Восстановление повреждённых данных в СУБД PostgreSQL

Евгений Бредня, Postgres Professional

PGConf.Russia 2023

PostgresPro

О чём

- Как спасать повреждённые данные из СУБД постгрес.
- Нет и не может быть готовых рецептов для восстановления повреждённых данных, иначе они давно уже были бы реализованы в виде утилит.
- Требует высокой квалификации: нужно знать как работает постгрес, как работает MVCC, где и в какой форме хранятся данные на диске, глубокие и широкие знания в предметной области.
- Долгий и кропотливый труд, который далеко не всегда приводит к успеху.
- Мотивация: поделиться опытом, который можно будет использовать.

Disclaimer

- Дальше речь пойдёт о вмешательстве в работу СУБД и в структуры, с которыми работает СУБД.
- **Любое неверное движение приведёт к частичной или полной потере данных.**
- **Если Вы вмешиваетесь в работу СУБД, то вся ответственность за возможную потерю данных лежит исключительно на Вас!**
- Не следует даже пытаться делать что-либо без полного понимания последствий своих действий, лучше позовите на помощь.

* Красным цветом тут и далее помечено особенно опасное или особенно важное

Пара слов о любви

PostgresPro

Пара слов о любви

...к бекапам!

PostgresPro

Причины повреждения данных

- Сложность современной инфраструктуры: количество уровней абстракции между программой и оборудованием давно превысило все мыслимые пределы и не поддаётся осмыслению.
- Сбои оборудования, под цифровыми технологиями лежат физические явления.
- Ошибки системного ПО, систем виртуализации, библиотек, файловых систем, драйверов, firmware, BIOS, микрокода и т. п.
- Ошибки конфигурации и эксплуатации программно-аппаратных комплексов.
- Ошибки прикладного ПО, в том числе СУБД.
- Человеческий фактор — ошибки пользователей и администраторов.
- Дублирование критичных частей, контрольные суммы, коды исправления ошибок и т.п.

Один предусмотренный заранее сбой обычно получается пережить без особых последствий, два наложившихся друг на друга сбоя — почти всегда фатальны.

Резервное копирование

Если вам дороги ваши данные, делайте их резервные копии!

PostgresPro

Что считать повреждением данных

Повреждения могут быть логическими и физическими:

- **Логические повреждения** — есть логическое несоответствие между связанными данными в базе.
- **Физические повреждения** — нарушения в структуре файлов данных в PGDATA, в том числе служебных.
- База с физическими повреждениями является **неконсистентной**.

Любые физические повреждения опасны, так как могут привести к некорректной работе СУБД и неожиданной потере данных, их следует устранять.

После устранения физических повреждений обязательно нужно предусмотреть логическую выверку данных.

Признаки физических повреждений

- ошибка контрольных сумм при чтении блока
- cannot open/read/write file/block
- invalid [status of] xid/mxid/xmin/xmax 123456
- странные номера транзакций из будущего или глубокого прошлого
- missing chunk 0 for TOAST value
- странное поведение процессов постгреса
- и так далее

PostgresPro

Что делать?

- Этап 1. Диагностика и анализ
  - восстановить картину инцидента (что случилось)
  - найти корневую причину (почему случилось)
  - найти все повреждения (что повредилось)
- Этап 2. Восстановление данных
  - план
  - выполнение плана

Что делать? (2)

- Не надо ничего исправлять до конца диагностики!
- Увлёкшись починкой первой попавшейся ошибки, мы рискуем не заметить более важную проблему.
- Из-за преждевременных действий можем потерять информацию, которая помогла бы нам восстановить картину инцидента и установить корневую причину.
- Не восстановив картину повреждений и не поняв корневой причины, нельзя понять, что повреждено и нужно исправить.
- Приступать к диагностике надо немедленно!
- Если диагностика затягивается, а проблема критична, то параллельно ищем обходные пути (work-around).

Корневая причина — это важно!

Найти корневую причину важнее, чем восстановить данные!

- Если причина один раз привела к неприятностям, и она не устранена, то неприятности будут продолжаться.
- Не понимая причины невозможно судить о последствиях: велик риск, что часть повреждений останется незамеченными, и это приведёт к ещё большим неприятностям в будущем. Например, к более значительной или полной потере данных.
- Устранение корневой причины должно быть одним из пунктов плана по восстановлению. Другими словами, пока не нашли причину, к действиям по устранению не приступаем.

PostgresPro

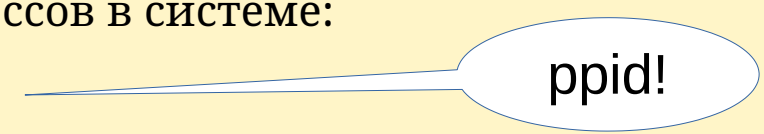
Э1. Диагностика: инструментарий

PostgresPro

Э1. Диагностика: инструментарий (1)

- Анализ работающих процессов в системе:

```
ps -ef | grep postgres
```




```
top/atop
```

посмотреть список процессов, утилизация системных ресурсов: CPU, RAM, IO

```
gdb
```

посмотреть back trace; анализ core dump; убедиться, что процесс работает

```
perf
```

чем заняты отдельные процессы или вся система.

- Результаты:
  - не работают ли два разных постгреса поверх одной PGDATA?
  - аномалии в поведении системы и постгреса
  - изучаем работу аномального процесса

PostgresPro

Э1. Диагностика: инструментарий (2)

- Анализ текстовых журналов (log) постгреса:

```
grep -E '(ERROR|FATAL|PANIC)' postgresql*.log
```

```
cat postgresql*.log \
| grep -v 'connected' \
| grep -v … \
| less
```

- Анализ журналов ОС:

```
dmesg
```

```
journalctl
```

- Результаты:
  - ошибки и их характер
  - хронология событий
  - какие данные повреждены
  - получить список таблиц, где появлялись ошибки в процессе работы СУБД

Э1. Диагностика: инструментарий (3)

- Проверка, читается ли таблица/база:

`pg_dump`

+ быстрая проверка, выгрузит данные(!), может работать в несколько потоков

```
select ctid, * from bad_table
 where ctid>'(100,0)' and ctid<'(200,0)';
```

+ быстрая проверка, можно интерактивно исследовать нечитаемые данные, осторожно с индексами!

```
select molotilka('bad_table', 0);
```

+ прочитает все строки в каждом блоке из таблицы, покажет нечитаемые блоки и строки.

- Результаты:
  - уточняем список повреждённых таблиц
  - локализация повреждённых данных в каждой повреждённой таблице

Э1. Диагностика: инструментарий (4)

```
CREATE or REPLACE FUNCTION molotilka(tbl regclass, start_page bigint) RETURNS bigint AS $$
DECLARE
    n_pages integer;      page integer;
    lps_in_page integer;  lp_p integer;
    err_count bigint = 0;
BEGIN
  SELECT pg_relation_size(tbl) / 8192 INTO n_pages;
  FOR page IN start_page .. n_pages-1 LOOP
    BEGIN
      SELECT coalesce(max(lp),0) from heap_page_items(get_raw_page(tbl::text, page)) INTO lps_in_page;
      FOR lp_p IN 1 .. lps_in_page LOOP
         BEGIN
           EXECUTE format('select row(t.*) from %s as t where ctid=''(%s,%s)''', tbl::text, page, lp_p);
         EXCEPTION WHEN OTHERS THEN
           err_count = err_count + 1;
           RAISE NOTICE 'TUPLE ERROR: ctid=(%,%), SQLSTATE=% DETAIL=%', page, lp_p, SQLSTATE, SQLERRM;
         END;
      END LOOP;
    EXCEPTION WHEN OTHERS THEN
      err_count = err_count + 1;
      RAISE NOTICE 'PAGE ERROR: page=%, SQLSTATE=%, DETAIL=%', page, SQLSTATE, SQLERRM;
    END;
  END LOOP;
  RETURN err_count;
END $$ language plpgsql;
```

PostgresPro

Э1. Диагностика: инструментарий (5)

- Что творится в блоках данных:

```
pg_filedump -i file
```

  в строках xmin, xmax, флаги
  в заголовке блока есть LSN — когда блок был изменён в последний раз.

- Как это получилось:

```
pg_waldump
```

  позволит посмотреть по WAL, какие данные и как менялись в транзакции и блоке, практически единственная возможность поймать ошибку в коде.

- По номеру транзакции или id можно найти другие записи с близкими номерами, посмотреть timestamp этих записей — даст приблизительное время инцидента.

- Результаты:
  - как выглядят данные с точки зрения постгреса
  - хронология событий на уровне изменения данных
  - характер повреждений и локализация повреждений

PostgresPro

Окончание диагностики

Диагностику и анализ можно заканчивать, если удалось:

- восстановить полную картину случившегося,
- установить корневую причину повреждений,
- получить полную информацию о повреждениях, что надо исправлять.

Можем переходить к восстановлению данных!

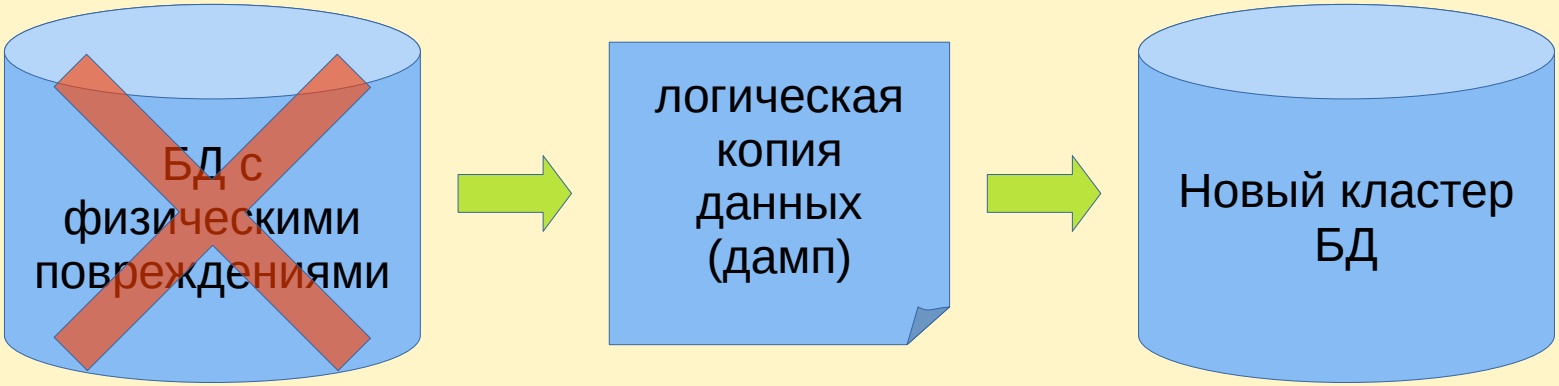
Если во время восстановительных работ обнаруживаются какие-либо неожиданности, то возвращаемся на этап диагностики и анализа.

До начала действий (1)

- Убрать нагрузку (пользователей) — pg_hba.conf или PGPORT.
- Сделать холодную(?) резервную копию.
- На неисправном оборудовании заниматься восстановлением нет смысла.
- Надо оценить целесообразность восстановления данных:
  - сколько времени это займёт?
  - будут ли восстановленные данные полезны?
- План восстановления должен гарантировать, что в базе не останется физических повреждений. *Как этого добиться?*

До начала действий (2)

Как сделать все таблицы и другие служебные файлы в PGDATA корректными?
Вручную это сделать совершенно нереально. Пусть постгрес сделает это сам!



Цель восстановительных работ — выгрузить данные в дамп для последующей загрузки в новый кластер!

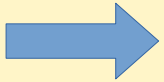
PostgresPro

До начала действий (3)

- Из базы выгружаем только содержимое пользовательских таблиц, остальное обычно можно восстановить другими способами.
- План должен включать и восстановление данных, и устранение причины повреждений, иначе данные опять будут повреждены.
- После устранения физических повреждений всегда следует проверить логическую целостность восстановленных данных.

Если что-то из ранее рассмотренного непонятно — лучше позвать на помощь. Дальше будут опасные действия.

Э2. План восстановления

1. 
2. 
3. 

PostgresPro

Э2. Восстановление (1)

- Нет и не может быть общих рекомендаций, они всегда зависят от текущего состояния БД, инцидента и корневых причин проблемы.
- Условно можно поделить на две больших области:
  - СУБД не запускается,
  - СУБД запускается, но повреждены пользовательские данные.
- Запускайте постгрес через `pg_ctl`, чтобы сразу видеть все сообщения об ошибках при запуске.

Э2. Восстановление (2)

Если не запускается СУБД:

- Смотрим на сообщения об ошибках, устраняем препятствия, повторяем до победного конца.
- Если применены деструктивные действия или признаки неконсистентности, то базу больше использовать нельзя — данные надо выгрузить и загрузить в заново инициализированный кластер.
- Серьёзные проблемы невозможности запуска часто связаны с
  - сильным повреждением системного словаря или
  - повреждением WAL.

Э2. Восстановление, словарь

- Повреждения системных индексов — можно запустить как:

```
postgres -P ...
```

- Малозначащие (для спасения данных) таблицы можно взять от другого кластера.
- pg_class и pg_attribute — может быть сохранились на реплике или в бекапе. Если не было изменений в структуре таблиц (DDL), то может помочь.
- Полная потеря системного словаря — увы, база потеряна. В лучшем случае получится восстановить что-то вручную из отдельных файлов данных с помощью pg_filedump, см. «последний шанс» (далее).
- Поможет только pg_dump/restore и скорее всего отдельных таблиц.

Э2. Восстановление, WAL (1)

- При старте СУБД делается проверка на необходимость восстановления БД до консистентного состояния по журналам WAL. Если сервер не смог запуститься из-за нехватки WAL, это обозначает, что база в неконсистентном состоянии.
- Проблемы с WAL:
  - есть повреждения в WAL, которые не позволяют его проиграть,
  - не найден необходимый WAL.
- Архив WAL (`archive_mode=on`) или реплики могут помочь, WAL в кластере везде одинаковый.

Э2. Восстановление, WAL (2)

- Утилитой pg_resetwal можно отключить механизм восстановления — база запустится, но останется в неконсистентном состоянии, то есть **гарантированно** повреждённой.
- Утилита pg_resetwal позволит экземпляру постгреса запуститься, даже если половины базы уже нет, а вторая половина является бессмысленным набором байтов.
- Никогда не используйте pg_resetwal, если не понимаете совершенно точно, что он делает, и какие будут последствия.

После применения pg_resetwal база непригодна для использования, из неё можно только выгрузить данные для последующей загрузки в новый кластер: только полный pg_dump/restore!

Э2. Восстановление таблиц (1)

- Это наиболее частый вид работ по восстановлению данных.
- Найденная ранее корневая причина должна помочь уточнить, где и как повреждены данные, выбрать оптимальную стратегию восстановления.
- Если сталкиваемся с неожиданностями — возвращаемся на этап диагностики и анализа.
- Если нет *полной* уверенности, что остальные данные не повреждены — всегда восстанавливаем все данные в новую базу через pg_dump/restore.
- Если точно знаем, что повреждена только одна таблица, то можно восстановить только её: pg_dump/restore bad_table.

Э2. Восстановление таблиц (2)

- В логах постгреса ошибки вида (в данном случае Postgres Pro Enterprise):

```
ERROR: could not access status of transaction 11416767659
DETAIL: Could not open file "pg_xact/0000000000AA": No such file or directory.
```

- Создать файл статусов транзакций:

```
dd if=/dev/zero of=PGDATA/pg_xact/0000000000AA bs=1K count=256
```

- В зависимости от xmin/xmax может появиться лишняя или пропасть нужная строка в таблице, могут появиться дубликаты уникальных ключей и т. п. В базе могут оказаться противоречивые данные — в одних строках данных транзакция может быть помечена как зафиксированная, в других как отменённая. То есть база становится неконсистентной вплоть до невозможности дальнейших работ по восстановлению.
- Кажущаяся простота *устранения сообщения* об ошибке не должна вводить в заблуждение — база имеет физические повреждения, которые надо устранить: ищем причину повреждений, делаем **полный pg_dump/resore всей базы** и логическую выверку данных после восстановления!

Э2. Восстановление таблиц (3)

- контрольную сумму в блоке можно игнорировать, параметр `ignore_checksum_failure=on`
- Если блок не читается совсем, его можно обнулить — данные в блоке будут потеряны. Пример команды (не забываем про сегменты!), выполнять надо на остановленной базе:

```
dd if=/dev/zero of=/PGDATA/base/db_oid/bad_table_filenodeid bs=8k
seek=XXXXX count=1 conv=notrunc
```

осторожно: команда физически и необратимо удаляет данные!

- Можно автоматически обнулять повреждённые блоки, (может быть полезно, если их много), параметр `zero_damaged_pages=on`
- После того, как таблица(-ы) стала читаться: pg_dump/restore и логическая выверка данных.

Э2. Восстановление таблиц (4)

- Если блок читается, но не читается отдельная строка в блоке, её можно удалить по ctid:

```
delete from bad_table where ctid='(123,1)';
```

- С 14 версии в contrib появился модуль **pg_surgery**, который позволяет сделать отдельную строку видимой или наоборот удалить её по tid.
- Если не читается только отдельное поле (например, TOAST), то перед удалением можно посмотреть и сохранить значения всех читаемых полей.
- После удаления всех нечитаемых строк и блоков, таблицу **нужно** пересоздать физически: **pg_dump/restore**.
- Если восстановить надо только одну таблицу в базе, то возможны варианты:

```
pg_dump -t bad_table
```

`vacuum full bad_table;` — может быть полезно при ссылочной целостности

```
create table good_table as select * from bad_table;
```

PostgresPro

Э2. Восстановление таблиц (5)

- «Последний шанс» — полностью ручное восстановление данных из файла постгреса с помощью pg_filedump:
  - если системный словарь безнадёжно повреждён,
  - или база не запускается вообще никак,
  - или остался вообще только один файл данных или только часть такого файла.

```
pg_filedump -i -D int,text path/to/datafile/12345
```

- Нужно знать какие типы полей есть в таблице, и в каком порядке они идут.
- Придётся сделать полный ручной разбор всех строк в MVCC — в выводе будут присутствовать все «мёртвые» версии каждой строки, которые будут найдены в блоках данных.
- Это действительно «последний шанс», так реально восстановить только совершенно бесценные данные.

Рекомендации

- Делайте бекапы! Всего ранее рассказанного можно избежать, если есть нормальное резервное копирование.
- Всегда (всегда!) включайте контрольные суммы: `initdb -k`
- Архив WAL часто бывает полезен.
- Готовьте систему к диагностике заранее: ставьте пакеты с отладочными символами постгреса, gdb, perf, atop, pg_filedump, настраивайте сбор core dump.
- Мониторинг — это хорошо.
- Делайте бекапы.

Конец

Спасибо за внимание

Берегите свои данные!

PostgresPro