OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

MVCC, vacuum и autovacuum

Коробков Виктор

Консультант команды технологического обеспечения
ООО «ИТ ИКС5 Технологии»

Telegram: @Korobkov_Viktor

Преподаватель

Виктор Коробков

более 20 лет в IT

специализация: проектирование баз данных (СУБД PostgreSQL, MS SQLServer)

В OTUS веду занятия на курсах: СУБД, PostgreSQL, SQL Server Developer, noSQL, Программист C

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в Telegram

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

Цели вебинара

После занятия вы сможете

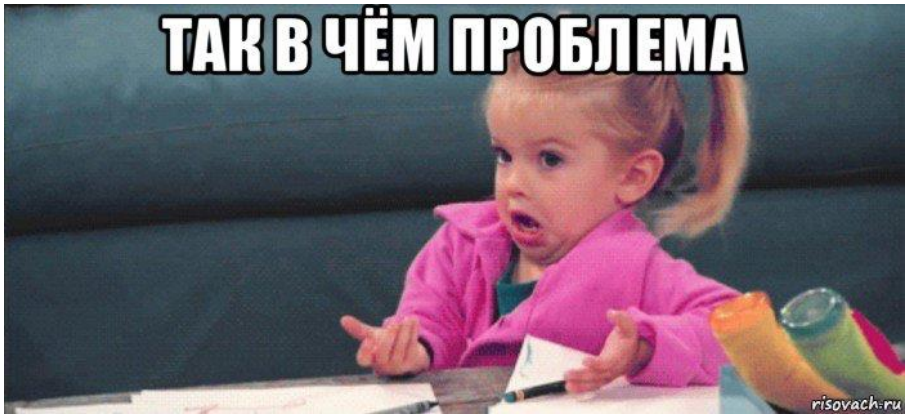
1. Понимать работу механизма многоверсионности в PostgreSQL
2. Знать и уметь использовать vacuum и autovacuum
3. Понимать назначение заморозки транзакций

Транзакция

- последовательность операций
- рассматривается базой данных как атомарное действие
- завершается подтверждением изменений (commit) либо откатом изменений (rollback).

Транзакция

- последовательность операций
- рассматривается базой данных как атомарное действие
- завершается подтверждением изменений (commit) либо откатом изменений (rollback).



Транзакция

Проблема во **множественной параллельной нагрузке!!!**

Как обеспечить параллельную работу множества сессий так, чтобы они могли:

- модифицировать данные;
- не мешать друг другу (ни при записи, ни при чтении);
- обеспечивать целостность данных;
- обеспечивать надежность данных.

Что делать ???

Транзакция


транзакция
обладает набором свойств
АСИД
в него входят
атомарность
изолированность
согласованность
долговечность
обеспечивает
сохранение целостности
параллельную работу
восстановление при отказах и сбоях

Способы реализации ACID

ARIES (Algorithms for Recovery and Isolation Exploiting Semantics) – алгоритмы восстановления систем:

- logging – запись в журнал всех действий транзакции, которые могут изменить состояние БД;
- поддержка покортежных блокировок;
- checkpoints;
- undo;
- redo.

Способы реализации ACID

MVCC (MultiVersion Concurrency Control) - механизм обеспечения параллельного доступа к БД:

• каждой сессии предоставляется «снимок» БД;

• изменения в БД невидимы другим пользователям до момента фиксации транзакции;

• читатели не блокируют читателей;

• писатели не блокируют читателей;

• читатели не блокируют писателей.

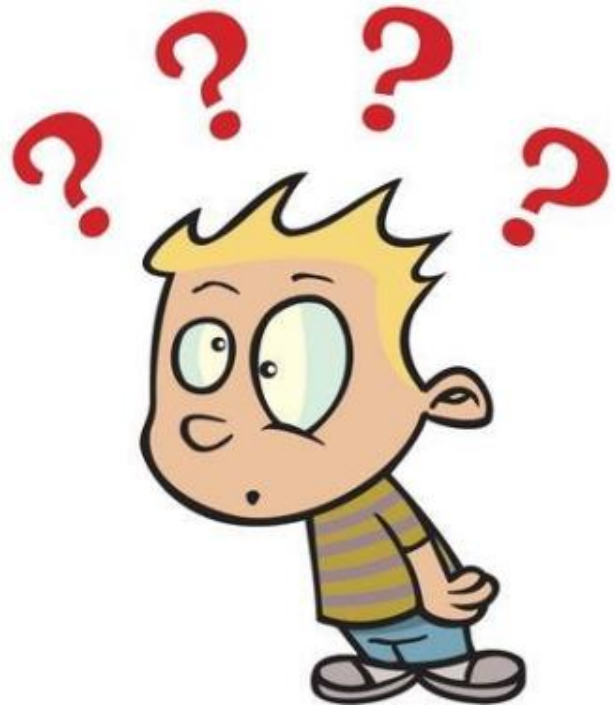
Что из этого конкурентный доступ к данным ?

A. Загрузка или выгрузка большого объема данных в рамках автоматического процесса.

B. Корпоративная система периодической отчетности.

C. Система продажи билетов в кинотеатры.

D. Система для снятия, хранения и обработки данных с большого количества датчиков температуры.



Подсказка

Конкурентный доступ - это там, где много субъектов и нельзя договорится:

- например подождать;
- или действовать упорядоченно;
- как правило это система у которой много пользователей и они люди;
- а если не люди, то нужен быстрый ответ от системы.

Минусы конкурентного доступа

- сложные алгоритмы реализации;
- дорого с точки зрения затрат на ресурсы - процессоры, память, диски …

Поэтому: используйте реляционные СУБД только там, где есть реальный конкурентный доступ и важен ACID !!!

Что было раньше …

Двухфазная блокировка - two-phase locking (2PL)

• самый первый метод обеспечения конкурентного доступа (алгоритм 2PL был представлен в 1976 г.);

• основан на блокировках.

Подробнее: How does the 2PL (Two-Phase Locking) algorithm work

Что сейчас …

Multi Version Concurrency Control (MVCC):

• современный и наиболее популярный метод обеспечения конкурентного доступа;

• практически не использует блокировок:

только писатель блокирует писателя если они пытаются работать с одной записью;

или ручная блокировка select for update…

ALL RDBMS MVCC

Используется сегмент отката - Undo (rollback) segment:

- в него копируются оригинальные версии измененных данных;
- на их место вставляются новые данные;
- если произошел откат (rollback), необходимо все вернуть на свои места;
- если много незавершенных транзакций - много лишней работы для СУБД.

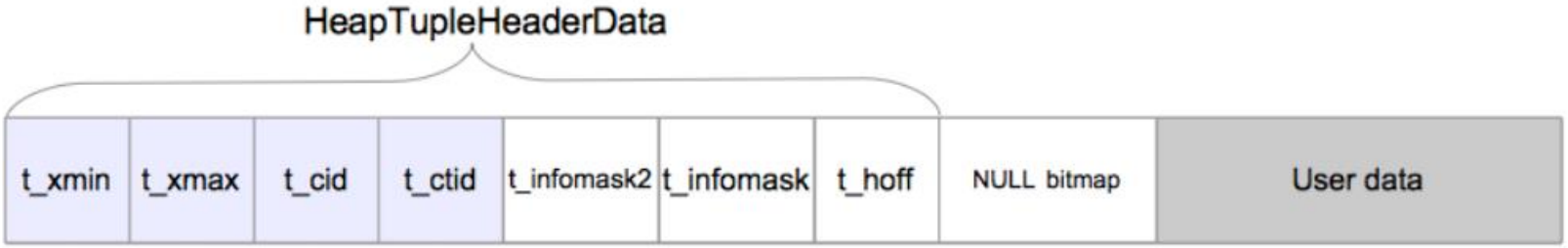
Используется: Oracle, MySQL InnoDB, ...

Postgres MVCC

Tuple multi versioning – механизм многоверсионности: данные не меняются и не удаляются в процессе обработки транзакций, а создаются новые версии записей.

Дорого для операции update…

Формат записи (tuple)



Скрытые атрибуты записи:

xmin – идентификатор транзакции, создавшей запись;

xmax – идентификатор транзакции, удалившей запись;

cmin – порядковый номер команды в транзакции, добавившей запись;

cmax – номер команды в транзакции, удалившей запись.

Формат записи (tuple)

infomask – содержит ряд битов, определяющих свойства данной версии:

xmin_commited, xmin_aborted

xmax_commited, xmax_aborted

ctid – является ссылкой на следующую, более новую, версию той же строки.

У самой новой, актуальной, версии строки ctid ссылается на саму эту версию.

Номера ctid имеют вид (x,y):

x — номер страницы,

y —порядковый номер указателя в массиве.

PostgreSQL MVCC DML

Insert

добавляется новая запись с xmin=txid_current() и xmax=0

Delete

в записи xmax=txid_current()

Update

в старой версии записи xmax=txid_current(), то есть делается delete

добавляется новая запись с xmin=txid_current() и xmax=0, то есть делается insert

PostgreSQL MVCC

Отмена выполняется быстро.

Хоть команда и называется ROLLBACK, отката изменений не происходит: все, что транзакция успела изменить в страницах данных, остается без изменений.

При обращении к странице будет проверен статус и в версию строки будет установлен бит подсказки xmax_aborted.

Сам номер xmax при этом остается в странице, но смотреть на него уже никто не будет.

Но, что делать с неиспользуемыми записями???

Vacuum

Неиспользуемые в рамках незавершенных транзакций строки называются **мертвыми (dead)**.
Они, равно как и соответствующим им записи в индексах, будут присутствовать в БД вплоть да принудительного удаления – т.н. vacuum.

Неиспользуемые строки:
• занимают место в памяти;
• занимают место на диски;
• участвуют в select и update where;
• достаточно ощутимо и негативно влияют на производительность.

Vacuum

Варианты использования:

- vacuum
- vacuum verbose
- vacuum full

За выполнением можно следить через:

- консоль
- pg_stat_progress_vacuum

Vacuum

```
SELECT name, setting, context, short_desc
        FROM pg_settings WHERE name like 'vacuum%';
```

Основные параметры:

- vacuum_cost_ delay – по умолчанию 0
- vacuum_cost_limit – по умолчанию 200
- vacuum_cost_page_hit – по умолчанию 1
- vacuum_cost_page_miss – по умолчанию 10 / рекомендуется 5
- vacuum_cost_page_dirty – по умолчанию 20 / рекомендуется 10

Vacuum / Autovacuum

В современных версиях PostgreSQL (начиная с 9-х) есть autovacuum.
Его можно и нужно использовать.
Ручное использование vacuum еще возможно но не рассматривайте его как альтернативу autovacuum.

Когда еще можно использовать vacuum (без использования autovacuum):
• если «пролетели» с настройками autovacuum;
• использование PostgreSQL в режиме DWH хранилища.

Autovacuum

Обязательно надо использовать !!!

Настраивается через параметры категории Autovacuum.
Инициируется выделенным фоновым процессом в зависимости от различных порогов срабатывания, которые задаются как на уровне кластера так и на уровне отдельных объектов.

Autovacuum

SELECT name, setting, context, short_desc
FROM pg_settings WHERE name like 'autovacuum%';

Основные параметры:

- autovacuum_vacuum_cost_ delay – если -1, то берется значение vacuum_cost_ delay
- autovacuum_vacuum_cost_limit – если -1, то берется значение vacuum_cost_limit
- autovacuum_max_worker = 3, рекомендуется увеличить
- autovacuum_vacuum_threshold = 50 (абсолютное значение)
- autovacuum_vacuum_scale_factor = 0.2 (определяет долю строк в таблице)

формула срабатывания очистки:

Количество мертвых строк (pg_stat_user_tables.n_dead_tup)
>= *autovacuum_vacuum_threshold* + *autovacuum_vacuum_scale_factor* *
pg_class.reltupes

Autovacuum

```
select c.relname,
current_setting('autovacuum_vacuum_threshold') as av_base_thresh,
current_setting('autovacuum_vacuum_scale_factor') as av_scale_factor,
(current_setting('autovacuum_vacuum_threshold')::int +
(current_setting('autovacuum_vacuum_scale_factor')::float * c.reltuples)) as av_thresh,
s.n_dead_tup
        from pg_stat_user_tables s join pg_class c ON s.relname = c.relname
where s.n_dead_tup > (current_setting('autovacuum_vacuum_threshold')::int
+ (current_setting('autovacuum_vacuum_scale_factor')::float * c.reltuples));
```

Пример настройки автовакуума в Amazon RDS для PostgreSQL

Autovacuum

Желательно настаивать максимально агрессивно

- log_autovacuum_min_duration = 0
- autovacuum_max_workers = 10
- autovacuum_naptime = 15s
- autovacuum_vacuum_threshold = 25
- autovacuum_vacuum_scale_factor = 0.05
- autovacuum_vacuum_cost_delay = 10
- autovacuum_vacuum_cost_limit = 1000

Но не забывать что autovacuum может стать источником проблем, например, высокий ввод-вывод

Autovacuum

Для отдельных таблиц можно:

1. Совсем отключить автовакуум

ALTER TABLE table_name set (autovacuum_enabled = off);

2. Установить персональные значения

ALTER TABLE table_name set (autovacuum_vacuum_threshold = 100);

Заморозка

Под номер транзакции в PostgreSQL выделено 32 бита (**около 4 млрд**)

При активной работе сервера - нагрузке 1000 транзакций в секунду переполнение произойдёт всего через **???**

Заморозка

Под номер транзакции в PostgreSQL выделено 32 бита (**около 4 млрд**)

При активной работе сервера - нагрузке 1000 транзакций в секунду переполнение произойдёт всего через **полтора месяца непрерывной работы.**

Заморозка

Механизм многоверсионности полагается на последовательность нумерации — тогда из двух транзакций **транзакцию** с **меньшим** номером можно считать **начавшейся раньше**.

Поэтому нельзя просто так **обнулить** счетчик и продолжить нумерацию **заново**.

Заморозка

Почему под номер транзакции не выделено 64 бита — ведь это полностью исключило бы проблему?

Заморозка

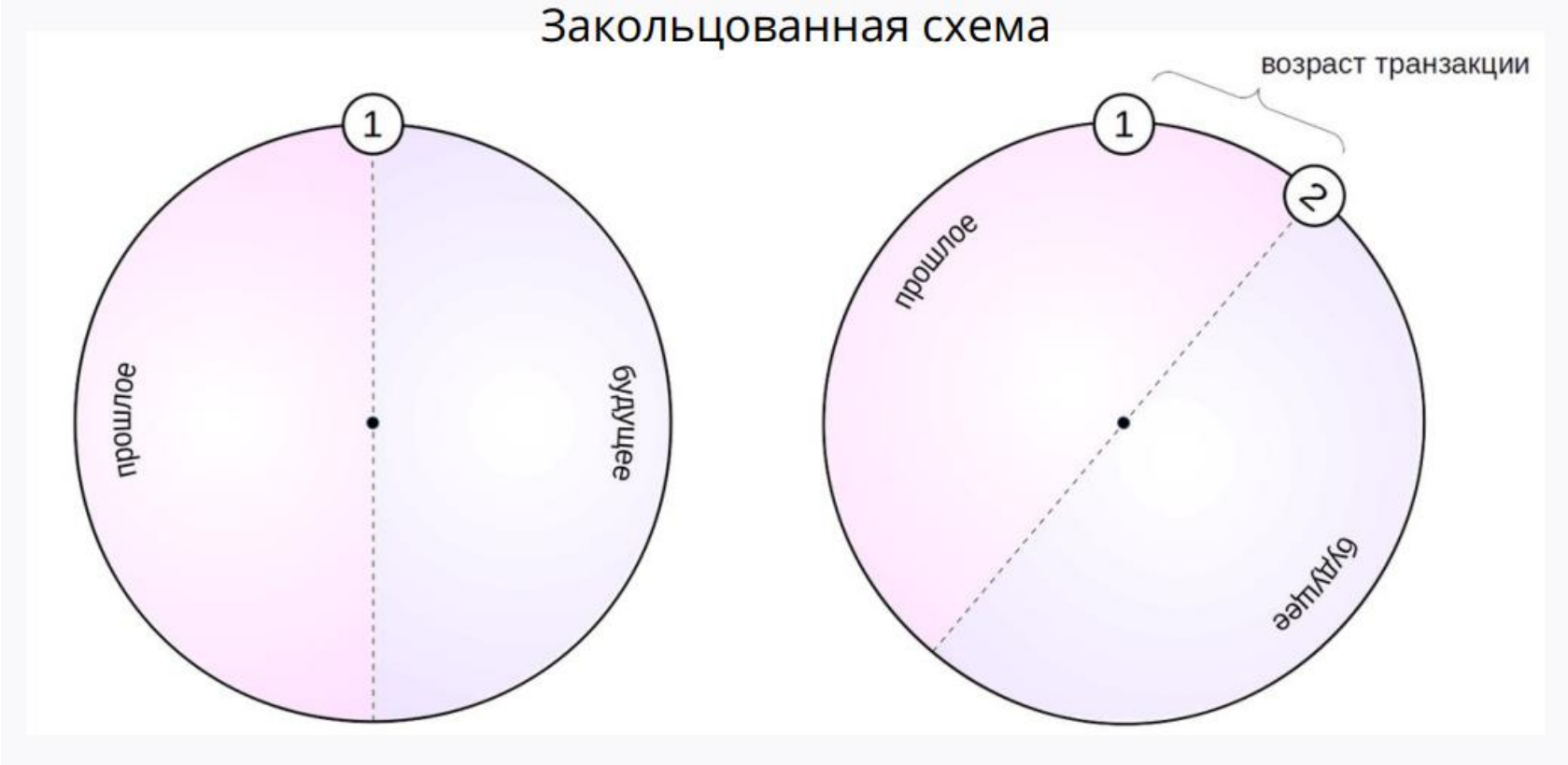
Почему под номер транзакции не выделено 64 бита — ведь это полностью исключило бы проблему?

Дело в том, что в заголовке каждой версии строки хранятся два номера транзакций — xmin и xmax. Заголовок и так достаточно большой, минимум 23 байта, а увеличение разрядности привело бы к его увеличению еще на 8 байт. А это уже много.

В платной версии PostgresPro есть вариант с 64 битными транзакциями.

Что делать???

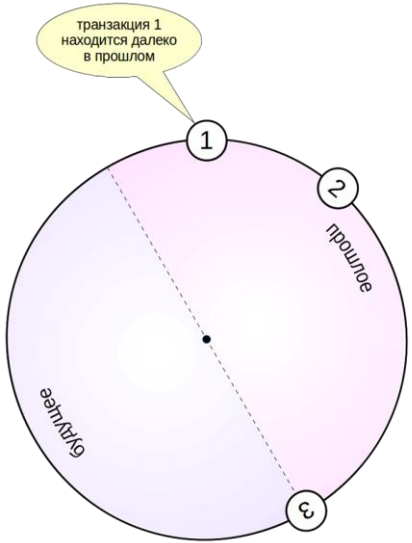
Заморозка

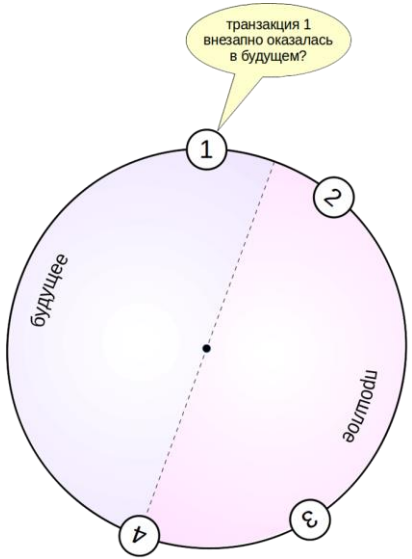

Закольцованная схема
возраст транзакции
1
прошлое
будущее
1
2
прошлое
будущее

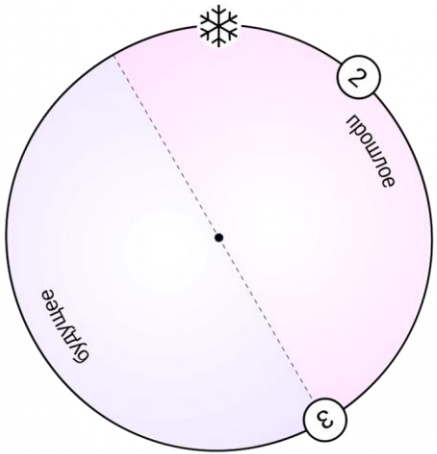
Заморозка

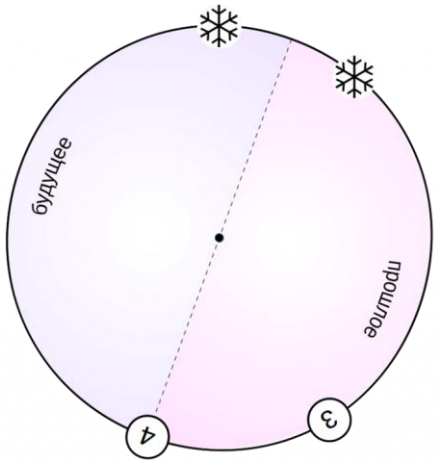
Какие могут быть проблемы???

Заморозка


транзакция 1
находится далеко
в прошлом
1
2
3
прошлое
будущее


транзакция 1
внезапно оказалась
в будущем?
1
2
3
4
будущее
прошлое


2
3
прошлое
будущее


3
4
будущее
прошлое

Заморозка

Для того, чтобы пометить номер транзакции xmin как замороженный, выставляются одновременно оба бита-подсказки — бит фиксации и бит отмены.

Одна из задач автовакуума пробегать и замораживать старые строки.
Замороженная версия строки считается старше любых обычных данных и всегда видна во всех снимках данных.

Заморозка

3 параметра настройки:

SHOW vacuum_freeze_min_age;
плохо делать очень маленьким для активно меняющихся данных - двойная работа автовакуума

SHOW vacuum_freeze_table_age;
вакуум пробегает полностью заполненные и неменяющиеся странички, куда обычно не заходит

SHOW autovacuum_freeze_max_age;
для аварийной заморозки

Рефлексия

Рефлексия

1. Зачем нужен вакуум ?
2. Назовите способы реализации ACID ?
3. Какая из команд insert, delete, update самая медленная и почему ?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Коробков Виктор