Multi Version Concurrency Control

- Механизм выделения версий данных основан на идее отказа от блокировок для транзакций, читающих данные.
- При этом транзакциям, читающим данные, представляется своя версия данных, существовавшая в тот момент, когда читающая транзакция начала свою работу.
- Для транзакций, меняющих данные, по-прежнему используется механизм блокировок

Некоторая модель

id	value	id_TR
12345	V1	111
12345	V2	121
12345	V3	553
12345	V4	781
12345	V5	900

id_TR	C/R
111	commit
121	rollback
553	commit
781	commit
900	

Transaction-level read consistency

- Какое значение получит транзакция, которая читает данные?
  - если несколько чтений в рамках одно транзакции, она может «увидеть» подтвержденные данные, возникшие после начала ее работы
  - TLRS гарантирует согласование данных в течение всего времени выполнения транзакции

Transaction-level read consistency

- Какая версия данных предоставляется транзакции?
  - Эта версия, существовавшая на момент старта транзакции, а не на момент запроса на чтение данных

T1	X	Y	T2	T3
begin	x0	y0		
W(Y)		y1		
commit				
			begin	
			W(X)	begin
	x2			
				R(X) = x0
			W(Y)	
		y2		
			commit	
				R(Y)= y1

Варианты MVCC

- Read-Only MVCC
- Read-Consistency MVCC
- SNAPSHOT

Read-Only MVCC

- простота реализации
- гарантирует сериализуемость расписания
- возникает при конфликте транзакций R/W и R
- все операции R-транзакции используют последнюю версию, существовавшую на момент старта (требуется TLRC)
- R/W транзакция использует пессимистическую блокировку для всех операций (и R, и W)

Read-Only MVCC

- В момент старта R – транзакция получает номер снимка(snapshot) VC– номер последней зафиксированной на этот момент транзакции
- в ее версию (снимок) попадают все данные, максимальный номер подтвержденных версий которых <= VC
- этот снимок связан с читающей транзакцией до конца ее выполнения
- возможно прерывание читающей транзакции, если ее VC становится «очень старой»
- цена – устаревшие данные

X	Y	Z	
ab 1	38 2	1,23 1	T1 (VC=3)
acb 2	37 3	2,05 4	T2 (VC=3) W(Z) Commit
bc 3	40 5		T3 (VC=4) W(Y)

Read-Consistency MVCC

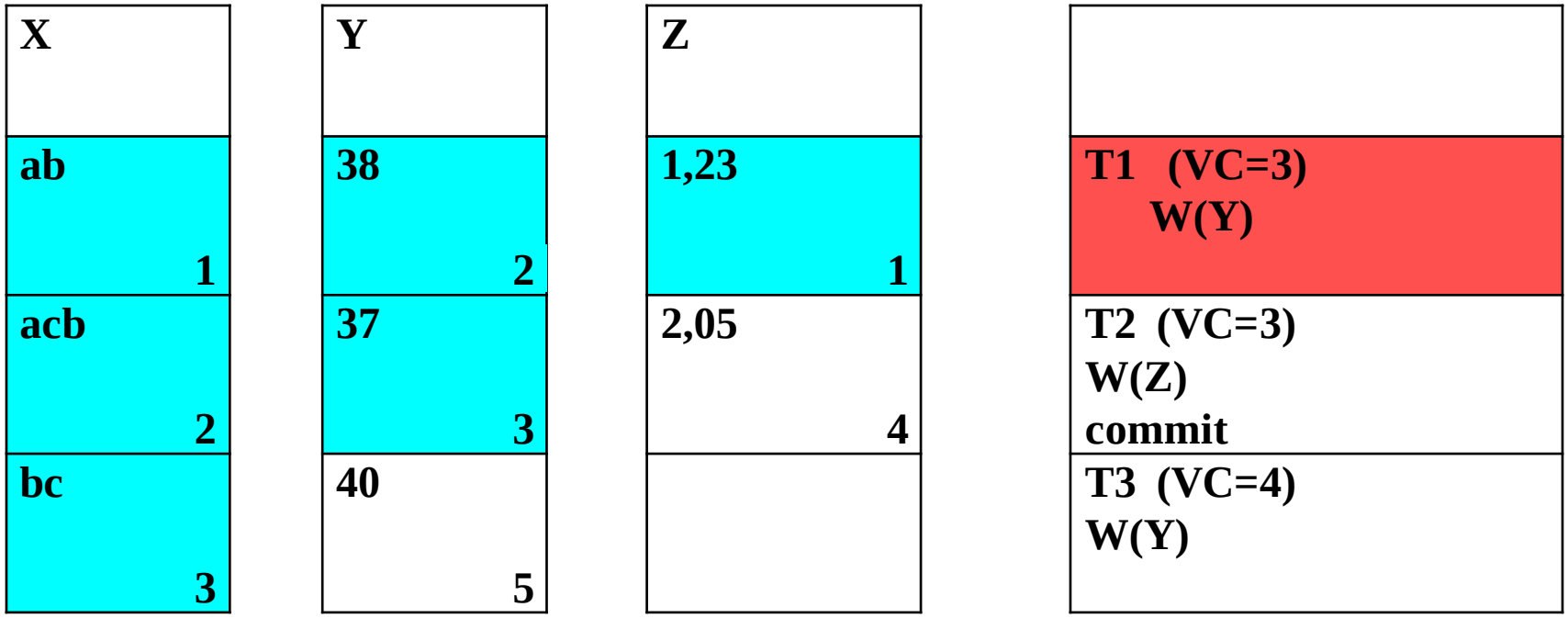
- если нет необходимости поддерживать повторяющиеся чтения
- транзакции R-Only работают также как и в случае Read-Only MVCC
- транзакции R/W
  - для операций записи блокируют последнюю актуальную версию данных
  - для операций чтения получают более новую версию данных (без блокировки)
- цена – возможны несериализуемые графики

Snapshot isolation

- Не только для конфликта R/W и R
- все операции чтения получают версию данных, которая существовала на момент первой операции чтения
- если две транзакции пытаются изменить одни данный (операция записи), одна из них будет принесена в жертву (disjoint-write property)
- важно, что является объектом блокировки - кортеж, таблица или отдельное значение

Snapshot Isolation

- Какую транзакцию принести в жертву?
  - преимущество первого commit (без блокировки). Сравниваются последние версии всех измененных данных и VC транзакции
  - механизм немедленной блокировки для операции, записывающей измененные данные. Заблокировать можно только данные, версия которых не больше VC


X
ab
1
acb
2
bc
3
Y
38
2
37
3
40
5
Z
1,23
1
2,05
4
T1 (VC=3)
W(Y)
T2 (VC=3)
W(Z)
commit
T3 (VC=4)
W(Y)

Snapshot Isolation

- сериализуемость не для всех графиков

T1	X	Y	T2
	10	10	
R(X) = 10			
R(Y) = 10			
			R(X) = 10
			R(Y) = 10
Снять 15			
W(X)	-5		Снять 15
commit		-5	W(Y)
			commit

Реализация MVCC в Firebird

Журнал транзакций

- Каждая транзакция в момент старта получает уникальный номер
- Сервером поддерживается журнал транзакций, в котором фиксируется номер и момент старта транзакции и ее состояние
- Когда транзакция завершит свою работу, в журнале будет отмечено как она была завершена: фиксацией изменений или откатом

Управление транзакциями

Когда сервер получает команду стартовать транзакцию, он делает следующее:

- выделяет номер для новой транзакции
- выделяет 2 бита в Transaction Inventory Page (TIP)-страницах базы данных, предназначенных для учета состояния транзакции

  Все версии записей, создаваемые этой транзакцией, будут помечены ее номером, а другие транзакции при обращении к данным будут следить за ее состоянием в TIP

Видимость версий

Версия записи видна, если
tr_id = self (т.е. транзакция сама создала версию)

- все tr_id < self, которые committed (для транзакций любого типа)
- все tr_id > self, которые committed, если текущая read committed

(где tr_id - идентификатор транзакции конкретной версии записи)

Состояния транзакций

Состояний у транзакции может быть четыре (два бита):

- Активная
- Committed
- Rolled back
- in Limbo

Не влияют на данные. Отмечается только в TIP

Версии данных

- Если транзакция меняет данные (фиксирует изменение), то она помечает эти данные своим системным номером, не уничтожая старые версии данных, помеченные другими системными номерами

- «Грануляция» версий – версия связывается с кортежем

- Новая версия записи создается как delta, т.е. перечень отличий от оригинальной версии

Накопление версий

- Как определять, что версия уже не нужна?
- Когда удалять такие версии?
- Если это происходит в оперативном режиме, то кто должен это делать?

Требования к уровням изоляции

- Никакая транзакция не может видеть неподтвержденные данные от других транзакций
- Самый низкий уровень изоляции позволяет транзакции видеть все подтвержденные изменения базы данных
- Самый высокий уровень изоляции позволяет гарантировать целостность на уровне всей базы данных для данных, с которыми работает транзакция

Уровни изоляции

- READ COMMITTED RECORD_VERSION
- READ COMMITTED NO RECORD_VERSION
- SNAPSHOT
- SNAPSHOT TABLE STABILITY

Уровни изоляции

Уровень изоляции	Проблема				
	потерянные обновления	«грязное» чтение	неповторяю-щееся чтение	фантомы	несовместный анализ
READ COMMITTED RECORD_VERSION	нет	нет	да	да	да
READ COMMITTED NO RECORD_VERSION	нет	нет	нет	да	да
SNAPSHOT	нет	нет	нет	нет	да
SNAPSHOT TABLE STABILITY	нет	нет	нет	нет	нет

Оператор запуска транзакции

```
SET TRANSACTION [NAME имя]
[READ WRITE | READ ONLY]
[WAIT | NO   WAIT]
[ISOLATION   LEVEL]
{SNAPSHOT [TABLE   STABILITY] |
 READ COMMITTED
[[NO] RECORD_VERSION]}
[RESERVING предложение резервирования|
 USING дескрипторы баз данных]
```

Транзакция по умолчанию

SET TRANSACTION
READ WRITE WAIT SNAPSHOT

Если транзакция не запущена явно оператором SET TRANSACTION, то она выполняется в рамках транзакции по умолчанию

READ COMMITTED

- Этот уровень позволяет транзакции видеть все подтвержденные изменения данных, включая и те, которые произошли уже после ее старта

- транзакции read_committed read only стартуют сразу в состоянии committed, поэтому не удерживают «мусорные» записи
- такая транзакция может длиться часами без ущерба для производительности сервера.
- Наиболее характерный пример использования – работа со справочниками

READ COMMITTED RECORD_VERSION

- транзакция имеет право читать и перезаписывать самую последнюю подтвержденную версию записи

READ COMMITTED NO RECORD_VERSION

- транзакция не сможет прочитать строку, если для нее есть измененные версии, ожидающие фиксации
- При этом поведение транзакции будет зависеть от установки ожидания WAIT/NO WAIT

SNAPSHOT

- гарантирует повторяемое чтение и параллельность транзакций
- транзакции SNAPSHOT изолированы от новых версий, подтвержденных другими транзакциями, на этом уровне недопустимо не только неповторяемое считывание, но и возникновение фантомов

SNAPSHOT TABLE STABILITY

- Гарантирует, что данные получаются транзакцией в неизменном состоянии и остаются внешне согласованными в пределах базы данных, пока транзакция не завершена
- Данные, обрабатываемые транзакцией такого уровня могут быть видны только для транзакций READ ONLY

SNAPSHOT TABLE STABILITY

- Блокировка включает все таблицы, к которым осуществляет доступ транзакция, включая те, которые связаны ссылочными ограничениями

SNAPSHOT TABLE STABILITY

- Транзакция такого уровня не может быть запущена до тех пор, пока хотя бы с одной строкой из таблиц, находящихся в зоне видимости транзакции, работает конкурирующая транзакция READ WRITE, даже если она только читает данные
- Уровень изоляции TABLE STABILITY является очень агрессивным и блокирует слишком много таблиц, включая зависимые таблицы

Резервирование

- Иногда более предпочтительным является выполнить резервирование таблиц, указав для них более четкие правила блокировки.
- Для этого используется предложение RESERVING
- Перечислить конкретные таблицы, для которых будет необходима блокировка, задав для каждой из них требуемы уровень защиты:

RESERVING таблица [FOR [SHARED | PROTECTED]]
{READ|WRITE}

Резервирование

- Резервирование является пессимистической блокировкой, оно блокирует все строки таблицы с момента запуска транзакции, не ожидая момента, когда понадобится блокировка отдельной строки
- Если возникает конфликт с другими транзакциями, изменившими данные таблицы и ожидающими завершения, результат зависит от условия WAIT/NOWAIT
- Транзакция может резервировать более одной таблицы

Точки сохранения

- Введены для совместимости с SQL-99
- Поддерживаются только на уровне динамического языка запросов DSQL (например, в утилите командной строки isql)
- Позволяют создавать вложенные транзакции

Точки сохранения

- создать точку сохранения

SAVEPOINT <идентификатор>;

- возврат

ROLLBACK [WORK] TO [SAVEPOINT] <ид-р>;

- освобождение точек сохранения

RELEASE SAVEPOINT <ид-р> [ONLY];

```
create table test (id integer);
commit;
insert into test values (1);
commit;
insert into test values (2);
savepoint   y;
delete from test;
select * from test;          -- returns no rows
rollback to y;
select * from test;          -- returns two rows
rollback;
select * from test;          -- returns one row
```

Явные блокировки

SELECT . . .

FROM <имя таблицы>

[WHERE …]

[FOR UPDATE [OF <список столбцов>]]

WITH LOCK;

Автономные транзакции

- Только в PSQL
- Параметры автономной транзакции наследуются от внешней транзакции, в контексте которой выполняется данный PSQL
- При любой ошибке внутри автономной транзакции она отменяется (происходит rollback автономной транзакции)
- Автономные транзакции не являются вложенными, поэтому между родительской транзакцией и автономной транзакцией возможны конфликты

Автономные транзакции

```
create trigger t_conn on connect
    as
    begin
if (current_user = 'BAD_USER') then
begin
    in autonomous transaction  do
    begin
        insert into log (logdate, msg)
          values (current_timestamp, 'Connection rejected');
    end
     exception e_conn;
end
end
```