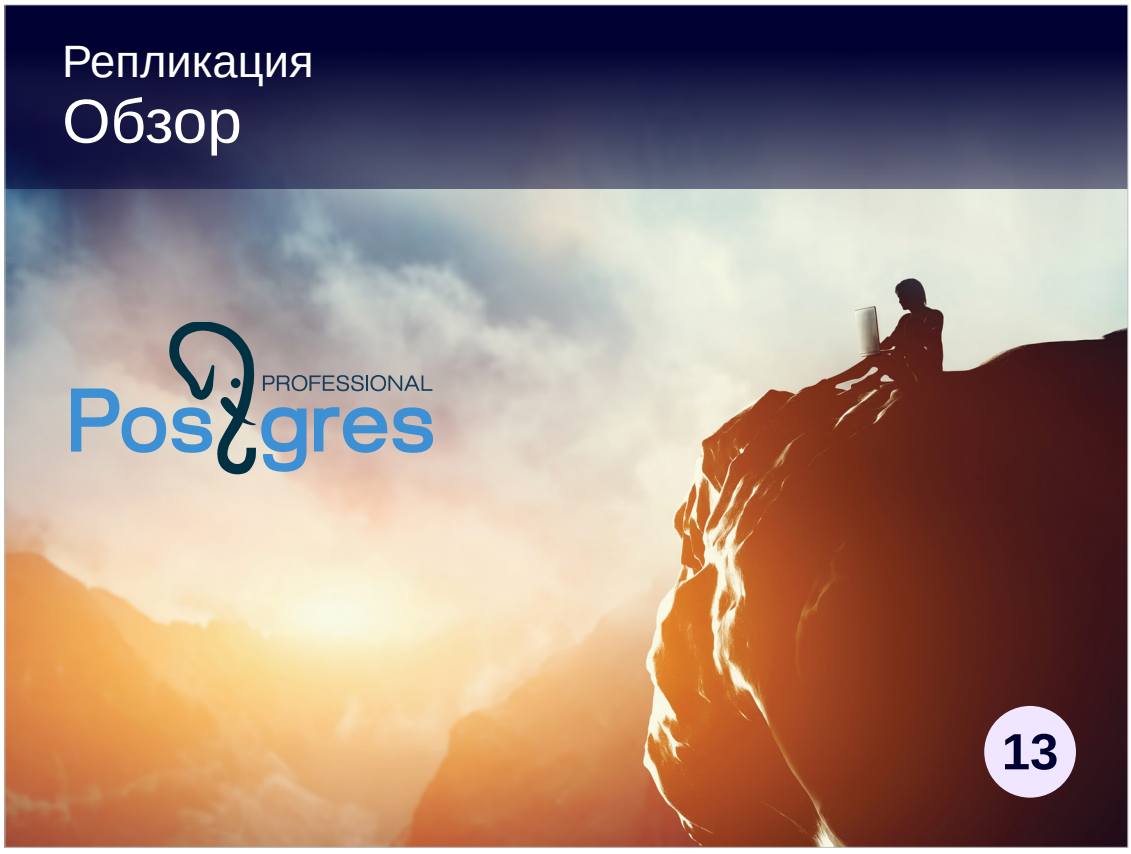


Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Задачи и виды репликации

Физическая репликация

Логическая репликация

Варианты использования механизма репликации

Задачи репликации

Репликация

процесс синхронизации нескольких копий кластера баз данных на разных серверах

Задачи

отказоустойчивость	при выходе из строя одного из серверов система должна сохранить доступность (возможна деградация производительности)
масштабируемость	распределение нагрузки между серверами

Одиночный сервер, управляющий базами данных, может не удовлетворять предъявляемым требованиям.

Во-первых, отказоустойчивость: один физический сервер — это возможная точка отказа. Если сервер выходит из строя, система становится недоступной.

Во-вторых, производительность. Один сервер может не справляться с нагрузкой. Зачастую наращиванию мощности сервера предпочтительна возможность масштабирования — распределения нагрузки на несколько серверов.

Таким образом, речь идет о том, чтобы иметь несколько серверов, работающих над одними и теми же базами данных. Под репликацией понимается процесс синхронизации этих серверов.

Виды репликации

Физическая

- мастер-реплика: поток данных только в одну сторону
- трансляция потока журнальных записей или файлов журнала
- требуется двоичная совместимость серверов
- возможна репликация только всего кластера

Логическая

- публикация-подписка: поток данных возможен в обе стороны
- информация о строках (уровень журнала logical)
- требуется совместимость на уровне протокола
- возможна выборочная репликация отдельных таблиц

Взаимодействие нескольких серверов можно организовать по-разному. Два основных подхода, доступных в PostgreSQL — физическая и логическая репликация.

При физической репликации серверы имеют назначенные роли: мастер и реплика. Мастер передает на реплику журнальные записи (в виде файлов или потока записей); реплика применяет эти записи к своим файлам данных. Применение происходит чисто механически, без «понимания смысла» изменений, поэтому важна двоичная совместимость между серверами (одинаковые платформы и основные версии PostgreSQL). Поскольку журнал общий для всего кластера, то и реплицировать можно только кластер целиком.

При логической репликации в журнал добавляется информация более высокого уровня, позволяющая реплике разобраться в изменениях на уровне строк таблиц (параметр *wal_level* = logical). Для такой репликации не нужна двоичная совместимость, реплика должна лишь уметь декодировать содержащуюся в журнале логическую информацию. Логическая репликация позволяет при необходимости проигрывать не все изменения, а только касающиеся отдельных таблиц.

Логическая репликация доступна, начиная с версии 10; более ранние версии должны были использовать расширение pg_logical, либо организовывать репликацию с помощью триггеров.

Физическая репликация

Схема работы репликации

Способы доставки журналов

Использование реплики

Переключение на реплику (и обратно)

Варианты настройки и использования репликации

Сначала познакомимся с физической репликацией.

Механизм ее работы — передача на реплику изменений в виде записей журнала предзаписи. Это очень эффективный механизм, но он требует, чтобы между серверами была двоичная совместимость (основная версия сервера, операционная система, аппаратная платформа).

Физическая репликация всегда однонаправлена: в ней может существовать только один мастер (и произвольное число реплик).

Репликация

Резервная копия

- базовая резервная копия — pg_basebackup
- архив файлов журнала предзаписи

Непрерывное восстановление

- разворачиваем резервную копию,
  задаем конфигурационные параметры,
  создаем сигнальный файл standby.signal
  и запускаем сервер
- сервер восстанавливает согласованность
  и продолжает применять поступающие журналы
- доставка — поток по протоколу репликации или архив WAL
- подключения (только для чтения) разрешаются
  сразу после восстановления согласованности

Настройка репликации выполняется очень похоже на настройку физического резервного копирования. Отличие в том, что резервная копия поднимается сразу, не дожидаясь поломки основного сервера, и работает в режиме постоянного восстановления: все время читает и применят новые сегменты WAL, приходящие с мастера. Для этого вместо сигнального файла recovery.signal создается другой файл, standby.signal.

Таким образом, реплика постоянно поддерживается в почти актуальном состоянии и в случае сбоя мы имеем сервер, готовый подхватить работу.

По умолчанию реплика работает в режиме «горячего резерва» — в процессе восстановления допускает подключения для чтения данных (как только будет восстановлена согласованность). Можно запретить подключения, тогда реплика называется сервером «теплого резерва».

В отличие от резервной копии, репликация не позволяет восстановиться на произвольный момент в прошлом. Иными словами, репликацию невозможно использовать, чтобы исправить допущенную ошибку (хотя есть возможность настроить репликацию так, чтобы она отставала от мастера на определенное время).

Использование реплики

Допускаются

- запросы на чтение данных (select, copy to, курсоры)
- установка параметров сервера (set, reset)
- управление транзакциями (begin, commit, rollback...)
- создание резервной копии (pg_basebackup)

Не допускаются

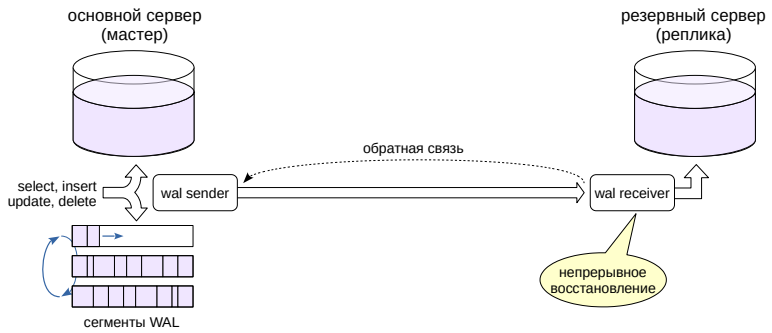
- любые изменения (insert, update, delete, truncate, nextval...)
- блокировки, предполагающие изменение (select for update...)
- команды DDL (create, drop...), в том числе создание временных таблиц
- команды сопровождения (vacuum, analyze, reindex...)
- управление доступом (grant, revoke...)
- не срабатывают триггеры и пользовательские (advisory) блокировки

В режиме горячего резерва на реплике не допускаются никакие изменения данных (включая последовательности), блокировки, команды DDL, такие служебные команды, как VACUUM и ANALYZE, команды управления доступом — словом, все, что так или иначе изменяет данные.

При этом реплика может выполнять запросы на чтение данных. Также будет работать установка параметров сервера и команды управления транзакциями — например, можно начать (читающую) транзакцию с нужным уровнем изоляции.

Кроме того, реплику можно использовать и для изготовления резервных копий (конечно, принимая во внимание возможное отставание от мастера).

Потоковая репликация



Есть два способа доставки журналов от мастера к реплике. Основной, который используется на практике — потоковая репликация.

В этом случае реплика подключается к мастеру по протоколу репликации и читает поток записей WAL. За счет этого при потоковой репликации отставание реплики сведено к минимуму, и даже к нулю при синхронном режиме.

Тонкий момент. Очистка на основном сервере может удалить версии строк, которые нужны для снимка запроса на реплике. Пострадавший запрос на реплике в таком случае придется отменить. Эта проблема решается в случае потоковой репликации с помощью специального механизма *обратной связи:* мастер будет знать, какой номер транзакции нужен для снимка данных на реплике и отложит очистку.

Физическая репликация. Резервная копия мастера

Настройки по умолчанию позволяют использовать протокол репликации:

- wal_level = replica,
- max_wal_senders = 10,
- разрешение на подключение в pg_hba.conf.

Создадим автономную резервную копию. С ключом -R утилита сформирует необходимые для репликации конфигурационные параметры.

```
student$ sudo rm -rf /home/student/basebackup
```

```
student$ pg_basebackup --pgdata=/home/student/basebackup -R
```

Убеждаемся, что второй сервер остановлен, и выкладываем автономную копию в его каталог данных. Владельцем файлов должен быть пользователь postgres.

```
student$ sudo pg_ctlcluster 13 replica status
```

```
pg_ctl: no server running
```

```
student$ sudo rm -rf /var/lib/postgresql/13/replica
```

```
student$ sudo mv /home/student/basebackup/ /var/lib/postgresql/13/replica
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/replica
```

Реплика

Утилита pg_basebackup добавила в postgresql.auto.conf параметры соединения с основным сервером:

```
student$ sudo cat /var/lib/postgresql/13/replica/postgresql.auto.conf
```

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=student passfile=''/home/student/.pgpass'' channel_binding=prefer host=''/var/run/postgresql'' port=5432 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
```

А также создала пустой файл standby.signal, который даст указание серверу войти в режим постоянного восстановления.

```
student$ sudo ls -l /var/lib/postgresql/13/replica/standby.signal
```

```
-rw------- 1 postgres postgres 0 мая 11 13:18 /var/lib/postgresql/13/replica/standby.signal
```

Можно запускать сервер.

```
student$ sudo pg_ctlcluster 13 replica start
```

Посмотрим на процессы реплики.

```
student$ sudo head -n 1 /var/lib/postgresql/13/replica/postmaster.pid
```

```
41007
```

```
student$ sudo ps -o pid,command --ppid 41007
```

```
    PID COMMAND
  41008 postgres: 13/replica: startup waiting for 000000010000000000000008
  41010 postgres: 13/replica: checkpointer
  41011 postgres: 13/replica: background writer
  41012 postgres: 13/replica: stats collector
  41013 postgres: 13/replica: walreceiver
```

Процесс walreceiver принимает поток журнальных записей, процесс startup применяет изменения.

И сравним с процессами мастера.

```
student$ sudo head -n 1 /var/lib/postgresql/13/main/postmaster.pid
```

```
27355
```

```
student$ sudo ps -o pid,command --ppid 27355
```

```
    PID COMMAND
  27357 postgres: 13/main: checkpointer
  27358 postgres: 13/main: background writer
  27359 postgres: 13/main: walwriter
  27360 postgres: 13/main: autovacuum launcher
  27361 postgres: 13/main: stats collector
  27362 postgres: 13/main: logical replication launcher
  40739 postgres: 13/main: student student [local] idle
  41014 postgres: 13/main: walsender student [local] idle
```

Здесь добавился процесс walsender.

Проверка репликации

Состояние репликации можно смотреть в представлении на мастере:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5432
```

```
=> SELECT * FROM pg_stat_replication \gx
```

```
-[ RECORD 1 ]----+------------------------------
pid              | 41014
usesysid         | 16384
usename          | student
application_name | 13/replica
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2022-05-11 13:18:39.542675+03
backend_xmin     |
state            | streaming
sent_lsn         | 0/8000060
write_lsn        | 0/8000000
flush_lsn        | 0/8000000
replay_lsn       | 0/8000000
write_lag        | 00:00:00.140064
flush_lag        | 00:00:00.140064
replay_lag       | 00:00:00.140064
sync_priority    | 0
sync_state       | async
reply_time       | 2022-05-11 13:18:39.685398+03
```

Выполним несколько команд на мастере:

```
=> CREATE DATABASE replica_overview;
```

```
CREATE DATABASE
```

```
=> \c replica_overview;
```

```
You are now connected to database "replica_overview" as user "student".
```

```
=> CREATE TABLE test(id integer PRIMARY KEY, descr text);
```

```
CREATE TABLE
```

Проверим реплику:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_overview
```

```
=> SELECT * FROM test;
```

```
 id | descr
----+-------
(0 rows)
```

```
=> INSERT INTO test VALUES (1, 'Раз');

INSERT 0 1
```

```
=> SELECT * FROM test;
```

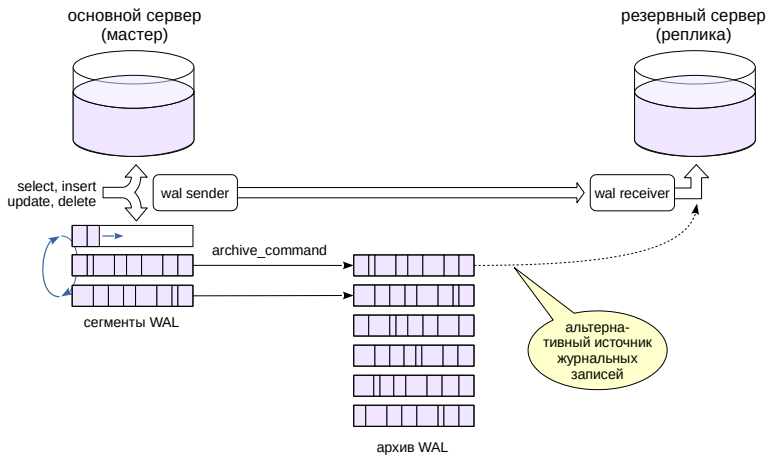
```
 id | descr
----+-------
  1 | Раз
(1 row)
```

При этом изменения на реплике не допускаются:

```
=> INSERT INTO test VALUES (2, 'Два');
```

```
ERROR:  cannot execute INSERT in a read-only transaction
```

Репликация с архивом



При использовании потоковой репликации есть опасность, что мастер удалит сегмент WAL, данные из которого еще не переданы на реплику. Для уверенности надо либо использовать слот репликации, либо применять потоковую репликацию вместе с архивом WAL (который все равно необходим для организации резервного копирования).

При использовании архива специальный процесс archiver на мастере записывает заполненные сегменты журнала с помощью команды *archive_command* (этот механизм рассматривается в модуле «Резервное копирование»).

Если реплика не сможет получить очередную журнальную запись по протоколу репликации, она попробует прочитать ее из архива с помощью команды из параметра *restore_command*.

В принципе, репликация может работать и с одним только архивом, без потоковой репликации. Но в этом случае:

- реплика вынужденно отстает от мастера на время заполнения сегмента;

- мастер ничего не знает о существовании реплики, поэтому очистка может удалить версии строк, нужные для снимка на реплике (можно настроить задержку применения конфликтующих записей, но далеко не всегда понятно, на какое время).

Переключение на реплику

Плановое переключение

останов основного сервера для технических работ без прерывания обслуживания

ручной режим

Аварийное переключение

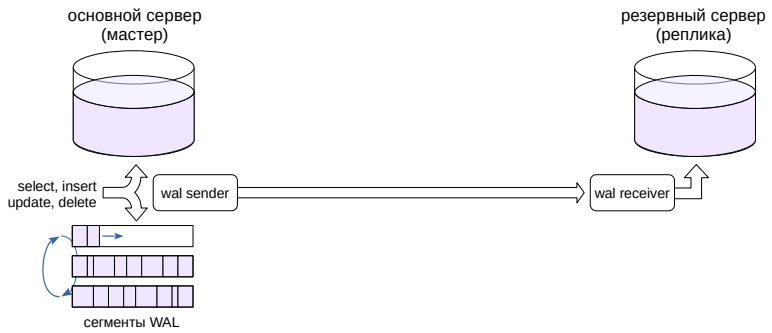
переход на реплику из-за сбоя основного сервера

ручной режим,
но в принципе можно автоматизировать с помощью дополнительного кластерного ПО

Причины перехода на резервный сервер бывают разные. Это может быть необходимость проведения технических работ на основном сервере — тогда переход выполняется в удобное время в штатном режиме. А может быть сбой основного сервера, и в таком случае переходить на резервный сервер нужно как можно быстрее, чтобы не прерывать обслуживание пользователей.

Даже в случае сбоя переход осуществляется вручную, так как PostgreSQL не имеет встроенного кластерного программного обеспечения (которое должно следить за состоянием серверов и инициировать переход).

Переключение на реплику



Переход на реплику в картинках: вначале имеются два сервера (мастер и реплика), между ними настроена репликация.

Переключение на реплику

В случае выхода основного сервера из строя или при штатной необходимости работ реплике дается команда прекратить восстановление и стать самостоятельным сервером, а бывший мастер отключается.

Конечно, требуется способ перенаправить пользователей на новый сервер; но это выполняется средствами вне PostgreSQL.

Переключение на реплику

Теперь переведем реплику из режима восстановления в обычный режим.

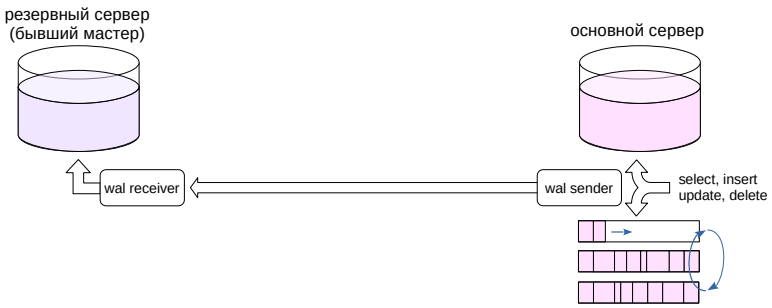
```
student$ sudo pg_ctlcluster 13 replica promote
```

```
  => INSERT INTO test VALUES (2, 'Два');
```

```
  INSERT 0 1
```

Получили два самостоятельных, никак не связанных друг с другом сервера.

Восстановление мастера



После восстановления бывшего основного сервера или окончания других работ на нем он подключается в качестве реплики к новому работающему мастеру.

Восстановление мастера

Простое подключение бывшего мастера — не работает

- проблема потери записей WAL, не попавших на реплику из-за задержки

Восстановление «с нуля» из резервной копии

- на месте бывшего мастера разворачивается абсолютно новая реплика
- процесс занимает много времени (отчасти можно ускорить rsync)

Утилита pg_rewind

- «откатывает» потерянные записи WAL, заменяя соответствующие страницы на диске страницами с нового мастера
- есть ряд ограничений, ограничивающий применение

Если переход на реплику произошел из-за аппаратуры (необходима замена дисков или сервера) или операционной системы (необходима переустановка ОС), то единственный вариант состоит в изготовлении абсолютно новой реплики на новом сервере.

Если же переход был штатным, полезен способ быстро вернуть старый мастер в строй (теперь уже в качестве реплики).

К сожалению, простой способ — подключить старый мастер к новому по репликационному протоколу — не работает. Причина в том, что из-за задержек в репликации часть записей WAL могла не дойти до реплики. Если на старом мастере остались такие записи, то применение записей с нового мастера приведет к повреждению базы.

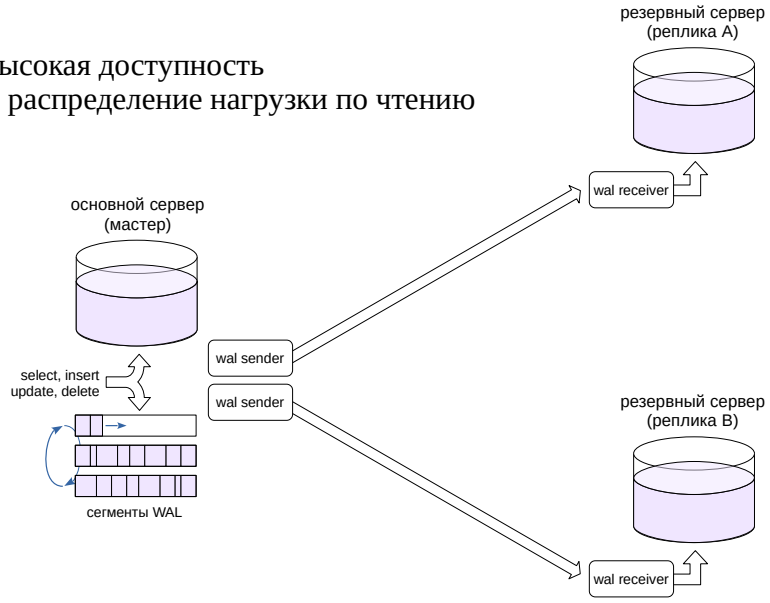
Всегда есть вариант создать абсолютно новую реплику путем изготовления базовой резервной копии. Однако для больших баз данных этот процесс может занимать много времени. Его можно ускорить с помощью rsync.

Еще более быстрый вариант состоит в использовании утилиты pg_rewind https://postgrespro.ru/docs/postgresql/13/app-pgrewind.

Утилита определяет записи WAL, которые не дошли до реплики (начиная с последней общей контрольной точки), и находит страницы, затронутые этими записями. Найденные страницы (которых должно быть немного) заменяются страницами с нового мастера. Кроме того, утилита копирует с сервера-источника (нового мастера) все служебные файлы. Дальше работает обычный процесс восстановления.

1. Несколько реплик

высокая доступность
и распределение нагрузки по чтению



Механизм репликации позволяет сконструировать систему так, чтобы она отвечала предъявляемым к ней требованиям. Рассмотрим несколько типичных задач и средства их решения.

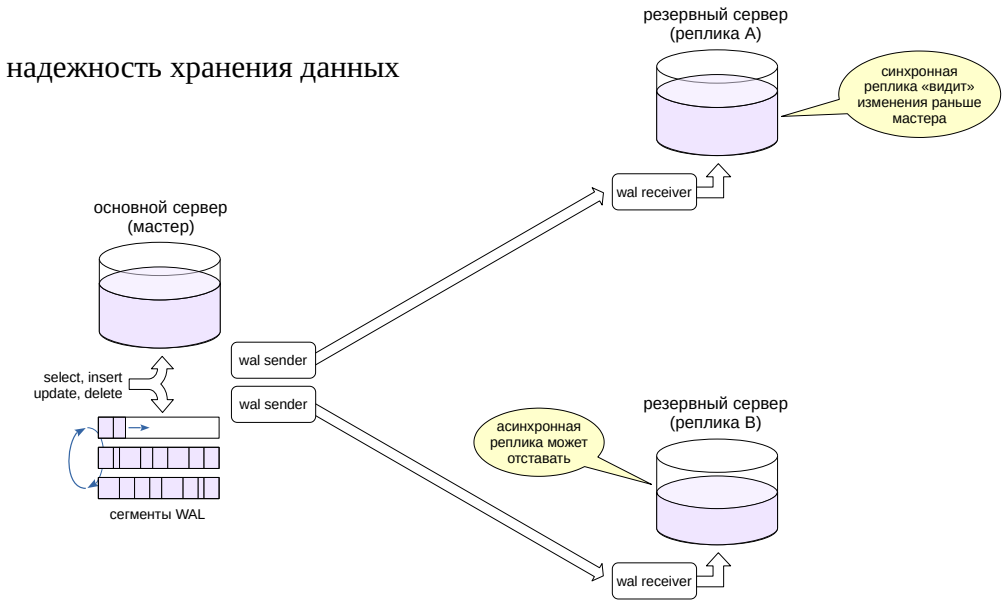
Задача: обеспечить высокую доступность и распределение нагрузки по чтению.

Для решения необходимо иметь мастер-сервер и несколько реплик. Реплики можно использовать для выполнения запросов только на чтение; при сбое основного сервера можно перейти на реплику с минимальным временем простоя.

Надо учитывать, что каждой реплике будет соответствовать отдельный процесс wal sender на основном сервере и, при необходимости, отдельный слот репликации.

Распределение нагрузки по чтению между репликами должно решаться внешними средствами.

2. Синхронная репликация



Задача: в случае сбоя основного сервера, не потерять никакие данные при переходе на реплику.

Решение состоит в использовании синхронной репликации. В случае одного сервера синхронная запись WAL гарантирует, что зафиксированные данные не будут потеряны при сбое. Аналогичный механизм работает и для репликации: фиксация изменений на мастере не завершается до тех пор, пока не получает подтверждение от реплики. При необходимости синхронностью можно управлять на уровне транзакций.

Синхронная репликация не обеспечивает идеальной согласованности данных между серверами: изменения могут становиться видимыми на мастере и на реплике в разные моменты времени.

Можно проводить синхронизацию с несколькими репликами, а также учитывать кворум.

На иллюстрации реплика B асинхронная и может отставать; реплика A синхронная. При фиксации изменений мастер выполняет следующее:

- делает запись в WAL (чтобы изменение не потерялось при сбое);

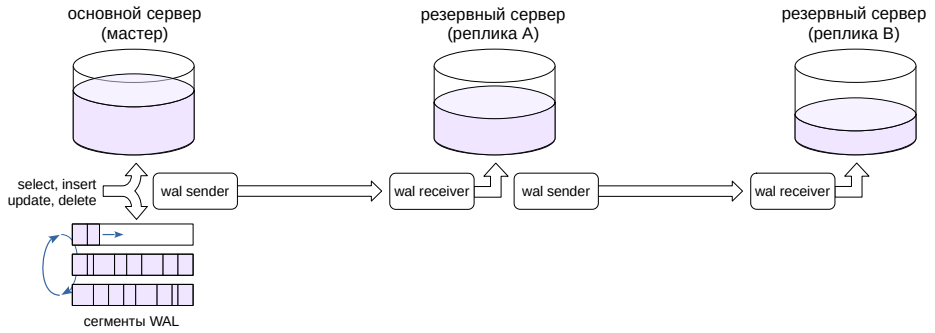
- дожидается подтверждения от реплики о получении записи WAL;

- изменяет состояние транзакции в буфере clog.

Таким образом, запрос к синхронной реплике может увидеть изменения даже раньше, чем запрос к мастеру.

3. Каскадная репликация

несколько реплик
без дополнительной нагрузки на мастер



Задача: иметь несколько реплик, не создавая дополнительной нагрузки на основной сервер.

Задача решается с помощью каскадной репликации, при которой одна реплика передает записи WAL другой реплике и так далее.

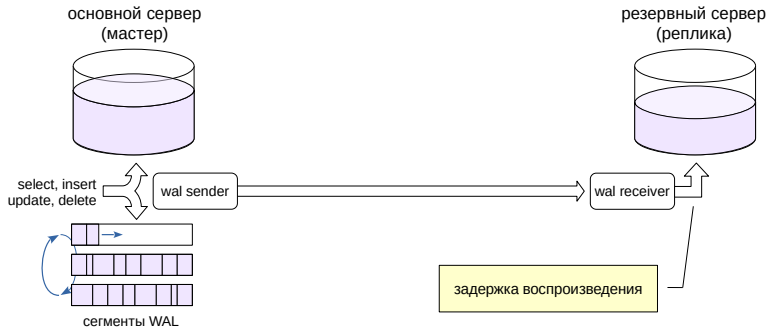
При каскадной репликации не поддерживается синхронизация. Однако обратная связь поступает основному серверу от всех реплик, так что этот функционал работает в полном объеме.

При необходимости переключения следует выбирать ближайшую к мастеру реплику как наименее отстающую.

На иллюстрации: на основном сервере только один процесс wal sender; реплики передают записи WAL друг другу по цепочке. Чем дальше от мастера, тем большее может накопиться запаздывание. Схема мониторинга усложняется: процесс надо контролировать на нескольких серверах.

4. Отложенная репликация

«машина времени»
и возможность восстановления на определенный момент без архива



Задача: иметь возможность просмотреть данные на некоторый момент в прошлом и, при необходимости, восстановить сервер на этот момент.

Проблема в том, что обычный механизм восстановления из архива на момент времени (point-in-time recovery) в принципе позволяет решить задачу, но требует большой подготовительной работы и занимает много времени. А способа построить снимок данных по состоянию на произвольный момент в прошлом в PostgreSQL нет.

Задача решается созданием реплики, которая применяет записи WAL не сразу, а через установленный интервал времени.

Чтобы задержка работала правильно, необходима синхронизация часов между серверами.

Если вывести реплику из режима непрерывного восстановления, остаток записей будет применен без каких-либо задержек.

При использовании обратной связи надо проявлять осторожность, поскольку большая задержка вызовет разрастание таблиц на мастере из-за того, что очистка не будет удалять старые версии строк, которые могут быть нужны реплике.

Логическая репликация

Публикации и подписчики

Обнаружение и разрешение конфликтов

Варианты настройки и использования репликации

Встроенная логическая репликация доступна в версиях PostgreSQL, начиная с 10.

Для передачи логических изменений (на уровне строк) используется протокол репликации. Для работы такой репликации требуется установка уровня журнала logical.

При логической репликации у сервера нет выделенной роли мастера или реплики, что позволяет организовать в том числе и двунаправленную репликацию.

Логическая репликация

Публикующий сервер

- выдает изменения данных построчно в порядке их фиксации (реплицируются команды INSERT, UPDATE, DELETE, TRUNCATE)
- возможна начальная синхронизация
- всегда используется слот логической репликации
- параметр *wal_level* = logical

Сервер подписки

- получает и применяет изменения
- без разбора, переписывания и планирования — сразу выполнение
- возможны конфликты с локальными данными

Логическая репликация использует модель «публикация-подписка». На одном сервере создается *публикация,* которая может включать ряд таблиц одной базы данных. Другие серверы могут *подписываться* на эту публикацию: получать и применять изменения.

Реплицируются только измененные строки таблиц (а не команды SQL). Команды DDL не передаются; таблицы-приемники на подписчике надо создавать вручную. Но есть возможность автоматической начальной синхронизации содержимого таблиц при создании подписки.

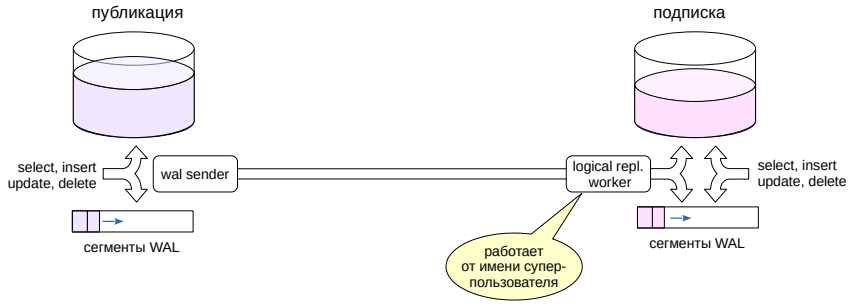
Технически информация об измененных строках извлекается и декодируется из имеющегося журнала WAL на публикующем сервере, а затем пересылается процессом wal sender подписчику по протоколу репликации в формате, независимом от платформы и версии сервера. Чтобы это было возможно, уровень журнала на публикующем сервере (параметр *wal_level*) должен быть установлен в значение logical.

Процесс logical replication worker на стороне подписки принимает и применяет изменения. Чтобы гарантировать надежность передачи (отсутствие потерь и повторов), в обязательном порядке используется *слот логической репликации* (похожий на слот физической репликации).

Применение изменений происходит без выполнения команд SQL и связанных с этим накладных расходов на разбор и планирование. С другой стороны, результат выполнения одной команды SQL может превратиться в множество однострочных изменений.

https://postgrespro.ru/docs/postgresql/13/logical-replication

Логическая репликация



На рисунке: фоновый процесс logical replication worker на сервере-подписчике получает информацию от публикующего сервера и применяет ее. В это же время сервер работает обычным образом и принимает запросы и на чтение, и на запись.

Логическая репликация

Теперь мы хотим настроить между серверами логическую репликацию. Для этого нам понадобится дополнительная информация в журнале.

```
=> ALTER SYSTEM SET wal_level = logical;

ALTER SYSTEM

student$ sudo pg_ctlcluster 13 main restart
```

На первом сервере создаем публикацию:

```
student$ /usr/lib/postgresql/13/bin/psql -d replica_overview

=> CREATE PUBLICATION test_pub FOR TABLE test;

CREATE PUBLICATION

=> \dRp+

                          Publication test_pub
  Owner  | All tables | Inserts | Updates | Deletes | Truncates | Via root
---------+------------+---------+---------+---------+-----------+----------
 student | f          | t       | t       | t       | t         | f
Tables:
    "public.test"
```

На втором сервере подписываемся на эту публикацию (отключаем первоначальное копирование данных):

```
=> CREATE SUBSCRIPTION test_sub
CONNECTION 'port=5432 user=student dbname=replica_overview'
PUBLICATION test_pub WITH (copy_data = false);

NOTICE:  created replication slot "test_sub" on publisher
CREATE SUBSCRIPTION

=> \dRs

            List of subscriptions
   Name   |  Owner  | Enabled | Publication
----------+---------+---------+-------------
 test_sub | student | t       | {test_pub}
(1 row)
```

```
=> INSERT INTO test VALUES (3, 'Три');

INSERT 0 1
```

```
=> SELECT * FROM test;

 id | descr
----+-------
  1 | Раз
  2 | Два
  3 | Три
(3 rows)
```

Состояние подписки можно посмотреть в представлении:

```
=> SELECT * FROM pg_stat_subscription \gx

-[ RECORD 1 ]---------+------------------------------
subid                 | 24618
subname               | test_sub
pid                   | 41681
relid                 |
received_lsn          | 0/8029E78
last_msg_send_time    | 2022-05-11 13:18:53.300097+03
last_msg_receipt_time | 2022-05-11 13:18:53.301537+03
latest_end_lsn        | 0/8029E78
latest_end_time       | 2022-05-11 13:18:53.300097+03
```

К процессам сервера добавился logical replication worker (его номер указан в pg_stat_subscription.pid):

```
student$ sudo ps -o pid,command --ppid 41007
```

```
  PID COMMAND
41010 postgres: 13/replica: checkpointer
41011 postgres: 13/replica: background writer
41012 postgres: 13/replica: stats collector
41362 postgres: 13/replica: student replica_overview [local] idle
41483 postgres: 13/replica: walwriter
41484 postgres: 13/replica: autovacuum launcher
41487 postgres: 13/replica: logical replication launcher
41681 postgres: 13/replica: logical replication worker for subscription 24618
```

Конфликты

Режимы идентификации для изменения и удаления

- столбцы первичного ключа (по умолчанию)
- столбцы указанного уникального индекса с ограничением NOT NULL
- все столбцы
- без идентификации (по умолчанию для системного каталога)

Конфликты — нарушение ограничений целостности

- репликация приостанавливается до устранения конфликта вручную
- либо исправление данных,
- либо пропуск конфликтующей транзакции

Вставка новых строк происходит достаточно просто. Интереснее обстоит дело при изменениях и удалениях — в этом случае надо как-то идентифицировать старую версию строки. По умолчанию для этого используются столбцы первичного ключа, но при определении таблицы можно указать и другие способы (replica identity): использовать уникальный индекс или использовать все столбцы. Можно вообще отказаться от поддержки репликации для некоторых таблиц (по умолчанию — таблицы системного каталога).

Поскольку таблицы на сервере публикации и сервере подписки могут изменяться независимо друг от друга, при вставке новых версий строк возможны конфликты — нарушения ограничений целостности. В этом случае процесс применения записей приостанавливается до тех пор, пока конфликт не будет разрешен вручную. Можно либо исправить данные на подписчике так, чтобы конфликт не происходил, либо отменить применение части записей.

Конфликты

Локальные изменения на стороне подписки не запрещаются, вставим строку в таблицу на втором сервере.

```
=> INSERT INTO test VALUES (4, 'Четыре (локально)');
```

```
INSERT 0 1
```

Если теперь строку с таким же значением первичного ключа вставить на первом сервере, при ее применении на стороне подписки произойдет конфликт.

```
=> INSERT INTO test VALUES (4, 'Четыре');
```

```
INSERT 0 1
```

```
=> INSERT INTO test VALUES (5, 'Пять');
```

```
INSERT 0 1
```

Подписка не может применить изменение, репликация остановилась.

```
=> SELECT * FROM pg_stat_subscription \gx
```

```
-[ RECORD 1 ]---------+---------
subid                 | 24618
subname               | test_sub
pid                   |
relid                 |
received_lsn          |
last_msg_send_time    |
last_msg_receipt_time |
latest_end_lsn        |
latest_end_time       |
```

```
=> SELECT * FROM test;
```

```
 id |       descr
----+-------------------
  1 | Раз
  2 | Два
  3 | Три
  4 | Четыре (локально)
(4 rows)
```

Чтобы разрешить конфликт, удалим строку на втором сервере и немного подождем...

```
=> DELETE FROM test WHERE id=4;
```

```
DELETE 1
```

```
=> SELECT * FROM test;
```

```
 id | descr
----+--------
  1 | Раз
  2 | Два
  3 | Три
  4 | Четыре
  5 | Пять
(5 rows)
```

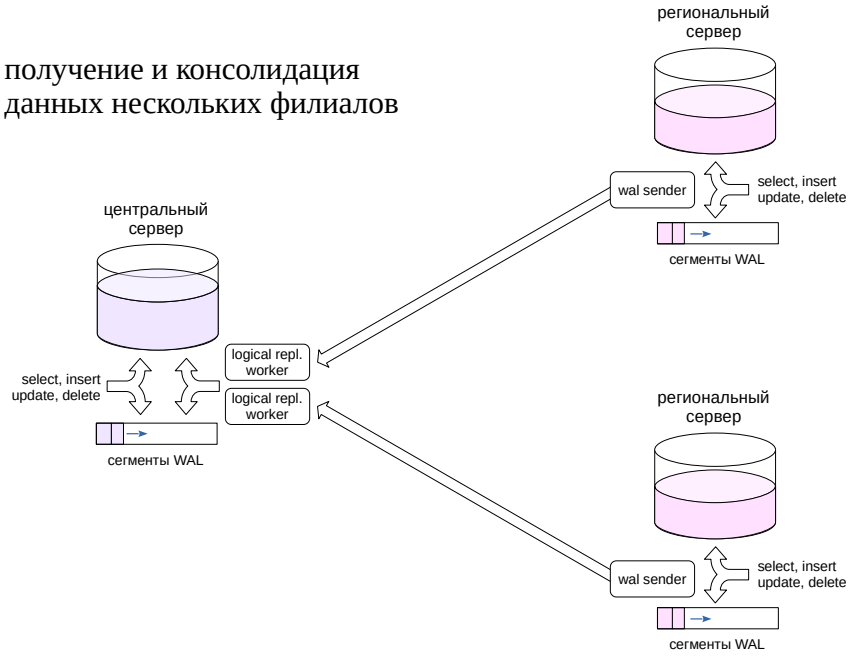
Репликация возобновилась.

Если репликация больше не нужна, надо аккуратно удалить подписку — иначе на публикующем сервере останется открытым репликационный слот.

```
=> DROP SUBSCRIPTION test_sub;
```

```
NOTICE:  dropped replication slot "test_sub" on publisher
DROP SUBSCRIPTION
```

1. Консолидация



Рассмотрим несколько задач, которые можно решить с помощью логической репликации.

Пусть имеются несколько региональных филиалов, каждый из которых работает на собственном сервере PostgreSQL. Задача состоит в консолидации части данных на центральном сервере.

Для решения задачи на региональных серверах создаются публикации необходимых данных. Центральный сервер подписывается на эти публикации. Полученные данные можно обрабатывать с помощью триггеров на стороне центрального сервера (например, приводя данные к единому виду).

Такая же схема, развернутая наоборот, позволяет, например, передавать справочную информацию от центрального сервера региональным.

Технический аспект: поскольку репликация основана на передаче журнала через слот, между серверами необходимо постоянное соединение, так как во время разрыва соединения центральный сервер вынужден сохранять файлы WAL.

С точки зрения бизнес-логики также есть множество особенностей, требующих всестороннего изучения. В каких-то случаях может оказаться проще передавать данные пакетно раз в определенный интервал времени.

На иллюстрации: на центральном сервере работают два процесса приема журналов, по одному на каждую подписку.

2. Обновление серверов

обновление основной версии
без прерывания обслуживания

28

Задача: обеспечить обновление основной версии сервера без прерывания обслуживания клиентов.

Поскольку между разными основными версиями нет двоичной совместимости, физическая репликация не помогает. Однако логическая репликация дает возможности для решения этой задачи.

Как обычно, требуются внешние средства для переключения пользователей между серверами.

Вначале создается новый сервер с желаемой версией PostgreSQL.

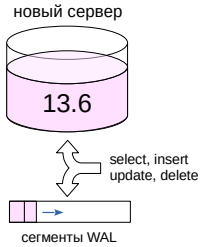
2. Обновление серверов

обновление основной версии
без прерывания обслуживания

Затем между серверами настраивается логическая репликация всех необходимых баз и данные синхронизируются. Это возможно благодаря тому, что для логической репликации не требуется двоичной совместимости между серверами.

2. Обновление серверов

обновление основной версии
без прерывания обслуживания

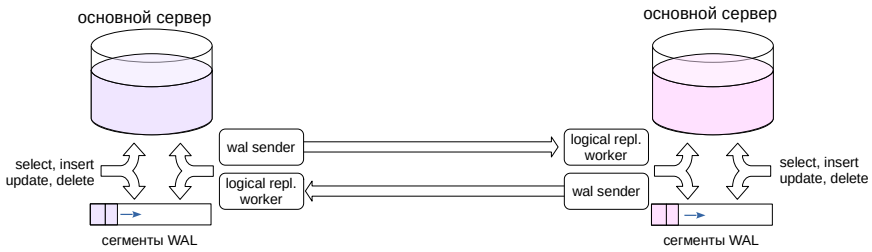


После этого клиенты переключаются на новый сервер, а старый выключается.

На самом деле процесс обновления с помощью логической репликации гораздо более сложен и сопряжен со значительными трудностями. Несколько подробнее он рассматривается в курсе DBA2, тема «Обновление сервера».

3. Мастер-мастер

кластер, в котором
данные могут изменять несколько серверов



Задача: обеспечить надежное хранение данных на нескольких серверах с возможностью изменения данных на любом узле.

Обычная физическая репликация позволяет изменять данные только на основном сервере. Логическая репликация дает возможность изменять данные одновременно на нескольких серверах. При этом прикладная система должна быть построена таким образом, чтобы избегать конфликтов при изменении данных в одних и тех же таблицах. Например, гарантировать, что разные серверы работают с разными диапазонами ключей.

Надо учитывать, что система мастер-мастер, построенная на логической репликации, не обеспечивает выполнение глобальных распределенных транзакций. При использовании синхронной репликации можно гарантировать надежность, но не согласованность данных между серверами. Кроме того, никаких средств для автоматизации обработки сбоев, подключения или удаления узлов из кластера и т. п. в PostgreSQL не предусмотрено — эти задачи должны решаться внешним средствами.

На иллюстрации: в конфигурации мастер-мастер каждый из серверов создает публикацию и подписку. Между ними происходит двусторонний обмен журнальными записями. PostgreSQL 13 не позволяет реализовать такую репликацию, но рано или поздно эта возможность должна будет появиться в ядре (см. расширения pg_logical https://www.2ndquadrant.com/en/resources-old/pglogical/ и BDR https://www.2ndquadrant.com/en/resources-old/postgres-bdr-2ndquadrant/).

Итоги

Механизм репликации основан на передаче журнальных записей на реплику и их применении

- трансляция потока записей или файлов WAL

Физическая репликация создает точную копию всего кластера

- однонаправленная
- требует двоичной совместимости

Логическая репликация передает изменения строк отдельных таблиц

- разнонаправленная
- совместимость на уровне протокола

Практика

1. Настройте физическую потоковую репликацию между двумя серверами в синхронном режиме.
2. Проверьте работу репликации. Убедитесь, что при остановленной реплике фиксация не завершается.
3. Выведите реплику из режима восстановления.
4. Создайте две таблицы на обоих серверах.
5. Настройте логическую репликацию первой таблицы от одного сервера к другому, а второй — в обратную сторону.
6. Проверьте работу репликации.

1. Для этого на мастере установите параметры:

- *synchronous_commit* = on,

- *synchronous_standby_names* = 'replica',

а на реплике в файле postgresql.auto.conf в параметр *primary_conninfo* добавьте «application_name=replica».

1. Физическая потоковая репликация в синхронном режиме

Ключом -R не пользуемся, сформируем postgresql.auto.conf вручную, так как требуются нестандартные значения параметров.

```
student$ pg_basebackup --pgdata=/home/student/basebackup
student$ sudo rm -rf /var/lib/postgresql/13/replica
student$ sudo mv /home/student/basebackup/ /var/lib/postgresql/13/replica
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/replica
student$ echo "primary_conninfo = 'user=student port=5432 application_name=replica'" | sudo tee /var/lib/postgresql/13/replica/postgresql.auto.conf
primary_conninfo = 'user=student port=5432 application_name=replica'
student$ sudo touch /var/lib/postgresql/13/replica/standby.signal
student$ sudo pg_ctlcluster 13 replica start
```

Настройки основного сервера:

```
student$ /usr/lib/postgresql/13/bin/psql
=> ALTER SYSTEM SET synchronous_commit = on;
ALTER SYSTEM
=> ALTER SYSTEM SET synchronous_standby_names = 'replica';
ALTER SYSTEM
student$ sudo pg_ctlcluster 13 main reload
```

2. Проверка физической репликации

```
student$ /usr/lib/postgresql/13/bin/psql
=> CREATE DATABASE replica_overview;
CREATE DATABASE
=> \c replica_overview
You are now connected to database "replica_overview" as user "student".
=> CREATE TABLE t(n integer);
CREATE TABLE
=> INSERT INTO t VALUES (1);
INSERT 0 1
=> SELECT * FROM pg_stat_replication \gx
-[ RECORD 1 ]----+------------------------------
pid              | 57607
usesysid         | 16384
usename          | student
application_name | replica
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2022-05-11 13:23:54.820523+03
backend_xmin     |
state            | streaming
sent_lsn         | 0/1401A620
write_lsn        | 0/1401A620
flush_lsn        | 0/1401A620
replay_lsn       | 0/14002B48
write_lag        | 00:00:00.000219
flush_lag        | 00:00:00.016133
replay_lag       | 00:00:00.318503
sync_priority    | 1
sync_state       | sync
reply_time       | 2022-05-11 13:23:59.965901+03
```

sync_state: sync говорит о том, что репликация работает в синхронном режиме.

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_overview
  => SELECT * FROM t;
   n
  ---
   1
  (1 row)
```

```
student$ sudo pg_ctlcluster 13 replica stop
=> BEGIN;
BEGIN
=> INSERT INTO t VALUES (2);
INSERT 0 1
=> COMMIT;
```

Фиксация ждет появления синхронной реплики.

```
student$ sudo pg_ctlcluster 13 replica start
```

```
COMMIT
```

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_overview
```

```
=> SELECT * FROM t;
```

```
 n
---
 1
 2
(2 rows)
```

3. Окончание восстановления

```
student$ sudo pg_ctlcluster 13 replica promote
```

У первого сервера необходимо отменить синхронный режим:

```
=> ALTER SYSTEM RESET synchronous_standby_names;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 main reload
```

4. Таблицы для проверки логической репликации

```
=> CREATE TABLE a(id integer);
```

```
CREATE TABLE
```

```
=> CREATE TABLE b(s text);
```

```
CREATE TABLE
```

```
=> CREATE TABLE a(id integer);
```

```
CREATE TABLE
```

```
=> CREATE TABLE b(s text);
```

```
CREATE TABLE
```

5. Настройка логической репликации

```
=> ALTER SYSTEM SET wal_level = logical;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 main restart
```

```
=> ALTER SYSTEM SET wal_level = logical;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 replica restart
```

```
student$ /usr/lib/postgresql/13/bin/psql -d replica_overview
```

```
=> CREATE PUBLICATION a_pub FOR TABLE a;
```

```
CREATE PUBLICATION
```

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_overview
```

```
=> CREATE PUBLICATION b_pub FOR TABLE b;
```

```
CREATE PUBLICATION
```

```
=> CREATE SUBSCRIPTION b_sub
CONNECTION 'port=5433 user=student dbname=replica_overview'
PUBLICATION b_pub;
```

```
NOTICE:  created replication slot "b_sub" on publisher
CREATE SUBSCRIPTION
```

```
=> CREATE SUBSCRIPTION a_sub
CONNECTION 'port=5432 user=student dbname=replica_overview'
PUBLICATION a_pub;
```

```
NOTICE:  created replication slot "a_sub" on publisher
CREATE SUBSCRIPTION
```

6. Проверка логической репликации

```
=> INSERT INTO a VALUES (1);
```

```
INSERT 0 1
```

```
=> SELECT * FROM a;
```

```
 id
----
  1
(1 row)
```

```
=> INSERT INTO b VALUES ('Раз');
```

```
INSERT 0 1
```

```
=> SELECT * FROM b;
```

```
  s
-----
 Раз
(1 row)
```

Удалим ненужные более подписки:

```
=> DROP SUBSCRIPTION b_sub;
```

```
NOTICE:  dropped replication slot "b_sub" on publisher
DROP SUBSCRIPTION
```

```
=> DROP SUBSCRIPTION a_sub;
```

```
NOTICE:  dropped replication slot "a_sub" on publisher
DROP SUBSCRIPTION
```