Лекция 3

Транзакции
ACID
MVCC

Конкурентный доступ

- если с БД работает один пользователь, и в каждый момент происходит только одна операция, то всё просто
- проблемы начинаются, когда многим пользователям нужны одни и те же данные
- тут на помощь приходят транзакции

Транзакция

- набор действий с данными, объединенный в логическую единицу
- либо выполняется целиком, либо нет
- классический пример: перевод денег со счета на счет
- транзакции нужны для получения предсказуемого результата

```
BEGIN;
UPDATE accounts SET balance = balance -
  100.00 WHERE name = 'Alice';
UPDATE accounts SET balance = balance +
  100.00 WHERE name = 'Bob';
COMMIT;
```

- COMMIT - подтверждение транзакции
- ROLLBACK - откат транзакции

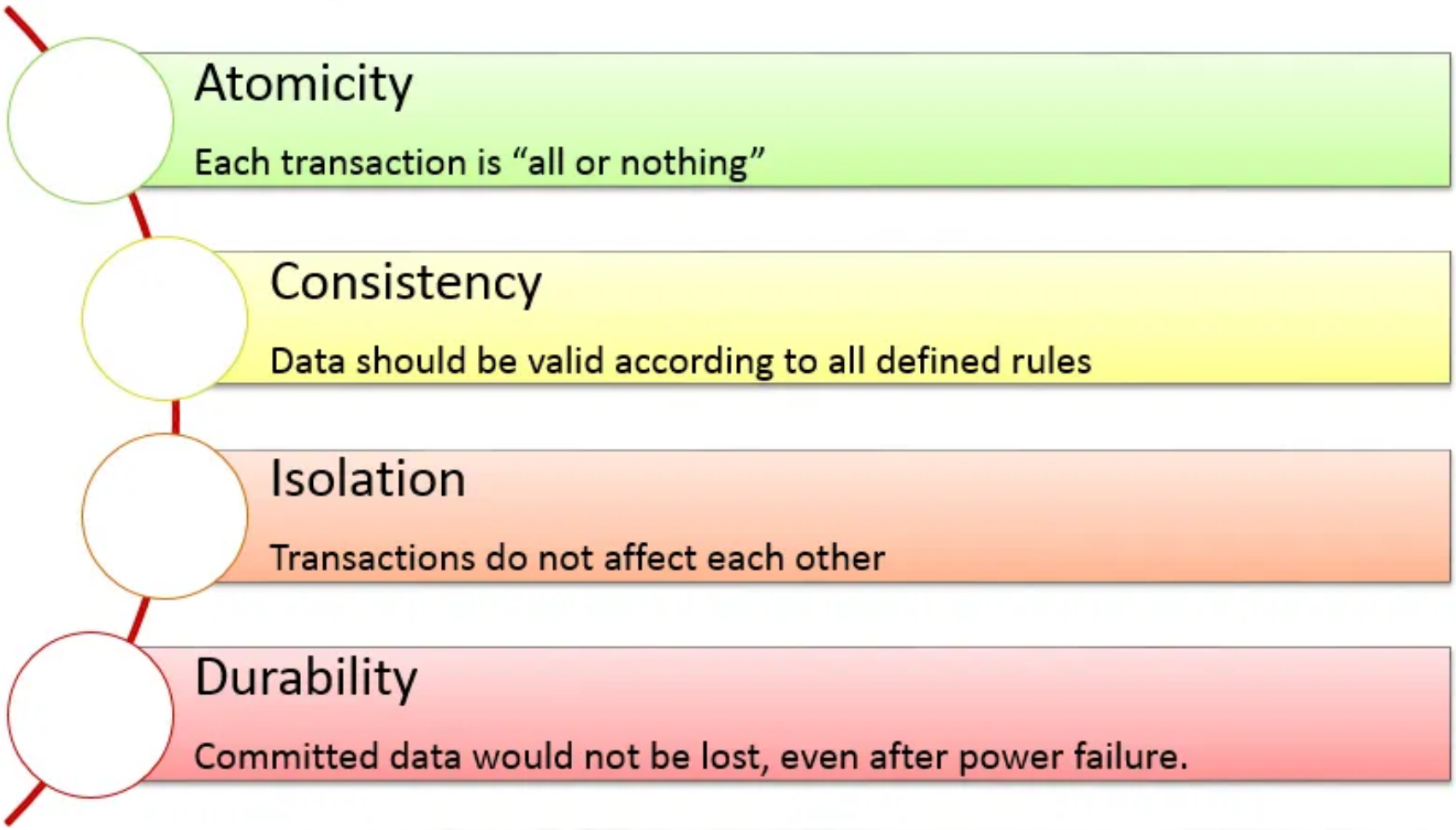
- если операции выполнять вне транзакции, то ошибка в любой из них оставит оба счета в несогласованном состоянии
- например, выйдет так, что деньги снимутся с одного счета, а на другой не придут
- с транзакцией при ошибке все операции откатятся и для пользователя ничего не изменится
- если в процессе возникли сложности и транзакция не может быть выполнена, то вся транзакция откатывается

ACID

ACID - требования к транзакционной системе, обеспечивающие наиболее надёжную и предсказуемую работу

1. Атомарность (atomicity)
2. Согласованность (consistency)
3. Изолированность (isolation)
4. Долговечность (durability)

ACID Properties



Атомарность (atomicity)

- гарантирует, что каждая транзакция будет выполнена полностью или не будет выполнена совсем
- никакая транзакция не будет зафиксирована в системе частично
- будут либо выполнены все её подоперации, либо не выполнено ни одной
- транзакция - единая неделимая часть работы

Согласованность (consistency)

- каждая успешная транзакция фиксирует только допустимые результаты
- все изменения, полученные в результате транзакции не нарушают её структуры
- ключи остаются уникальными, условия ограничений выполняются
- например, если сказано, что баланс не может быть отрицательным, то в результате транзакции он и не будет

Изолированность (isolation)

- параллельные транзакции не будут оказывать влияния на результат других транзакций
- результаты работы транзакции никому не видны, пока она не завершится
- никто не видит промежуточных результатов

Долговечность (durability)

- изменения, получившиеся в результате транзакции, должны оставаться сохраненными вне зависимости от каких-либо сбоев
- если транзакция завершена, данные будут сохранены
- данные переживут аварийное отключение: отключение питание, сбой процесса и т.д. (любую незапланированную остановку работы)

Вопрос

- Как это всё может быть реализовано на практике?

  ACID

  1. Атомарность (atomicity)
  2. Согласованность (consistency)
  3. Изолированность (isolation)
  4. Долговечность (durability)

Как это всё реализовано?

- атомарность и долговечность - журнал транзакций
- изолированность - многоверсионность (MVCC)
- согласованность - проверки на уровне строк

Журнал транзакций

- в PostgreSQL называется журнал предзаписи (WAL - Write-Ahead Logging)
- в простейшем случае журнализация изменений заключается в последовательной записи всех изменений, выполняемых в базе данных

В журнал пришется:

- номер транзакции
- время начала и завершения транзакции
- какие данные изменились (старые-новые данные)
- когда данные сбрасывались на диск
- когда была последняя контрольная точка

Журнал транзакций

- Принцип двойной записи: сначала создаётся запись в журнал, потом изменяются сами данные в таблице
- Основная проблема в обеспечении сохранности данных при записи на диск:

  жёсткий диск не имеет атомарной функции записи

  жёсткий диск медленный. Есть кэши, но они не всегда помогают

Общий алгоритм - 1

- начинается транзакция, ей присваивается какой-то номер
- изменения пишутся в журнал транзакций (состояние до и после) и изменяются страницы БД в памяти (кеши, которые соответствуют данным в таблице)
- приходит подтверждение от приложения, что транзакция завершена (COMMIT)

Общий алгоритм - 2

- в журнал транзакций устанавливается флаг успешного завершения транзакции
- сохраняется на диск журнал транзакций
- меняются файлы, которые соответствуют данным таблицы, и они сохраняются на диск
- данные сохранены, теперь их не потерять

Если происходит сбой

- если сбой произошёл до того, как транзакция завершилась, то в журнале транзакций содержится мусор, при восстановлении они игнорируются
- если журнал транзакций был сохранён, и в журнале отмечено, что транзакция корректно завершена, то эти изменения из журнала транзакций потом применяются к реальным данным

Ещё немного

- сохранение данных таблиц - это дорогая операция, поэтому часто СУБД говорит, что транзакция успеншно сохранена в тот момент, когда записан на диск журнал транзакций (но не сами данные)
- сами данные какое-то время накапливаются, и после накопления какой-то кричиной массы, данные уже сохраняются на диск. Создаётся контрольная точка
- последовательность реального сохранения данных на жёсткий диск никем не регламентируется

Бонус

На основе журнала транзакций реализованы:

1. Point in time recovery (резервная копия + логи журналов)
2. Репликация: есть такая же база на другой машине, журнал транзакций передаётся по сети. В итоге будут две одинаковые БД

Изолированность. MVCC. Блокировки

- разные пользователи могут одновременно работать с одними и теми же данными
- нужно как-то изолировать их друг от друга
- как этого можно добиться?

Вариант 1

- все данные, которые нужны пользователям, сразу блокируются (на чтение и запись)
- плохо для достижения нормальной конкуренции

Решение (вариант 2)

- вместо того, чтобы изменять те данные, что уже есть в базе, сделать новую версию этих данных
- построить движок таким образом, что те запросы, которые работают, видят старую версию, а те, кто придут потом - новую
- появляется идея версионности
- MVCC = MultiVersion Concurrency Control

Мультиверсионность

- пользователь, работая с БД, видит срез данных, который был на момент начала транзакции
- то, что меняется в соседних транзакциях, никак не меняется
- (если разрешать первой транзакции изменять данные таблицы сразу же, то вторая транзакция может получить абсолютно разные данные)

Жила-была таблица без мультиверсионности

id	name	notes
1	Alice	Great at programming
2	Bob	Always talking to alice
3	Eve	Listens to everyone's conversations

Пришла первая транзакция

id	name	notes
1	Alice	Great at programming
2	Bob	Always talking to alice
3	Eve	Listens to everyone's conversations

read

UPDATE

id	name	notes
1	Alice	Great at programming
2	Bob	Always talking to alice
3	Eve	Listens to everyone's conversations

~ update

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
3	Eve	Listens to everyone's conversations

~ update

INSERT

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
3	Eve	Listens to everyone's conversations
4	**Dave**	**Very promising new-hire**

+ insert

DELETE

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
3	Eve	Listens to everyone's conversations
4	**Dave**	**Very promising new-hire**

- delete

Итого

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
4	**Dave**	**Very promising new-hire**

Что на самом деле?

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
3	Eve	Listens to everyone's conversations
4	**Dave**	**Very promising new-hire**

+ update

- delete

+ insert

read

Как было раньше?

- блокировки были на уровне всей таблицы: когда кто-то пытается обновить одну запись в таблице, вся таблица заблокирована, никто не мог даже читать
- блокировки на уровне таблиц сохранились, они используются для всех DDL-операций (например, ALTER TABLE)

Мультиверсионность

- изменение данных в самой таблице уничтожает все данные, которые нужны для обеспечения изолированности
- вводится идентификатор транзакции
- для каждой записи есть id транзакции, которая создала эту запись и id транзакции, до которой эта запись актуальна

Снова жила-была таблица (но с мультиверсионностью)

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	0	2	Bob	Always talk to alice
102	0	3	Eve	Listens to everyone's conversations

TXID: 103

UPDATE

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	0	2	Bob	Always talk to alice
102	0	3	Eve	Listens to everyone's conversations

TXID: 103

- update

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talk to alice*
102	0	3	Eve	Listens to everyone's conversations

TXID: 103

- update

UPDATE

	xmin	xmax	id	name	notes
	100	0	1	Alice	Great at programming
- update	*101*	***103***	*2*	*Bob*	*Always talk to alice*
	102	0	3	Eve	Listens to everyone's conversations
+ update	**103**	**0**	**2**	**Bob**	**Working very hard**

TXID: 103

Пришла следующая транзакция

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talk to alice*
102	0	3	Eve	Listens to everyone's conversations
103	**0**	**2**	**Bob**	**Working very hard**

TXID: 104

INSERT

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talk to alice*
102	0	3	Eve	Listens to everyone's conversations
103	**0**	**2**	**Bob**	**Working very hard**
104	**0**	**4**	**Dave**	**Very promising new-hire**

TXID: 104

+ insert

И ещё одна

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talk to alice*
102	0	3	Eve	Listens to everyone's conversations
103	**0**	**2**	**Bob**	**Working very hard**
104	**0**	**4**	**Dave**	**Very promising new-hire**

TXID: 105

DELETE

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talk to alice*
102	*105*	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	**0**	**4**	**Dave**	**Very promising new-hire**

TXID: 105

- delete

Итого

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talk to alice*
102	*105*	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	**0**	**4**	**Dave**	**Very promising new-hire**

TXID: 106

Плюсы версионности

- писатели не блокируют читателей, отличная конкурентная среда для читателей
- писатели сериализуются по блокировкам: пока одна транзакция работает с данными, никакая другая не сможет эти данные изменить, они останутся целостными
- такая логика была реализована в PostgreSQL в 1999 году (версия 6.5) - MVCC

Варианты реализации MVCC

Heap (PostgreSQL)

- все кортежи лежат в общей "куче"
- мертвые кортежи удаляются в фоне отдельным сборщиком мусора (AUTOVACUUM)
- фискация и откат транзакции имеют одинаковую стоимость

Варианты реализации MVCC

Rollback segment (MySQL, Oracle)

- в страницах таблицы лежат только актуальные данные
- старые версии хранятся в журнале отката
- не требуется сборщик мусора и более эффективное использование дискового пространства
- долгий откат транзакции

Ещё немного разного

- процедура "заморозки" записи. Если СУБД видит, что запись была добавлена очень давно, то есть проставляется специальный индентификатор, что она видна всем и всегда
- версионируются только записи в таблице, индексы не версионируются
- в индексе могут появляться указатели на разные версии с одним и тем же ключом -> необходимость вакуумирования

Проблемы параллельного доступа

- потерянное обновление (Lost Update)
- «грязное» чтение (Dirty Read)
- неповторяющееся чтение (Non-Repeatable Read)
- чтение "фантомов" (Phantom Reads)
- аномалии сериализации

Потерянное обновление (Lost Update)

перезатирание изменений другой транзакцией до завершения транзакции, сделавшей изменения

(ни одна СУБД такого не допускает)

Транзакция 1	Транзакция 2
`UPDATE tbl1 SET f2=f2+20 WHERE f1=1;`	
	`UPDATE tbl1 SET f2=f2+25 WHERE f1=1;`

«Грязное» чтение (Dirty Read)

- чтение данных, добавленных или изменённых еще не завершенной транзакцией
- нельзя говорить о целостности данных
- невозможно в PostgreSQL

«Грязное» чтение (Dirty Read)

Транзакция 1	Транзакция 2

`SELECT f2 FROM tbl1 WHERE f1=1;`
`f2`
`---------`
`100`
`(1 row)`	

`UPDATE tbl1 SET f2=200 WHERE f1=1;`	

	`SELECT f2 FROM tbl1 WHERE f1=1;`
	`f2`
	`---------`
	`200`
	`(1 row)`

`ROLLBACK;`	

Неповторяющееся чтение (Non-Repeatable Read)

при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными

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

Неповторяющееся чтение

- в процессе транзакции видим данные, которых не было на момент начала транзакции
- по умолчанию в PostgresSQL такое поведение допустимо

Чтение "фантомов" (Phantom Reads)

- ситуация, когда при повторном чтении в рамках одной транзакции одна и та же выборка дает разные множества строк.
- от неповторяющегося чтения оно отличается тем, что результат повторного обращения к данным изменился не из-за изменения/удаления самих этих данных, а из-за появления новых (фантомных) данных

Чтение "фантомов" (Phantom Reads)

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

Аномалии сериализации

Ситуация, когда параллельное выполнение транзакций приводит к результату, невозможному при последовательном выполнении тех же транзакций

Транзакция 1	Транзакция 2

`SELECT SUM(value) FROM mytab`
`WHERE class = 1;`
`---------`
`30`
`(1 row)`
	`SELECT SUM(value) FROM mytab`
	`WHERE class = 2;`
	`---------`
	`300`
	`(1 row)`

`INSERT INTO mytab (value, class)` `VALUES (30, 2)`	`INSERT INTO mytab (value, class)` `VALUES (300, 1)`
`COMMIT;`	`COMMIT;`

Уровни изоляции транзакций

- Read uncommitted (чтение незафиксированных данных)
- Read committed (чтение фиксированных данных)
- Repeatable read (повторяемость чтения)
- Serializable (упорядочиваемость)

Уровни изоляции транзакций

Уровень изоляции	Потерянное обновление	«Грязное» чтение	Неповторяющееся чтение	Фантомное чтение	Аномалии сериализации
Read uncommited	не возможно	допускается	возможно	возможно	возможно
Read commited	не возможно	не возможно	возможно	возможно	возможно
Repeatable read	не возможно	не возможно	не возможно	допускается	возможно
Serializable	не возможно	не возможно	не возможно	не возможно	не возможно

Более выской уровень изоляции транзакций уменьшает количество аномалий за счет увеличения количества блокировок и вероятности отката транзакции

Лекция окончена

- Вопросы?
- Пожелания?
- Предложения?