Транзакции

Под транзакцией понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными

Основное назначение транзакций в базе данных — переводить ее из одного согласованного состояния в другое

Транзакции также являются *единицами восстановления* данных после сбоев

Транзакции позволяют реализовывать *откладываемые ограничения целостности.*

Такие ограничения проверяются в конце транзакции, в случае их выполнения транзакция завершится оператором COMMIT.

Если ограничения будут нарушены, транзакция может быть откачена оператором ROLLBACK.

Откат транзакции

- В случае отката транзакции аннулируются все изменения, произведенные в ее рамках
- Проблемы
  - Как это реализуется?
  - Кто должен выдавать команду ROLLBACK?
  - Возможны ли вложенные транзакции?

Транзакции в многопользовательских системах

Транзакция рассматривается как последовательность элементарных атомарных операций

Элементарные операции различных транзакций могут выполняться в произвольной очередности

Определения

Набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется ***смесью транзакций***

Последовательность, в которой выполняются элементарные операции заданного набора транзакций, называется ***графиком запуска*** набора транзакций

Если транзакции выполняются одна за другой, то график запуска называется ***последовательным***

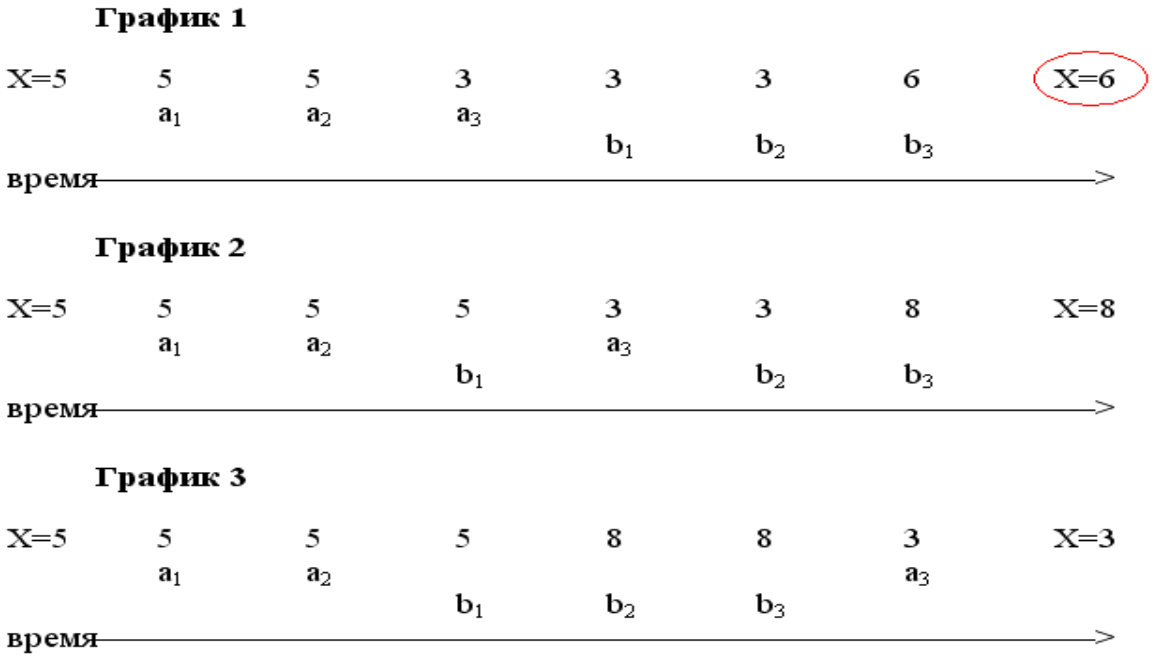
Транзакции называются ***конкурирующими***, если они пересекаются по времени и обращаются к одним и тем же данным

Транзакция A

1. Читать из X
2. Уменьшить на 2
3. Записать в X

Транзакция B

1. Читать из X
2. Увеличить на 3
3. Записать в X



Определение

План (график) выполнения набора транзакций называется ***сериальным***, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного графика выполнения этих же транзакций.

Сериальный план гарантирует предсказуемость результата

Требования к транзакциям (ACID)

Неделимость (Atomicity). Транзакция либо выполняется, либо не выполняется полностью

Согласованность (Consistency). Транзакция переводит базу данных из одного согласованного состояния в другое

Изолированность (Isolation). Результаты транзакции становятся доступны для других транзакций только после ее фиксации

Продолжительность (Durability). После фиксации транзакции изменения становятся постоянными

Конфликты между транзакциями

Утраченные изменения (*конфликт типа W-W)*

Транзакция A	X	Транзакция B
R(x)	X0	
		R(x)
W(x)	Xb	
COMMIT	Xb	
	Xa	W(x)
		COMMIT

Утраченные (не восстановимые) изменения (*конфликт типа W-W)*

Транзакция A	X	Транзакция B
$R(x_0)$	x_0	
		$R(x_0)$
$W(x_a)$	x_a	
COMMIT	x_a	
	x_b	$W(x_b)$
	x_0	ROLLBACK

Незафиксированные зависимости (*конфликт типа W-R)*

Транзакция A	X	Транзакция B
	x_0	$R(x_0)$
	x_0	$W(x_0)$
$R(x_0)$		
обработка (x_0)	x_0	
	x_0	ROLLBACK
COMMIT		

Несовместный анализ (*конфликт типа R-W)*

Случай 1
Неповторяемое считывание

Транзакция A	X	Транзакция B
$R(x_0)$	x_0	
		$R(x_0)$
	x_0	$W(x_0)$
		COMMIT
$R(x_0)$		
COMMIT		

Случай 2
Фиктивные элементы (фантомы)

Пишущая транзакция

```
insert into T(Id, val1)
          values(1000,1);
commit;
```

Читающая транзакция

```
select count(*) from T;



select count(*) from T;
commit;
```

Случай 3
Несогласованное чтение

Пишущая транзакция 1	Пишущая транзакция 2	X	Y	Читающая транзакция
R(x)		X_0		
R(y)			Y_0	
	R(y)			
	W(y)		Y_2	
	COMMIT			R(x),R(y) { X_0, Y_2}
W(x)		X_1		
COMMIT				

Изоляция транзакций

- СУБД должна иметь механизмы, для обеспечения требований ACID
- Эти механизмы могут обеспечивать разную степень (уровень) изоляции транзакций
- СУБД должна всегда обеспечивать недопустимость конфликта W-W. **Недопустимы** утраченные изменения

Изоляция транзакций

Стандартом SQL предусматриваются четыре уровня изоляции транзакций:

- *READ UNCOMMITTED* – чтение незафиксированных данных;
- *READ COMMITTED* – чтение зафиксированных данных;
- *REPEATABLE READ* – повторяющееся чтение;
- *SERIALIZABLE* – последовательное выполнение.

Изоляция транзакций

Уровень изоляции	Проблема			
	потерянные обновления	«грязное» считывание	неповторяющееся чтение	Фантомы
READ UNCOMMITTED	нет	да	да	да
READ COMMITTED	нет	нет	да	да
REPEATABLE READ	нет	нет	нет	да
SERIALIZABLE	нет	нет	нет	нет

Механизмы

•обеспечить, чтобы конкурирующие транзакции выполнялись в разное время

- *Блокировки*
- *Временные метки*

• обеспечить, чтобы конкурирующие транзакции работали с разными версиями данных

Блокировки

- Монопольные (X) eXclusive locks
- Разделяемые (S) Shared locks

Транзакция A наложила	Транзакция B пытается наложить	
	S-блокировку	X-блокировку
S-блокировку	Да	НЕТ (Конфликт R-W)
X-блокировку	НЕТ (Конфликт W-R)	НЕТ (Конфликт W-W)

Что является объектом блокировки?

Логические

- Поле данных
- Кортеж
- Группа кортежей
- Таблица
- Группа таблиц
- База данных

Физические

- Блок
- Сектор
- Файл
- Группа файлов

Правила

- Перед чтением накладывается S блокировка
- Перед изменением накладывается X блокировка
- Если блокировка отвергается, транзакция переходит в состояние ожидания
- Двухфазный протокол синхронизационных захватов (2PL) – сначала наложить слабую блокировку S, если удалось – пытаться наложить X блокировку

Решение проблем с помощью блокировок

- Потеря результатов – да, *но возможны тупиковые ситуации*
- Чтение «грязных данных» - да
- Неповторяемое считывание – да
- «Фантомы» - нет
- Несовместный анализ– да, *но возможны тупиковые ситуации*

Решение проблем тупиков

- Транзакция может задать максимально возможное время ожидания и выполнить откат, если оно превышено
- СУБД может обнаружить тупик и принести одну из транзакций в жертву

Метод временных меток

- «Оптимистические» блокировки
- Используется, если вероятность возникновения конфликтов невелика
- Каждая транзакция получает временную метку (момент начала)
- Если транзакция блокирует объект, она помечает его своей меткой

Метод временных меток

- Прежде, чем заблокировать объект, транзакция анализирует его временную метку и конфликтность ситуации
- При отсутствии конфликта объект помечается наименьшей временной меткой и транзакции продолжают работу
- Если обнаружен конфликт, то более старая транзакция продолжит работу, пометив своим временем объект, а более молодая будет откачена и автоматически стартует вновь

Версии данных
Multi Version Concurrency Control

- Отказ от блокировок для транзакций, читающих данные
- Транзакциям, читающим данные, представляется я версия данных, существовавшая в тот момент, когда читающая транзакция начала свою работу
- Для транзакций, меняющих данные, по-прежнему используется механизм блокировок

Multi Version Concurrency Control

Читающая транзакция ищет среди версий данных те, которые существовали на момент ее старта, т.е. помеченные системным номером меньшим, чем у нее самой

Это гарантирует, что транзакция не увидит изменений, произошедших в данных, начиная с того момента как она стартовала

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

Multi Version Concurrency Control

При использовании MVCC постоянно накапливаются устаревшие версии строк

От СУБД требуется следить за этими версиями и, как только они становятся неактуальными, предпринимать меры по их удалению

Проблема тупиков по-прежнему существует для конфликта W-W

Реализация MVCC в Firebird

Журнал транзакций

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

Некоторая модель

id	Value (видит)	Value (new)	id_TR
12345	null	V1	111
12345	V1	V2	121
12345	V1	V3	553
12345	V1 V3	V4	781
12345	V1 V3 V4	V5	900

Версии строки

id_TR	Time start	C/R
111		commit
121	111 commit	rollback
553	111 commit 121 rollback	commit
781	111 commit 121 rollback 583 commit	commit
900	111 commit 121 rollback 583 commit 781 commit	

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 проверять ее состояние 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

Накопление версий

- «Грануляция» версий – версия связывается с кортежем
- Новая версия записи создается как delta, т.е. перечень отличий от оригинальной версии
- Как и когда определять, что версия уже не нужна?
- Когда удалять такие версии?
- Если это происходит в оперативном режиме, то кто должен запустить этот процесс?

Принципы изоляции в MVCC

- 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 Firebird

- 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

Любые операции с БД происходят только в транзакции. После завершения очередной транзакции оператором commit или rollback, можно выполнить старт новой транзакции, задав для нее уровень изоляции. Если не сделать явно запуск транзакции, как только будет введен любой SQL-оператор, сервер автоматически запустит транзакцию с параметрами по умолчанию

```
SET TRANSACTION
READ WRITE
WAIT
ISOLATION LEVEL SNAPSHOT;
```

Имя транзакции

- Использование именованных транзакций позволяет в приложениях запускать одновременно несколько транзакций
- Безымянная транзакция может быть запущена только одна

Режим доступа

READ WRITE / READ ONLY

READ WRITE – режим по умолчанию

В режиме READ ONLY в контексте данной транзакции могут выполняться только операции выборки данных SELECT. Любая попытка изменения данных в контексте такой транзакции приведёт к исключениям базы данных.

Разрешение блокировок

- Блокировки могут возникать, когда одна транзакция вносит неподтверждённые изменения в строку таблицы или удаляет строку, а другая транзакция пытается изменять или удалять эту же строку
- WAIT / NO WAIT
- WAIT по умолчанию

NO WAIT

- при появлении конфликта блокировки данная транзакция немедленно вызовет исключение и должна будет завершиться командой rollback

WAIT

- при появлении конфликта с параллельными транзакциями, выполняющими конкурирующие обновления данных в той же базе данных, такая транзакция будет ожидать завершения конкурирующей транзакции путём её подтверждения (COMMIT) или отката (ROLLBACK)
- результат разрешения конфликта будет зависеть от уровней изоляции конфликтующих транзакций
- Если для режима WAIT задать е LOCK TIMEOUT, то ожидание будет продолжаться только указанное в этом предложении количество секунд. По истечении его для транзакции буде выброшено исключение

Уровень изолированности

- SNAPSHOT
  - Это уровень изолированности по умолчанию
- SNAPSHOT TABLE STABILITY
- READ COMMITTED
  - Имеет два варианта:

  READ COMMITTED NO RECORD_VERSION

  READ COMMITTED RECORD_VERSION

SNAPSHOT

- Этой транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции.
- Любые подтверждённые изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции в процессе ее активности без её перезапуска.
- Чтобы увидеть эти изменения, нужно завершить транзакцию (подтвердить её или выполнить полный откат) и запустить транзакцию заново
- Изменения, вносимые автономными транзакциями, также не будут видны в контексте той ("внешней") транзакции, которая запустила эти автономные транзакции, если она работает в режиме SNAPSHOT

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

- Транзакцию с таким уровнем можно запустить только, если в базе нет неподтвержденных изменений данных ни в одной из таблиц
- После старта такой транзакции на одна из параллельных транзакций не сможет менять данные, которые были изменены в этой транзакции
- Эта транзакция не будет видеть те изменения в данных, которые произошли после ее старта

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
- 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

READ COMMITTED

- Позволяет в транзакции без её перезапуска видеть все подтверждённые изменения данных базы данных, выполненные в других параллельных транзакциях.
- Неподтверждённые изменения не видны в транзакции и этого уровня изоляции.
- Для получения обновлённого списка строк интересующей таблицы необходимо лишь повторное выполнение оператора SELECT в рамках активной транзакции READ COMMITTED без её перезапуска

READ COMMITTED RECORD_VERSION

- транзакция имеет право читать и перезаписывать самую последнюю подтвержденную версию записи
- для этой записи могут существовать неподтвержденные версии, они не влияют на старт транзакции
- режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции при её старте

READ COMMITTED NO RECORD_VERSION

- Транзакция не сможет прочитать строку, если для нее есть измененные версии, ожидающие фиксации
- Если указана стратегия NO WAIT, то будет немедленно выдано исключение.
- Если указана стратегия разрешения блокировок WAIT, то это приведёт к ожиданию завершения или отката конкурирующей транзакции.
- Если конкурирующая транзакция откатывается, или, если она завершается и её идентификатор старее (меньше), чем идентификатор текущей транзакции, то изменения в текущей транзакции допускаются.
- Если конкурирующая транзакция завершается и её идентификатор новее (больше), чем идентификатор текущей транзакции, то будет выдана ошибка конфликта блокировок.

READ COMMITTED READ ONLY

- 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 записи
- такая транзакция может длиться часами без ущерба для производительности сервера.

Резервирование

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

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

Совместимость блокировок

	SHARED READ	SHARED WRITE	PROTECTED READ	PROTECTED WRITE
SHARED READ	да	да	да	да
SHARED WRITE	да	да	нет	нет
PROTECTED READ	да	нет	да	нет
PROTECTED WRITE	да	нет	нет	нет

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
- Следует учитывать, что резервирование может иметь разный эффект при разных уровнях изолированности транзакции

Точки сохранения

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

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
```