БАЗЫ ДАННЫХ

Лекция 4

ПЛАН НА СЕГОДНЯ

- Теория вокруг WAL
- WAL в Postgres
- Индексы на B+-деревьях

АБСТРАКТНАЯ ТРАНЗАКЦИЯ

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t := t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t := t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

ОБЩИЕ МОМЕНТЫ

- Заводим лог-файл
- Заносим туда записи в режиме добавления и делаем flush
- В начале транзакции пишем <START t>
- В конце пишем <COMMIT t>

ОБЩИЕ МОМЕНТЫ

- Между START и COMMIT могут быть другие записи
- Записи, соответствующие транзакции, помечаются id транзакции
- Транзакции могут перемежаться

COMMIT

- Конкретная семантика COMMIT зависит от метода
- В целом означает "все сохранено, ничто не пропадет"
- Но не факт, что все лежит прямо в штатной таблице
- Если нет, то гарантируется, что при восстановлении оно туда попадет

UNDO LOG

- Перед OUTPUT пишем в лог запись типа <T, A, value>
- T - транзакция, A - переменная, value - значение
- Перед COMMIT все измененные переменные обязаны быть записаны на диск

ПРИМЕР

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t*2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<$T, A, 8$>
5)	READ(B,t)	8	16	8	8	8	
6)	t := t*2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<$T, B, 8$>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<COMMIT T>
12)	FLUSH LOG						

ДЕТАЛИ

- Если успели записать (OUTPUT), но не успели записать COMMIT - транзакция пропадет
- Запись в лог перед OUTPUT - для каждой отдельной переменной
- Если OUTPUT раньше - можем не восстановить (если не успели записать в лог)

ОПАСНЫЙ СЦЕНАРИЙ

- "Переменной" считаем запись
- Вошли в одну транзакцию
- Сделали WRITE
- Есть основания не торопиться с записью в лог и с OUTPUT

ОПАСНЫЙ СЦЕНАРИЙ

- Другая транзакция работает с той же страницей
- Но с другой записью
- Она хочет закончить
- И она делает OUTPUT
- Заодно - OUTPUT в первой транзакции

ВОЗМОЖНЫЕ РЕШЕНИЯ

- Стимулировать OUTPUT в другой
- Временно "откатить" другие WRITE, потом вернуть
- Не допускать параллельного использования страниц

ВОССТАНОВЛЕНИЕ НА ПОЛНОМ ЛОГЕ

- Вот мы перезагрузились
- Хотим все привести в корректное состояние
- Если всем началам транзакций соответствует завершение - все ОК
- Все незавершенное надо вернуть в первосданный вид

ВОССТАНОВЛЕНИЕ НА ПОЛНОМ ЛОГЕ

- Сканируем лог с конца (это важно)
- Отмечаем встреченные COMMIT-ы
- Встретив запись об изменении, смотрим на транзакцию
- Если не закрытая, то восстанавливаем значение
- Пишем в конец лога маркер <ABORT t>

ВАЖНЫ ДЕТАЛИ

- С конца - чтобы обработать вариант одной переменной, измененной в нескольких транзакциях
- Сначала восстанавливаем - потом пишем
- Потому что во время восстановления тоже можем упасть

CHECKPOINT

- Не хотим всегда идти по всему логу
- Хотим ставить метку, раньше которой не смысла идти
- Раньше которой все гарантированно хорошо

CHECKPOINT

- Такая метка легко ставится при восстановлении
- Или при нормальном рестарте
- А мы хотим, чтобы база год работала без рестарта
- И лог не хранить за год

CHECKPOINT

- И притормаживать новые транзакции до завершения старых - тоже
- Или при нормальном рестарте
- Мы знаем, какие транзакции активны
- Запишем в лог специальную запись: <START CKPT (T_1 ,. .. , T_k) >
- Зафиксировали начало чекпойнта
- И номера активных транзакций

CHECKPOINT

- Принимаем новые транзакции
- Ждем завершения старых
- Если дождались - пишем <END CKPT> в лог
- Важно: чекпойнты под нашим конктролем

CHECKPOINT

- Важно: чекпойнты под нашим конктролем
- О вложенности речи не идет
- Если в логе есть <END CKPT>, то у него есть парный START
- И все, что до него - можно выкинуть
- Хоть онлайн, хоть при рестарте - как удобнее

CHECKPOINT

- Если при восстановлении видим <END CKPT>, то продолжаем идти по логу
- Там могут быть куски незавершенных транзакций
- И за START идти нет смысла
- А если видим START без <END CKPT> - то надо идти
- Но только за теми, кто указан в START

В КОНТЕКСТЕ СУБД

- Сомнительный вариант при частых маленьких изменениях
- Может быть неплох при крупных транзакциях
- Когда много изменений (многостраничный update)
- Может помогать поддерживать версионность

В КОНТЕКСТЕ СУБД

- Если кто-то делает SELECT
- И хочет увидеть страницы на начало транзакции
- Его можно направить в лог
- Особенно есть страницы кладутся в лог "как есть"

В КОНТЕКСТЕ СУБД

- Может пригодиться вспомогательный словарик
- Указывающий, где искать старую версию страницы
- Возможно - отдельный кеш лога

REDO-ЛОГ

- "Инверсия" Undo
- Пишем в лог не старое, а новое
- При восстановлении игнорируем незавершенные
- И обрабатываем завершенные

REDO-ЛОГ

- Сначала commit, потом запись
- <T, X , v> - "транзакция T записала v в X"
- v - новое значение

ПРИМЕР

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t*2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<$T, A, 16$>
5)	READ(B,t)	8	16	8	8	8	
6)	t := t*2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<$T, B, 16$>
8)							<COMMIT T>
9)	FLUSH LOG						
10)	OUTPUT(A)	16	16	16	16	8	
11)	OUTPUT(B)	16	16	16	16	16	

КРАТКИЙ АНАЛИЗ

- Killer-feature - запись после коммита
- Есть ряд минусов: дорогое восстановление, например
- Это плата за killer-feature

ВОССТАНОВЛЕНИЕ

- Определяем незакрытые транзакции
- Идем по логу от начала
- Если запись незакрытой транзакции - пропускаем
- Если от закрытой, то восстанавливаем

СЛОЖНОСТИ REDO

- Два прохода (или доп. приседания)
- Больше действий (или доп. приседания)
- Особая важность порядка - транзакции не спасут
- И checkpoint похитрее

CHECKPOINT В REDO

- То же пишем START и END
- И в START указываем начатые
- Но совсем в других целях
- И ждем мы совсем не их завершения

CHECKPOINT В REDO

- Храним еще знание о том, какие записи не сделаны
- И ждем, пока они завершатся
- По факту - стимулируем их завершение
- И когда они завершатся - можно завершать checkpoint

ПРИМЕР

<START T_1>
<$T_1, A, 5$>
<START T_2>
<COMMIT T_1>
<$T_2, B, 10$>
<START CKPT (T_2)>
<$T_2, C, 15$>
<START T_3>
<$T_3, D, 20$>
<END CKPT>
<COMMIT T_2>
<COMMIT T_3>

ВОССТАНОВЛЕНИЕ

- При закрытом последнем чекпойнте
  - Все закомиченное до его начала приехало на диск
  - Чего нельзя сказать про то, что в START-списке
  - Нужно заглядывать раньше, чем START

ВОССТАНОВЛЕНИЕ

- При незакрытом последнем чекпойнте
  - Ничего не знаем про output
  - Ищем последний закрытый
  - Дальше - знаем, что делать

UNDO-REDO

- Смешанный подход
- Пишем больше данных
- Расширяем возможности

UNDO-REDO

- Пишем старое значение и новое
- Единственное требование: output после write
- Порядок COMMIT и OUTPUT - произвольный

ПРИМЕР

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t*2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<$T, A, 8, 16$>
5)	READ(B,t)	8	16	8	8	8	
6)	t := t*2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<$T, B, 8, 16$>
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT(B)	16	16	16	16	16	

ВОССТАНОВЛЕНИЕ

- Проходим по закомиченным - как в Redo
- Проходим по незакомиченным - как в Undo
- Именно в таком порядке

CHECKPOINT

- В целом - как в Redo
- Можно гибче - в пределе можно скинуть все грязные страницы кеша
- Если на диск попадет незакомиченное - все равно откатим

POSTGRES

- Redo-схема
- wal writer
- bg writer
- checkpointer

ИНДЕКСЫ НА B+-ДЕРЕВЬЯХ

- Почему нас не устраивают AVL или красно-черные ?
- Потому что они слишком высокие
- Потому что слишком малые узлы
- А еще потому что там перебалансировки

ИНДЕКСЫ НА B+-ДЕРЕВЬЯХ

- Пусть узлы будут размером со страницу
- Это позволит иметь больше детей
- И снизит высоту дерева

ИНДЕКСЫ НА B+-ДЕРЕВЬЯХ

- Пусть у нас числовые ключи и они не равны
- Отведем место на ключ
- На указатель на записи
- На ссылку на страницу
- Получается сотни ключей на узел дерева

ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ

- Сначала выделяем страницу
- И в нее добавляем пары (ключ, указатель на запись)
- Пока хватает места

КОГДА ЗАПОЛНИЛАСЬ

- Берем средний ключ
- Выделяем две страницы
- На одну из них переезжает половина ключей
- Превосходящие средний ключ

ПРОДОЛЖЕНИЕ

- На другую - средний ключ
- И ссылки на страницы с остальными ключами
- Теперь при добавлении выбираем, в какую страницу

ПРОДОЛЖЕНИЕ

- Когда заполнилась страница, делим ее ключи
- По отношению к их среднему
- Выделяем страницу
- Ключи, бОльшие среднего, идут туда
- В корне - новая запись