,col2) VALUES (1,'First row','First row');
  INSERT INTO table1 (i,col1,col2) VALUES (2,NULL,'Second row');
  INSERT INTO table1 (i,col1,col2) VALUES (3,'Third row','Third row');
 COMMIT TRAN;
END TRY
BEGIN CATCH
 ROLLBACK TRAN
END CATCH;

SELECT @@error;
```

Тупики и бесконечные ожидания

- SET LOCK_TIMEOUT 2000;
- SET LOCK_TIMEOUT -1 – бесконечное ожидание

Гранулярность блокировок

Ресурс	Описание
RID	Идентификатор строки, используемый для блокировки одной строки в куче.
KEY	Блокировка строки в индексе, используемая для защиты диапазонов значений ключа в сериализуемых транзакциях.
PAGE	Страница в базе данных, например страница данных или индекса.
EXTENT	Упорядоченная группа из восьми страниц, например страниц данных или индекса.
HoBT	Куча или сбалансированное дерево. Блокировка, защищающая сбалансированное дерево (индекс) или кучу страниц данных в таблице, не имеющей кластеризованного индекса.
TABLE	Таблица полностью, включая все данные и индексы.
DATABASE	База данных, полностью.

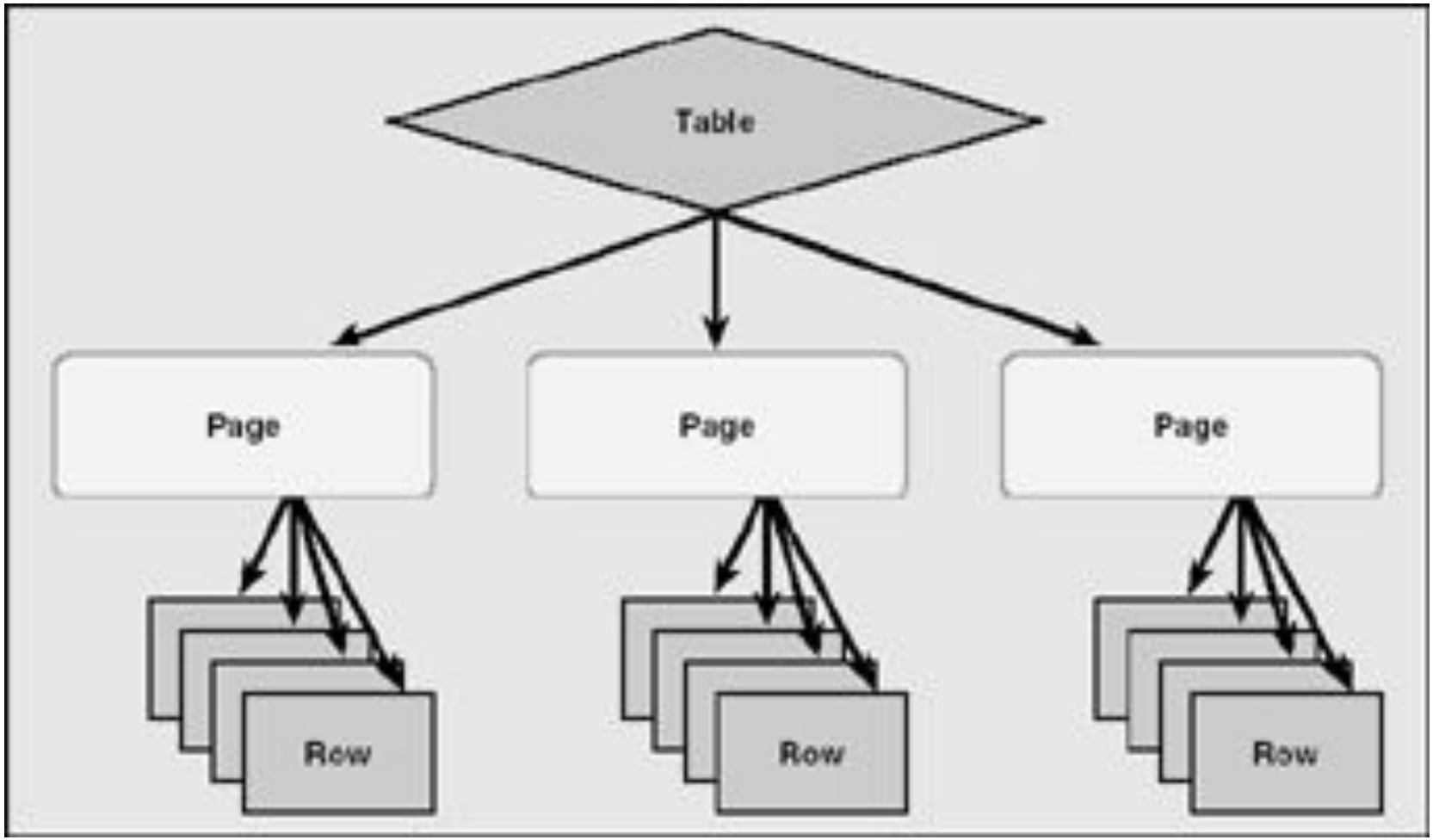
Уровни блокировок

- Table (TAB): Это самая грубая логическая блокировка, которую может использовать SQL Server.
- Extent (EXT): Эти блокировки не используются для блокирования логических строк, а используются, когда SQL Server создает новые таблицы или расширяет существующие, также вы можете их видеть, когда файл увеличивается в размере.
- Page (PAG): Когда SQL Server требуется заблокировать одновременно множество строк, а свободные слоты блокировок заканчиваются, то он может использовать страничные блокировки.
- Key (KEY): Лучший уровень блокировки, возможный в SQL Server, вместе с RID блокировкой. KEY блокировки используются в индексах, а RID блокировки - в таблицах-кучах.

Уровни блокировок

- Высокая конкурентность означает, что множество пользователей может работать одновременно. По возможности это достигается путем небольших блокировок, чтобы не блокировать без необходимости данные, нужные другим пользователям.
- С другой стороны, высокая скорость может быть достигнута при помощи больших блокировок, что быстрее, чем накладывание множества маленьких блокировок.

Иерархия объектов



Эскалация блокировок

- Эскалация блокировок – это процесс, при котором множество блокировок с маленькой гранулярностью, конвертируются в одну блокировку на более высоком уровне иерархии с большей гранулярностью.

Проблемы маленьких блокировок

- **Locking overhead**. иногда выгоднее наложить одну блокировку с большей гранулярностью, чем несколько (или несколько тысяч) более мелких.
- **Data contention**. Наложение одной блокировки - действие атомарное, а наложение нескольких блокировок уже таковым не является.
- **Resource contention**. Блокировки занимают место + нуждаются так же и в некотором обслуживании, которое расходует системные ресурсы (проверка на тупики). + предел количества транзакций, которые могут выполняться в системе одновременно без ущерба для производительности.

Подсказки оптимизатору

- По умолчанию сервер старается наложить блокировку min гранулярности. Если сервер посчитает, что блокировать на уровне отдельной записи не самое оптимальное решение, то он заблокирует больший объем.
- Можно указать явно, с какой гранулярностью блокировать, с помощью специальных подсказок оптимизатору (hints) (в сторону увеличения!).
- ROWLOCK – блокировка на уровне записи.
- PGLOCK - блокировка на уровне страницы данных.
- TABLOCK – блокировка на уровне таблицы.

Восстановление

- Индивидуальный откат транзакции.
- Восстановление после внезапной потери содержимого оперативной памяти.
- Восстановление после поломки основного внешнего носителя базы данных.

Отказы приложений

- завершение оператором ROLLBACK;
- аварийное завершение работы прикладной программы;
- принудительный откат транзакции в случае взаимной блокировки при параллельном выполнении транзакций.

Технические проблемы

- Мягкий сбой
  - при аварийном выключении электрического питания;
  - при возникновении неустранимого сбоя процессора и т. д. Потеря той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.
- Жесткий сбой
  - Восстановление после поломки основного внешнего носителя базы данных. Основой восстановления является архивная копия и журнал изменений базы данных.

Возобновление при отказах

- Отказы приложений/транзакции: транзакции обрываются
- Отказы системы: функционирование возобновляется после рестарта и восстановления согласованности БД
- Разрушение носителя: рестарт, восстановление с резервной копии, восстановление согласованности

Восстановление оборванных транзакций: откат

- Обрывы транзакций могут быть вызваны ошибками при выполнении приложений или невозможностью выполнить транзакцию сериализуемым образом
- Операции обрыва можно заменить на выполнение отката для всех операций записи, включаемых в расписание в обратном порядке с последующей фиксацией

Восстановимость

- Восстановимость расписаний: транзакции должны фиксироваться до того, как их результаты используются другими транзакциями
- Пример невосстановимого расписания
  - $w_1(x)\ w_2(x)\ c_2\ a_1$
- S2L (Двухфазный протокол с удержанием замков на запись до конца транзакции) гарантирует восстановимость

Откат

- По команде rollback система откатит транзакцию на начало (на неявную точку отката)
- По команде commit – зафиксирует всё до неявной точки фиксации, которая соответствует последней завершённой команде в транзакции. Если в транзакции из нескольких команд во время выполнения очередной команды возникнет ошибка, то система откатит только эту ошибочную команду, т.е. отменит её результаты и сохранит прежнюю неявную точку фиксации.

Оптимистический и пессимистический подходы

- Для обеспечения целостности транзакции СУБД может откладывать запись изменений в БД до момента успешного выполнения всех операций, входящих в транзакцию, и получения команды подтверждения транзакции (commit).
- Иногда используется другой подход: система записывает изменения в БД, не дожидаясь завершения транзакции, а старые значения данных сохраняет на время выполнения транзакции в сегментах отката.

Сегмент отката (rollback segment, RBS)

- Сегмент отката – это специальная область памяти на диске, в которую записывается информация обо всех текущих изменениях. Обычно записывается "старое" и "новое" содержимое изменённых записей, чтобы можно было восстановить прежнее состояние БД при откате транзакции (по команде rollback) или при откате текущей операции (в случае возникновения ошибки).
- Данные в RBS хранятся до тех пор, пока транзакция, изменяющая эти данные, не будет завершена. Потом они могут быть перезаписаны данными более поздних транзакций.

Savepoint

- savepoint запоминает промежуточную "текущую копию" состояния базы данных для того, чтобы при необходимости можно было вернуться к состоянию БД в точке сохранения: откатить работу от текущего момента до точки сохранения (rollback to <имя_точки>) или зафиксировать работу от начала транзакции до точки сохранения (commit to <имя_точки>).
- На одну транзакцию может быть несколько точек сохранения (ограничение на их количество зависит от СУБД).

Журнал транзакций

- Журнал транзакций – это часть БД, в которую поступают данные обо всех изменениях всех объектов БД. Журнал недоступен пользователям СУБД и поддерживается особо тщательно. Форма записи в журнал изменений зависит от СУБД. Но обычно там фиксируется:
  - номер транзакции (номера присваиваются автоматически по возрастанию);
  - состояние транзакции (завершена фиксацией или откатом, не завершена, находится в состоянии ожидания);
  - точки сохранения (явные и неявные);
  - команды, составляющие транзакцию, и проч.

Ведение журнала

- Журнал ведется последовательно
- Опережающая запись в журнал (WAL, write-ahead log)
- Регистрируются операции записи, начала и конца транзакций
- Каждая запись содержит данные отката (undo)и «наката» (redo)
- При фиксации запись журнала обязательно «выталкивается» на диск

Фиксация транзакции

- Изменения, внесённые транзакцией, помечаются как постоянные.
- Уничтожаются все точки сохранения для данной транзакции.
- Если выполнение транзакций осуществляется с помощью блокировок, то освобождаются объекты, заблокированные транзакцией.
- В журнале транзакций транзакция помечается как завершенная, уничтожаются системные записи о транзакции в оперативной памяти.

Восстановление после системных отказов

- Необходим рестарт сервера БД
- Для того, чтобы восстановление было возможным, необходимо вести журнал обновлений
- При нормальной работе БД все изменения записываются в журнал
- При рестарте журнал используется для повторения операций и для отката незавершенных транзакций

Общие принципы восстановления

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Что нужно сделать при восстановлении

- Определить, какие транзакции были зафиксированы до отказа и какие были активными (не завершены)
- Если необходимо, повторить изменения зафиксированных транзакций
- Оборвать незавершенные до отказа транзакции (выполнить откат)

Журнал транзакций

- сохранение промежуточных состояний,
- подтверждение транзакции,
- отката транзакции

Журнал транзакций поддерживает:

- восстановление отдельных транзакций;
- восстановление всех незавершенных транзакций при запуске SQL Server;
- откат восстановленной базы данных, файла, файловой группы или страницы до момента сбоя.
- поддержка репликации транзакций;
- поддержка решений высокого уровня доступности и аварийного восстановления

Алгоритм восстановления при рестарте

- Фаза просмотра: найти все зафиксированные и активные транзакции (прямой просмотр журнала)
- Фаза наката (redo): при прямом просмотре, выполнить все операции зафиксированных транзакций
- Фаза отката (undo): обратный просмотр журнала, откат всех операций незавершенных транзакций

Завершение восстановления

- После завершения фазы отката необходимо
  - Записать на диск все измененные блоки БД
  - После записи БД можно очистить журнал
- Возобновить нормальную работу системы

```
CREATE DATABASE Sales
ON ( NAME = Sales_dat, FILENAME =
    'C:\tmp\saledat.mdf',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 5 )
LOG ON ( NAME = Sales_log, FILENAME =
    'C:\tmp\salelog.ldf',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB ) ;
```

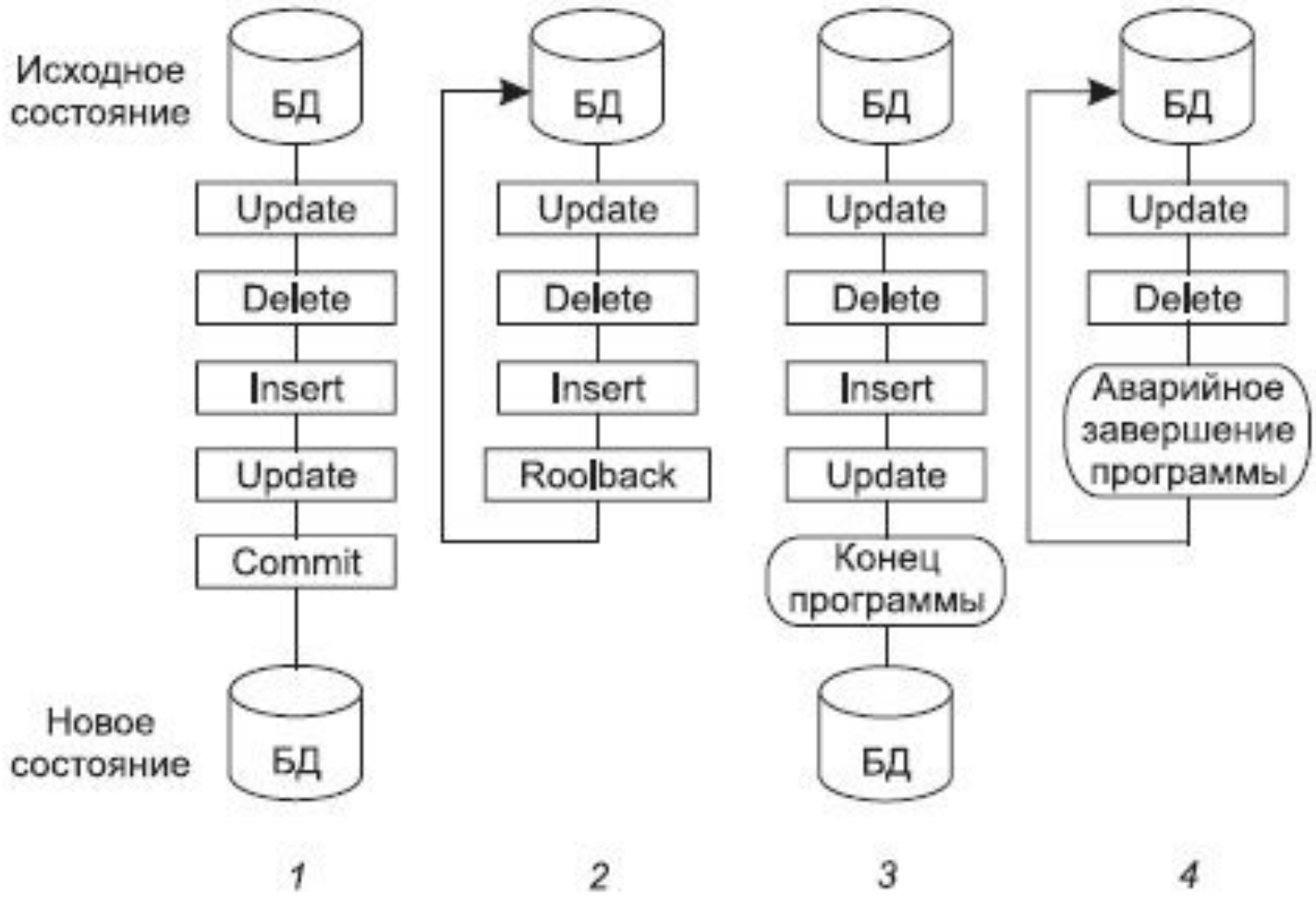
Усечение журнала транзакций

- Процесс усечения журнала освобождает место в файле журнала для повторного использования журналом транзакций.
- Усечение журнала необходимо для предотвращения переполнения журнала.
- При усечении журнала удаляются неактивные виртуальные файлы журнала из логического журнала транзакций базы данных SQL Server.
- Если усечение журнала транзакций не выполняется, со временем он заполняет все доступное место на диске, отведенное для файлов физического журнала.
- Усечение журнала выполняется автоматически

Сжатие журнала

- Усечение журнала не приводит к уменьшению размера физического файла журнала.
- Для уменьшения реального размера физического файла журнала необходимо выполнить его сжатие.
- DBCC SHRINKFILE (DataFile1, 7);
  DataFile1 – имя файла, 7 – размер в МБ);

Модель транзакций



Модель транзакций

1. оператор COMMIT означает успешное завершение транзакции; его использование делает постоянными изменения, внесенные в базу данных в рамках текущей транзакции;
2. оператор ROLLBACK прерывает транзакцию, отменяя изменения, сделанные в базе данных в рамках этой транзакции; новая транзакция начинается непосредственно после использования ROLLBACK;
3. успешное завершение программы, в которой была инициирована текущая транзакция, означает успешное завершение транзакции (как будто был использован оператор COMMIT);
4. ошибочное завершение программы прерывает транзакцию (как будто был использован оператор ROLLBACK).