Postgres PROFESSIONAL

# Потоковая репликация

# Темы

Потоковая репликация

Мониторинг процесса

Репликация без архива

Проблемы и решения: слоты репликации и обратная связь

# Потоковая репликация

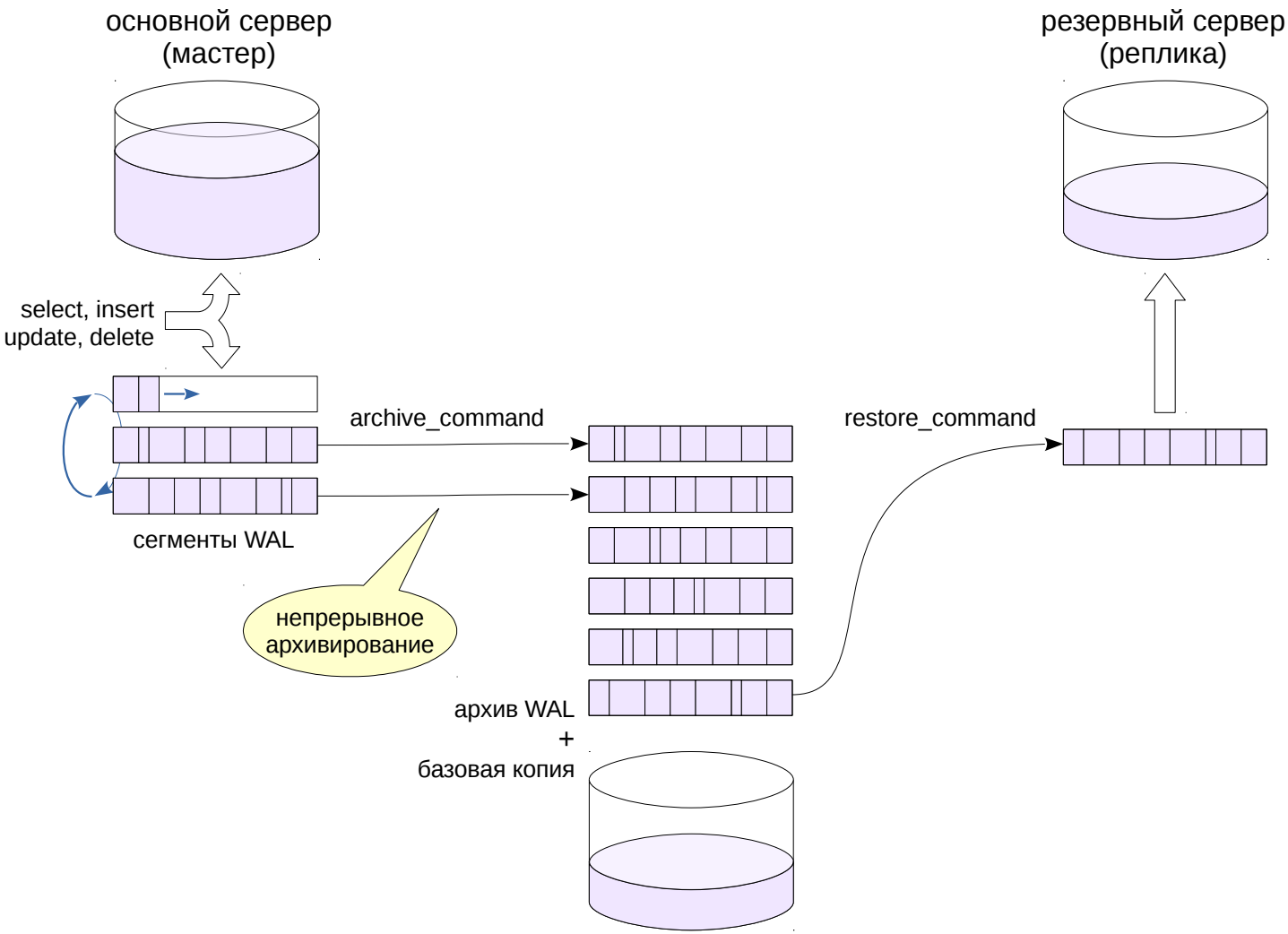
## Потоковая репликация

- реплика «догоняет» мастер, используя архив
- затем получает поток записей WAL, не дожидаясь заполнения сегмента

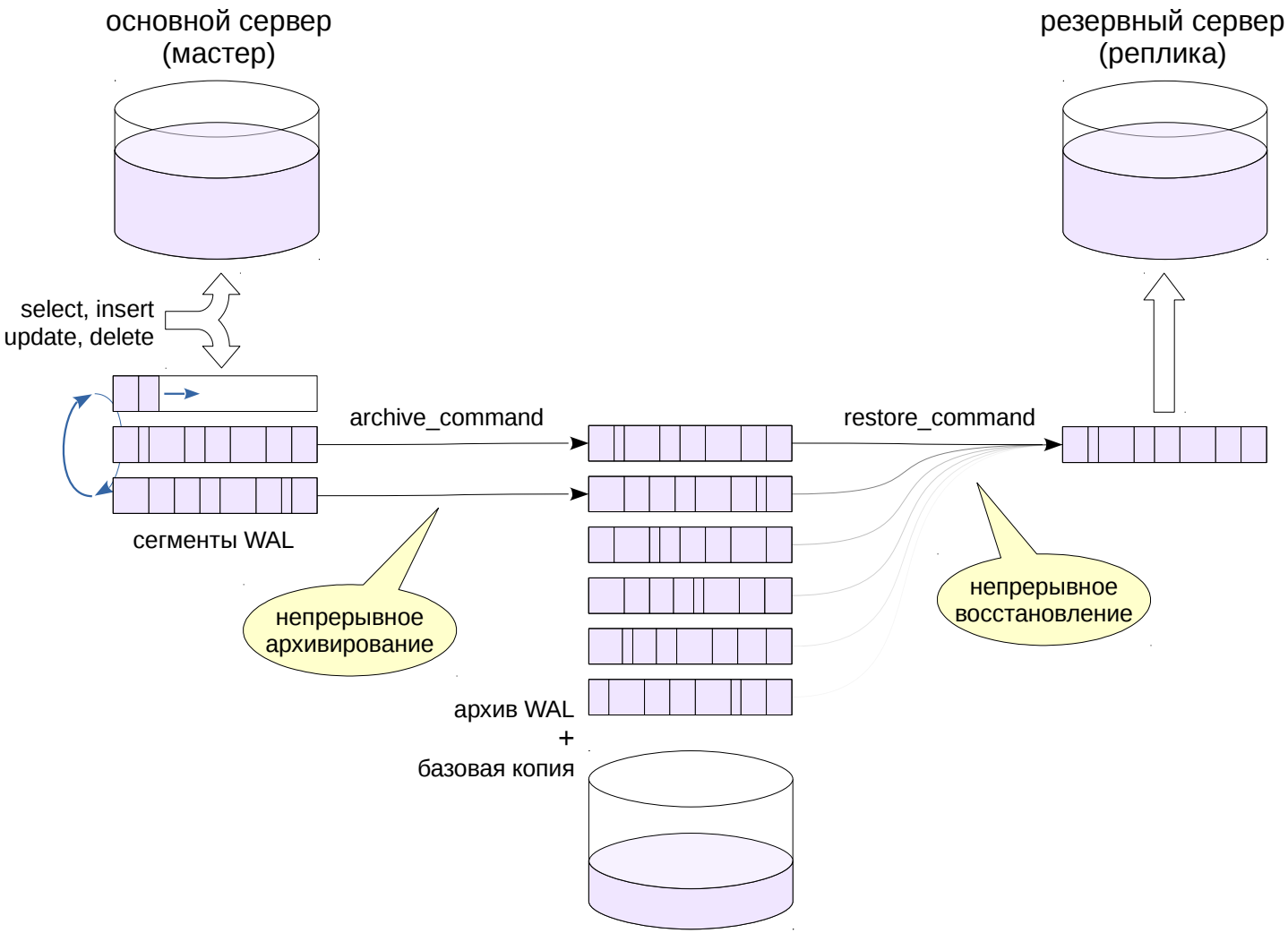
## Сравнение

| *трансляция файлов журнала* | *потоковая репликация* |
|---|---|
| вынужденное отставание реплики от мастера | отставание сведено к минимуму |
| мастер ничего не знает про реплику | есть возможность взаимодействия |

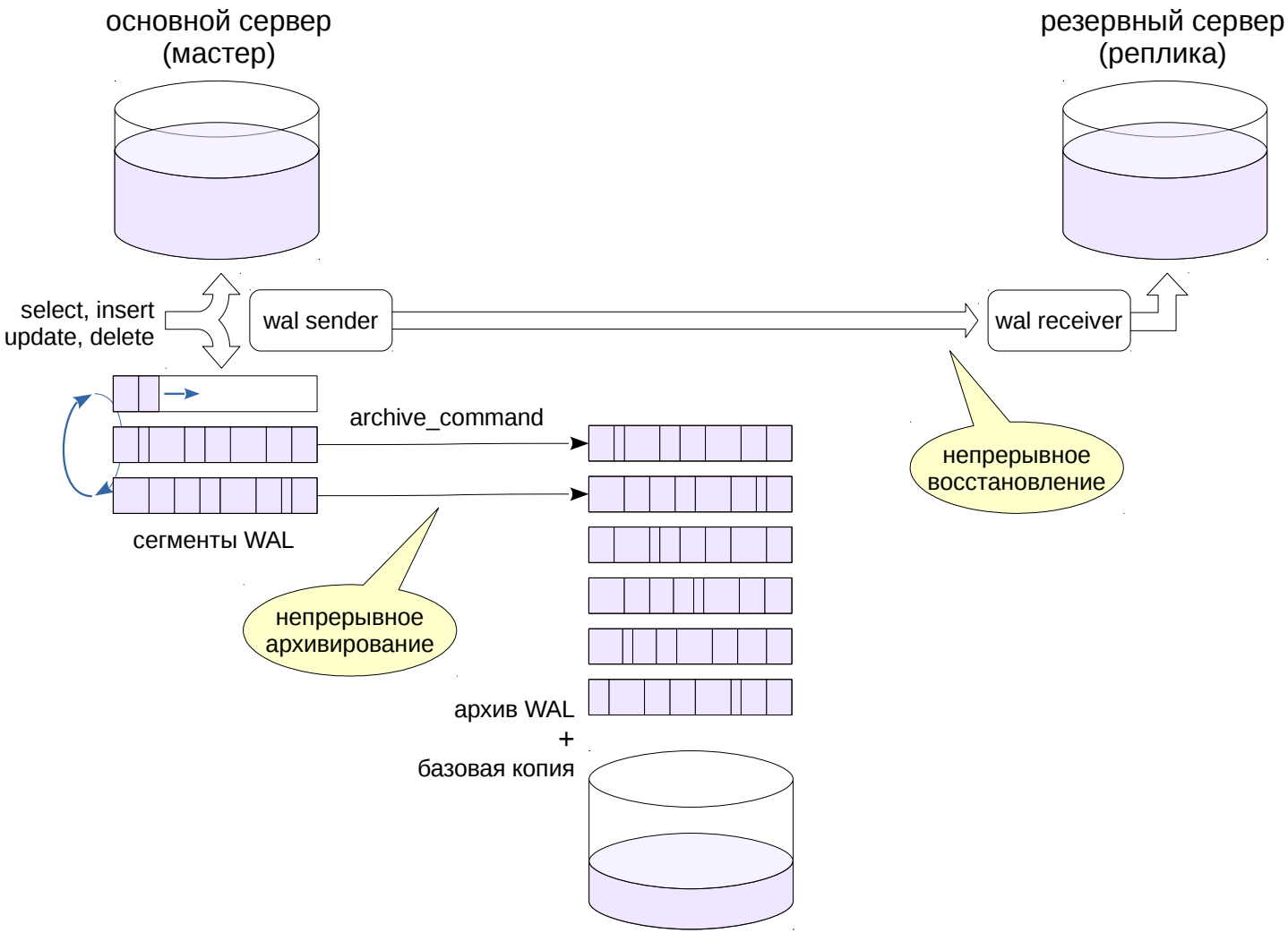
# Потоковая репликация



# Потоковая репликация



# Потоковая репликация



# Настройка

## Мастер

*postgresql.conf*

max_wal_senders — *по одному для каждой реплики + pg_basebackup*

*pg_hba.conf*

разрешение для базы replication

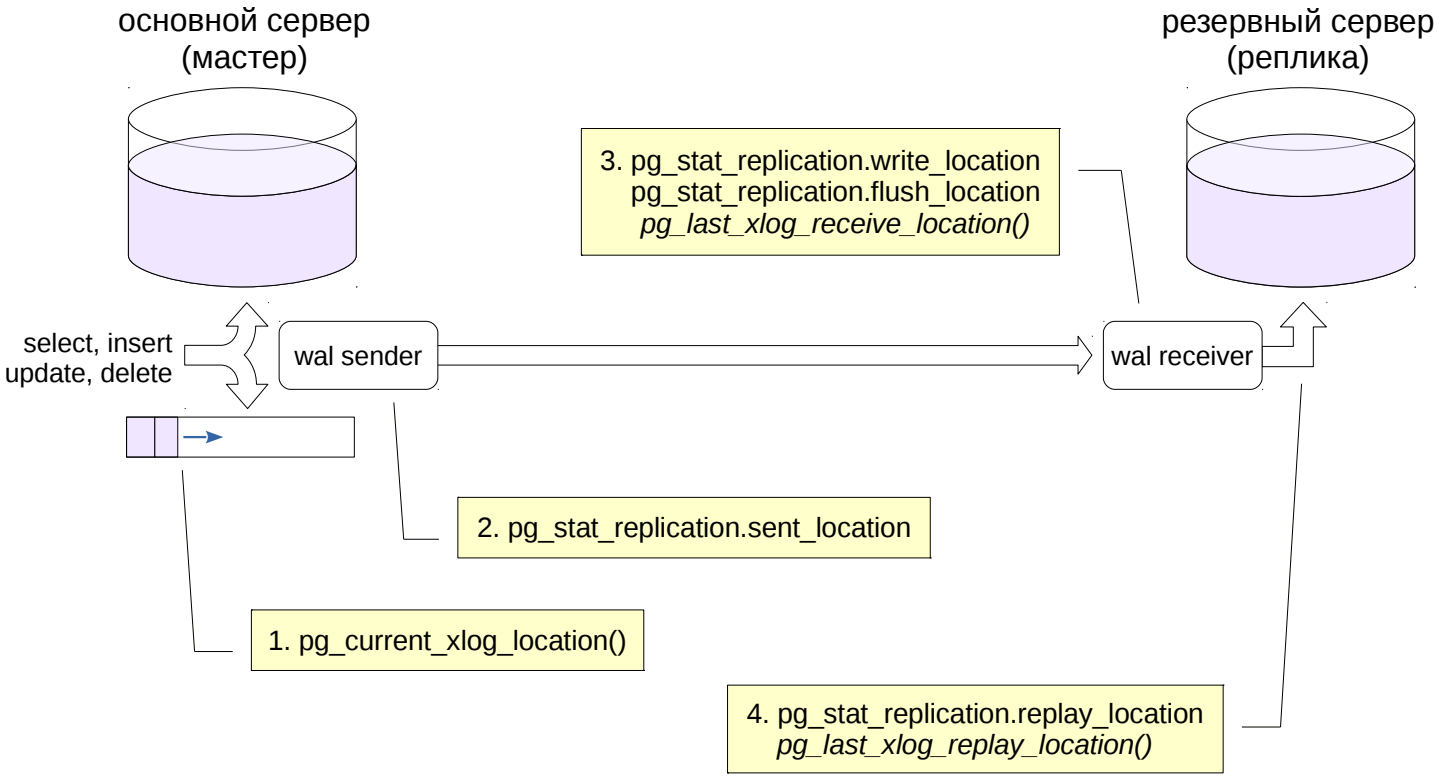
## Реплика

*recovery.conf*

primary_conninfo = 'host=*узел* port=*порт* user=*пользователь*'

- host — имя узла (или hostaddr — IP-адрес сервера)
- port — номер порта
- user — роль (с атрибутом replication или суперпользователь)
- password — если необходим (можно указать в ~/.pgpass)

# Мониторинг




1 – 2 = задержка записи в поток на мастере

2 – 3 = задержка получения данных репликой

3 – 4 = задержка применения данных на реплике

# Восстановление

## Общий алгоритм непрерывного восстановления

1. чтение сегментов WAL из архива с помощью restore_command, пока команда завершается успешно
2. чтение сегментов WAL из pg_xlog, пока находятся нужные файлы
3. чтение потока WAL (если настроена потоковая репликация), пока есть соединение
4. повторять с п. 1

# Можно и без архива

## Мастер

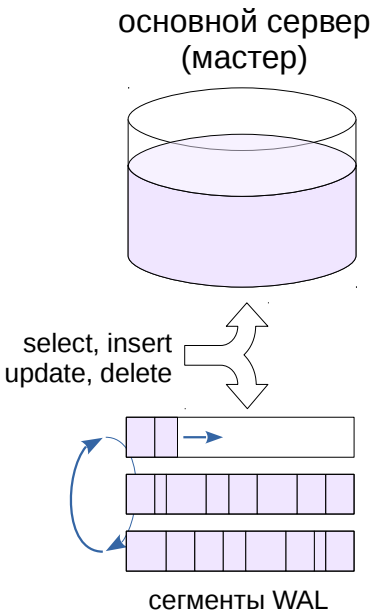
- не настраиваем непрерывное архивирование
- создаем базовую резервную копию
  с необходимым набором сегментов:

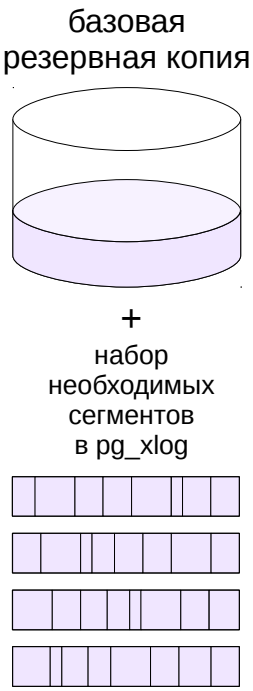
```
pg_basebackup --xlog-method fetch|stream
```

## Реплика

- восстанавливается из резервной копии
- в отсутствии архива немедленно
  переключается на потоковую репликацию

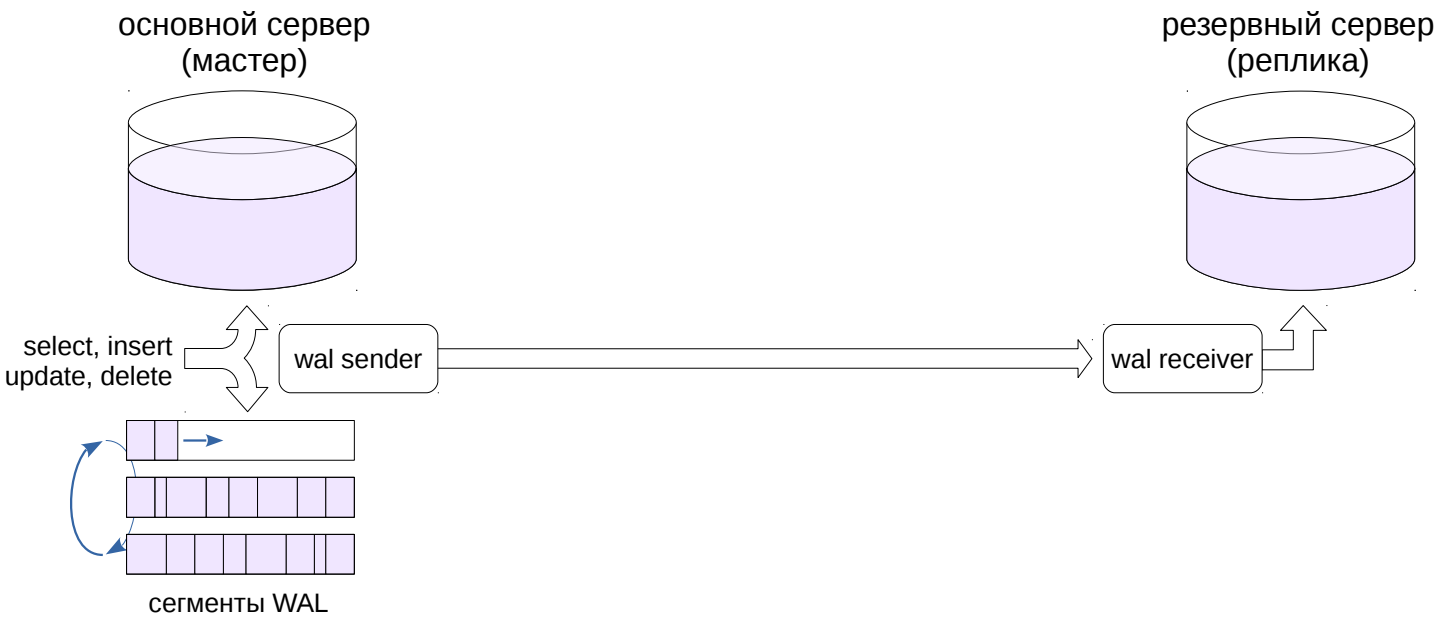
# Можно и без архива



# Можно и без архива

# Можно и без архива



# Мастер не знает о реплике

Мастер удалит сегмент, получила реплика данные или нет

- например, реплика долго восстанавливалась из резервной копии, или реплика была выключена некоторое время
- *не проблема, если есть архив,*
  *иначе надо увеличивать wal_keep_segments*

Конфликтующие с запросами записи WAL

- мастер очищает версии строк, необходимые для снимка на реплике
- *vacuum_defer_cleanup_age действует «вслепую»*
- *max_standby_streaming_delay откладывает применение WAL*
  *по аналогии с max_standby_archive_delay для архива*

# Мастер знает о реплике!

## Слот репликации

- серверный объект для получения записей WAL
- помнит, какая запись была считана последней
- сегмент WAL не удаляется, пока он не прочитан полностью

## Обратная связь

- процесс wal receiver шлет процессу wal sender информацию о наиболее старом номере транзакции, необходимом для снимков
- процесс wal sender выступает как серверный процесс, защищая нужные реплике версии строк от очистки

# Настройка слота

## Мастер

| | |
|---|---|
| создание | pg_create_physical_replication_slot('*имя*') |
| мониторинг | представление pg_replication_slots |
| удаление | pg_drop_replication_slot('*имя*') |

*postgresql.conf*

max_replication_slots ≥ 1

## Реплика

*recovery.conf*

primary_slot_name = '*имя*'

# Настройка обратной связи

Мастер

настройка не требуется

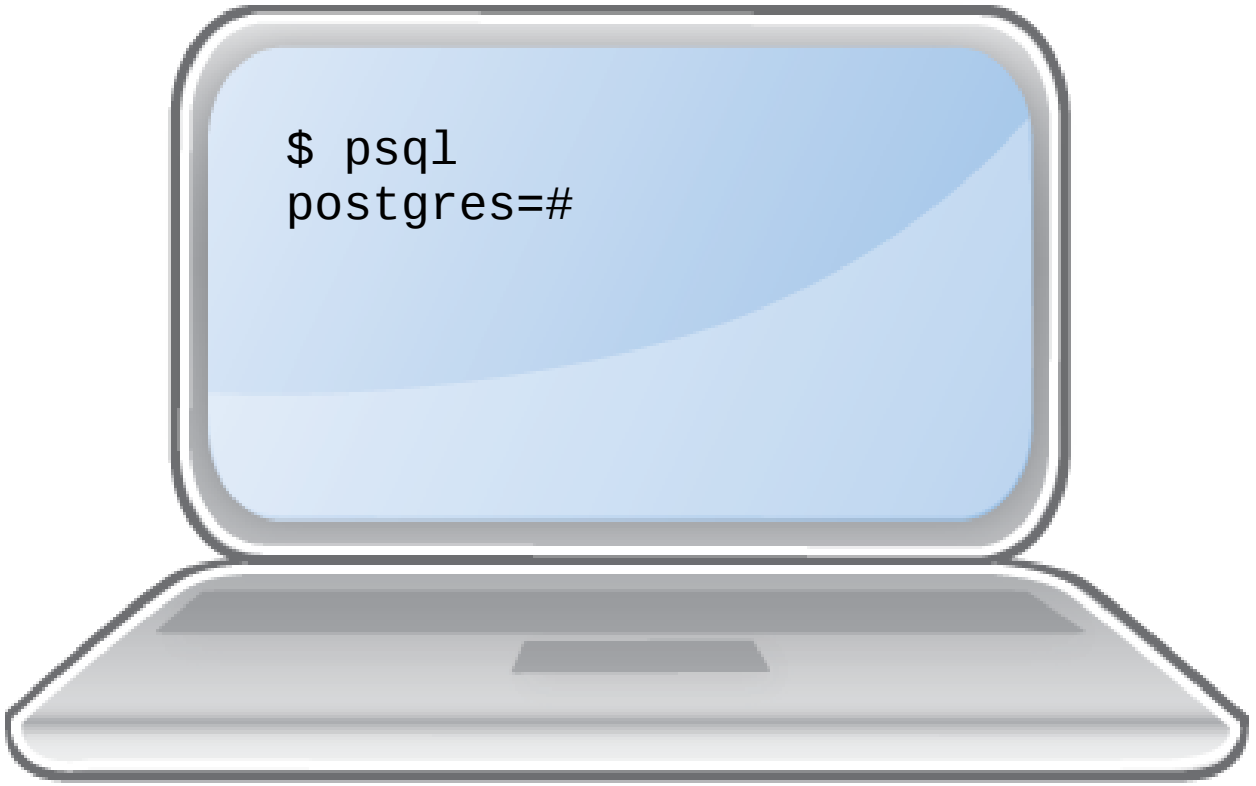
Реплика

*postgresql.conf*

hot_standby_feedback = on

wal_receiver_status_interval — *частота обратной связи (10 секунд)*

# Демонстрация



# Итоги

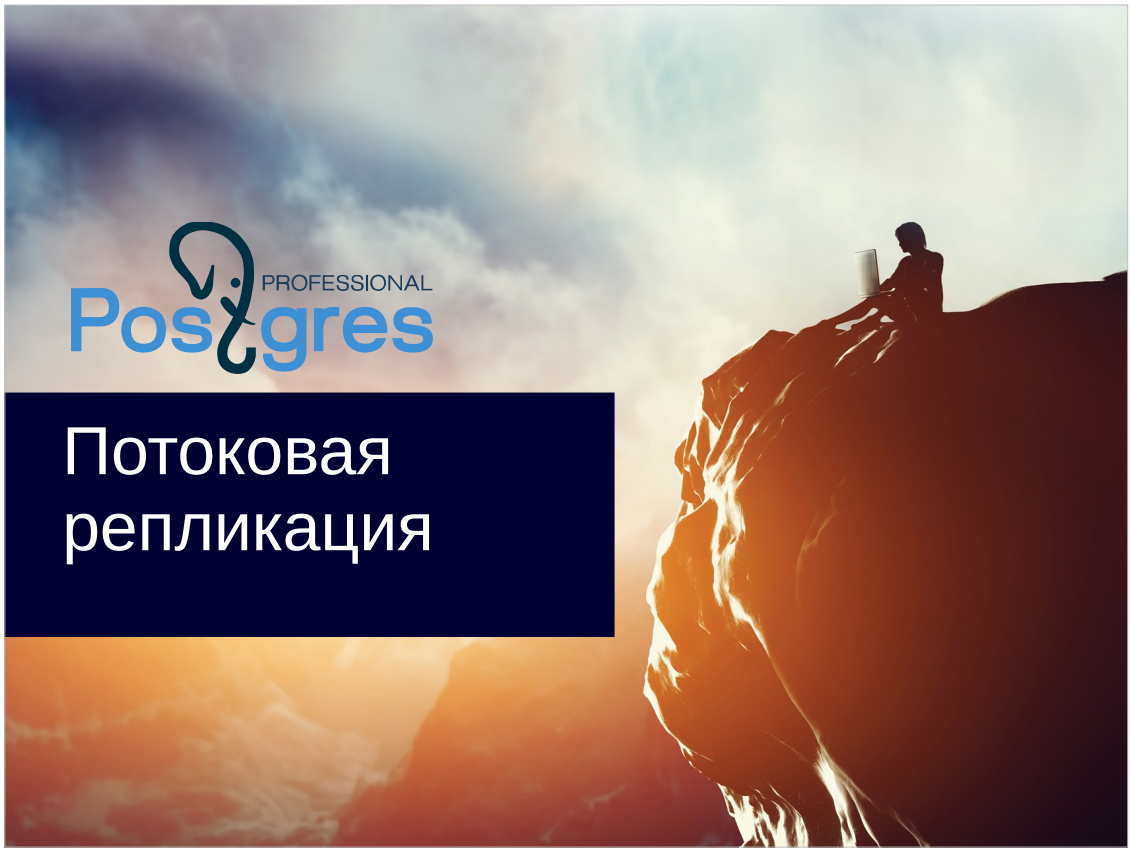
Потоковая репликация позволяет уменьшить отставание реплики до минимума

Мастер может учитывать интересы реплики: слоты репликации и обратная связь

Механизм слотов позволяет обходиться без архива, но требуется мониторинг

# Практика

1. Выполните необходимую настройку мастера для потоковой репликации с использованием слота, без непрерывного архивирования.
2. Создайте базовую резервную копию на месте каталога с данными реплики (сервер установлен под пользователем postgres2).
3. Выполните настройку реплики для восстановления в режиме горячего резерва с использованием слота и обратной связи.
4. Запустите реплику, создайте на мастере таблицу в базе данных DB11 и убедитесь, что она появилась на реплике.
5. Остановите реплику и вставьте много данных в таблицу. Измерьте размер сгенерированных при этом журнальных записей.
6. Снова запустите реплику и посмотрите, как изменяется задержка репликации.





Авторы: Егор Рогов, Павел Лузанов



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

# Темы

Потоковая репликация

Мониторинг процесса

Репликация без архива

Проблемы и решения: слоты репликации и обратная связь

# Потоковая репликация

## Потоковая репликация

реплика «догоняет» мастер, используя архив

затем получает поток записей WAL, не дожидаясь заполнения сегмента

## Сравнение

| *трансляция файлов журнала* | *потоковая репликация* |
|---|---|
| вынужденное отставание реплики от мастера | отставание сведено к минимуму |
| мастер ничего не знает про реплику | есть возможность взаимодействия |

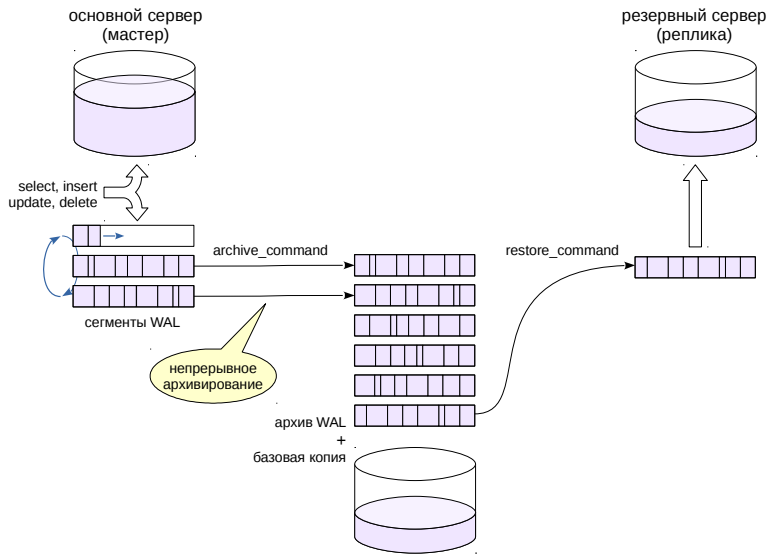


В прошлой теме была рассмотрена репликация на основе трансляции на реплику сегментов WAL из архива. Одна из основных проблем такого подхода состоит в том, что реплика вынужденно отстает от мастера на время заполнения сегмента. Максимум можно регулировать параметром archive_timeout, но при сильном уменьшении этого времени крайне неэффективно будет расходоваться место (только небольшая часть из 16 МБ будет заполнена полезными данными).

Решение этой проблемы состоит в использовании потоковой репликации. При этом реплика сначала «догоняет» мастера с помощью архива, а затем подключается к нему по специальному протоколу репликации и получает поток записей WAL, не дожидаясь заполнения сегмента. Мы уже встречались с протоколом репликации: утилита pg_basebackup использует его при потоковом способе получения журналов (--xlog-method=stream).

Потоковая репликация позволяет решить и вторую проблему файловой доставки: отсутствие у мастера информации о том, что происходит на реплике.

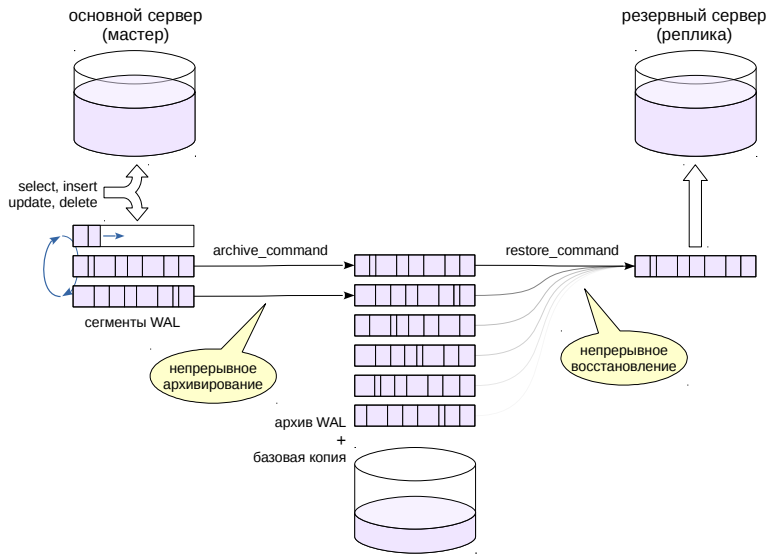
# Потоковая репликация






Сначала все происходит так же, как и при репликации с трансляцией файлов журнала. Реплика, развернутая из базовой резервной копии, приступает к восстановлению из архива.

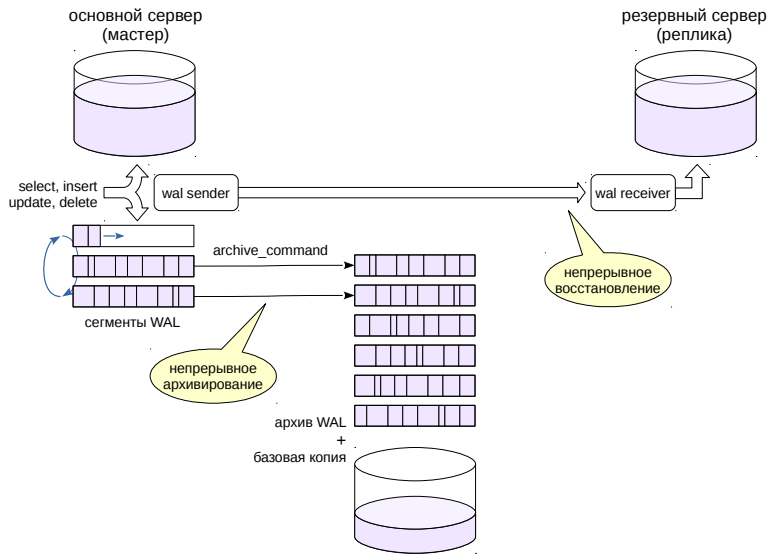
# Потоковая репликация






Реплика прочитала и применила все сегменты WAL, база данных пришла в (почти) актуальное состояние.

# Потоковая репликация






После того, как очередная команда restore_command не найдет в архиве сегмента (так как он еще не заполнен), реплика подключается к мастеру по протоколу репликации и дальше получает поток записей WAL. Отставание реплики от мастера сводится к минимуму.

В целом схема аналогична обычному подключению, при котором запросы клиента обрабатываются серверным процессом.

Подключение инициируется процессом wal receiver на реплике. Как обычно, подключение прослушивается процессом postmaster. Но в данном случае для обслуживания подключения порождается не серверный процесс, а специальный процесс wal sender, который понимает не запросы SQL, а протокол репликации (http://www.postgresql.org/docs/current/static/protocol-replication.html).

Следует заметить, что, несмотря на параметр archive_mode = on, реплика ничего не записывает в архив.

# Настройка

Мастер

*postgresql.conf*

max_wal_senders — *по одному для каждой реплики + pg_basebackup*

*pg_hba.conf*

разрешение для базы replication

Реплика

*recovery.conf*

primary_conninfo = 'host=*узел* port=*порт* user=*пользователь*'

- host — имя узла (или hostaddr — IP-адрес сервера)
- port — номер порта
- user — роль (с атрибутом replication или суперпользователь)
- password — если необходим (можно указать в ~/.pgpass)



Настройка потоковой репликации не сильно отличается от настройки репликации с доставкой сегментов WAL.

На мастере надо убедиться, что параметр max_wal_senders установлен достаточно большим (1 для каждой реплики, плюс 1 или 2 для pg_basebackup).

Также необходимо, чтобы реплика смогла подключиться к мастеру (в pg_hba.conf должно быть прописано разрешение для поключения к базе replication).

Единственная настройка, которую надо дополнительно сделать на реплке, это параметр primary_conninfo в recovery.conf, в котором указывается информация, необходимая для подключения к мастеру, в виде пар параметр-значение, разделенных пробелами.

Наиболее употребительные параметры:

- host (или hostaddr) указывает имя (или IP-адрес) мастера

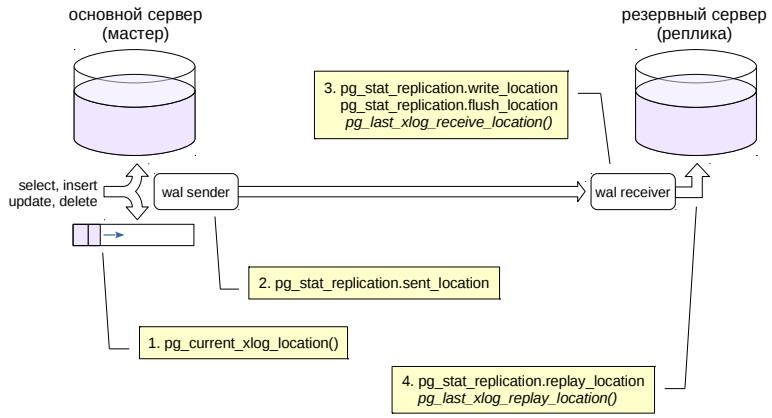
- port указывает порт

- user указывает пользователя, который должен быть создан с атрибутом replication, либо являться суперпользователем.

- password указывает пароль, если это необходимо для аутентификации (пароль можно указать и отдельно в файле ~/.pgpass)

http://www.postgresql.org/docs/current/static/standby-settings.html

# Мониторинг




1 – 2 = задержка записи в поток на мастере
2 – 3 = задержка получения данных репликой
3 – 4 = задержка применения данных на реплике



В ходе репликации возможны проблемы, о которых должен предупредить заранее настроенный мониторинг.

Мониторинг репликации построен на отслеживании позиции в журнале упреждающей записи. Можно выделить четыре контрольные точки:

1. запись в сегмент WAL на мастере

2. трансляция записи процессом wal sender

3. получение записи процессом wal reciver

4. применение полученной записи

Все точки отслеживаются на мастере, первая — функцией pg_current_xlog_position(), остальные — представлением pg_stat_replication. Реплика передает мастеру статус репликации при каждой записи на диск, но как минимум раз в wal_receiver_status_interval секунд, (по умолчанию — 10 секунд).

Для получения номера записи на реплике есть функции pg_last_xlog_receive_location() и pg_last_xlog_replay_location() (на рисунке выделены наклонным шрифтом).

Разница между второй и первой точками означает задержку записи в поток (мастер не справляется с нагрузкой?).

Разница между третьей и второй точками означает задержку получения данных репликой (реплика не справляется с нагрузкой?).

Разница между четвертой и третьей точкой может быть вызвана параметром max_standby_streaming_delay.

# Восстановление

Общий алгоритм непрерывного восстановления

1. чтение сегментов WAL из архива с помощью restore_command, пока команда завершается успешно
2. чтение сегментов WAL из pg_xlog, пока находятся нужные файлы
3. чтение потока WAL (если настроена потоковая репликация), пока есть соединение
4. повторять с п. 1



В режиме непрерывного восстановления PostgreSQL (процесс startup process) последовательно читает записи WAL из трех источников:

- сначала из архива, до тех пор, пока restore_command завершается со статусом 0

- затем из локального каталога pg_xlog (на тот случай, если происходит восстановление из резервной копии и надо «подхватить» сегмент WAL, не попавший в архив)

- затем, если настроена потоковая репликация, происходит подключение к мастеру по протоколу репликации

Если потоковая репликация завершается, процесс повторяется сначала.

Например, потоковая репликация может завершится с ошибкой из-за того, что реплика долго не получала записи и за это время мастер успел перезаписать нужные сегменты WAL (вероятный сценарий, если реплика останавливается на некоторое время). В этом случае восстановление автоматически продолжится из архива, а затем снова перейдет на потоковую репликацию.

# Можно и без архива

## Мастер

- не настраиваем непрерывное архивирование
- создаем базовую резервную копию
  с необходимым набором сегментов:

```
pg_basebackup --xlog-method fetch|stream
```

## Реплика

- восстанавливается из резервной копии
- в отсутствии архива немедленно
  переключается на потоковую репликацию



В прошлой теме мы рассматривали возможность удаления из архива сегментов, которые больше не нужны для работы реплики (если архив используется только для поддержки репликации).

Можно ли настроить потоковую репликацию без архива? Можно. Для этого надо создать базовую резервную копию, включив в нее необходимые сегменты WAL (ключ --xlog-method утилиты pg_basebackup).

Реплика, восстановившись из этой резервной копии, переключится на потоковую репликацию.

# Можно и без архива

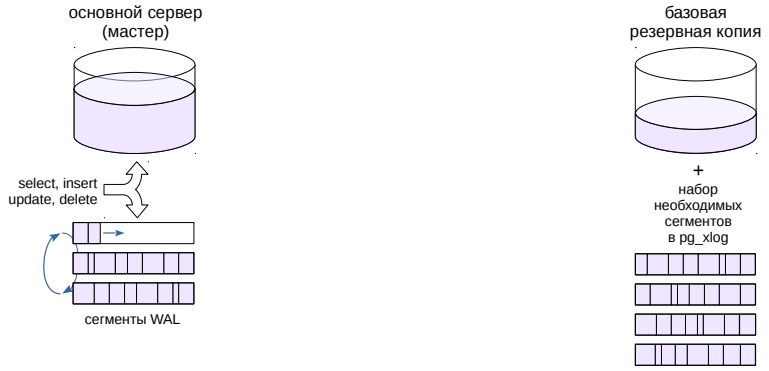


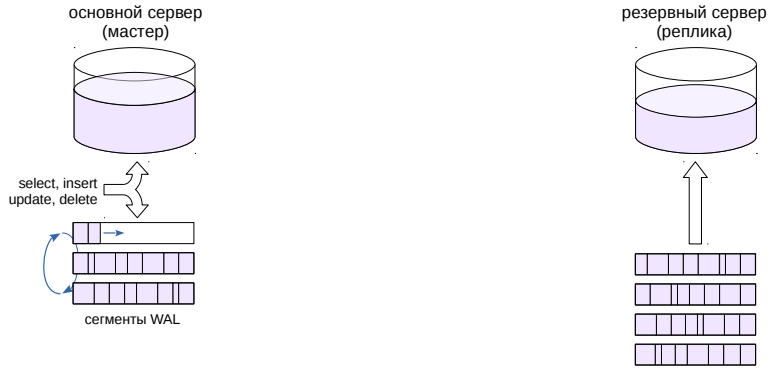



Иллюстрация.

Мастер не использует непрерывной архивирование.

Базовая резервная копия с необходимым набором WAL-файлов создается на месте будущей реплики.

# Можно и без архива

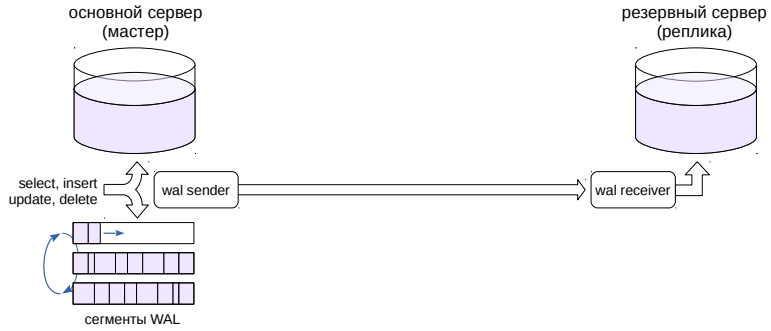





Прочитав и применив все сегменты WAL из резервной копии, реплика восстановится до состояния, соответствующего времени окончания создания резервной копии.

Понятно, что за время восстановления мастер успеет уйти вперед.

# Можно и без архива






Далее реплика подключается к мастеру по протоколу потоковой репликации и «догоняет» мастер с помощью полученных записей WAL.

Главное, чтобы мастер не успел удалить «ненужные» с его точки зрения сегменты WAL.

# Мастер не знает о реплике

Мастер удалит сегмент, получила реплика данные или нет

- например, реплика долго восстанавливалась из резервной копии, или реплика была выключена некоторое время
- *не проблема, если есть архив, иначе надо увеличивать wal_keep_segments*

Конфликтующие с запросами записи WAL

- мастер очищает версии строк, необходимые для снимка на реплике
- *vacuum_defer_cleanup_age действует «вслепую»*
- *max_standby_streaming_delay откладывает применение WAL по аналогии с max_standby_archive_delay для архива*



В начале разговора мы определили две главные проблемы репликации: отставание реплики и отсутствие у мастера информации о состоянии дел реплике. Первая проблема решается потоковой репликацией. Однако вторая проблема постоянно дает о себе знать.

Во-первых, если реплика не успевает потреблять данные из потока, сегмент WAL может быть перезаписан. Без архива это превращается в серьезную проблему. Ее можно решать, увеличивая параметр wal_keep_segments, но это угадывание подходящего значения с риском потерять реплику.

Во-вторых, проблема с записями WAL, конфликтующими с запросами на реплике, остается в том же виде, в котором мы сталкивались с ней в прошлой теме. Для защиты от очистки нужных реплике версий строк можно воспользоваться параметром vacuum_defer_cleanup_age (который откладывает очистку на указанное количество транзакций), но это снова угадывание правильного значения. Другой параметр — max_standby_streaming_delay, определяющий максимальное время, на которое может быть отложено применение записей WAL (по аналогии с max_standby_archive_delay).

В целом хочется иметь более надежное решение.

## Мастер знает о реплике!

Слот репликации

- серверный объект для получения записей WAL
- помнит, какая запись была считана последней
- сегмент WAL не удаляется, пока он не прочитан полностью

Обратная связь

- процесс wal receiver шлет процессу wal sender информацию о наиболее старом номере транзакции, необходимом для снимков
- процесс wal sender выступает как серверный процесс, защищая нужные реплике версии строк от очистки



И такое решение есть: мастер должен знать о том, что происходит на реплике. Для этого есть два механизма.

Во-первых, слоты репликации. Слот — серверный объект, создаваемый на мастере, который хранит информацию о том, какая запись была прочитана последней (независимо от того, включена или остановлена реплика) и не дает удалять сегменты WAL, которые будут нужны, чтобы продолжить передачу.

Во-вторых, обратная связь. При включенной обратной связи процесс wal receiver шлет процессу wal sender не только статус репликации, но и наиболее старый номер транзакции, необходимый для поддержания снимков на реплике. Процесс wal sender удерживает нужные версии строк от очистки. (Если реплика отключается, то соединение разрывается, wal sender завершается и перестает удерживать версии строк.)

Эти два механизма можно использовать как по отдельности, так и вместе.

Заметим, что для таких конфликтов, как эксклюзивные блокировки (например, при удалении таблиц), нет другого решения, кроме как max_standby_streaming_delay.

# Настройка слота

Мастер

| | |
|---|---|
| создание | pg_create_physical_replication_slot('*имя*') |
| мониторинг | представление pg_replication_slots |
| удаление | pg_drop_replication_slot('*имя*') |

*postgresql.conf*

max_replication_slots ≥ 1

Реплика

*recovery.conf*

primary_slot_name = '*имя*'



Слот репликации создается на мастере с помощью функции pg_create_physical_replication_slot(). Для этого параметр max_replication_slots должен быть выставлен в достаточно большое значение.

Состояние репликации, использующей слот, можно посмотреть в представлении pg_replication_slots.

Надо понимать, что если слот был создан и начал использоваться, но перестал, то это приведет к неограниченному накоплению сегментов WAL на мастере. Поэтому ненужные слоты необходимо удалять функцией pg_drop_replication_slot().

Чтобы реплика использовала созданный слот, в файле recovery.conf надо указать имя слота в параметре primary_slot_name.

Начиная с версии 9.6, утилита pg_basebackup тоже сможет использовать слот репликации. Этой возможностью полезно будет пользоваться при создании реплики без архива.

Утилита pg_receivexlog может использовать слот (ключ --slot) уже сейчас.

# Настройка обратной связи

Мастер

настройка не требуется

Реплика

*postgresql.conf*

hot_standby_feedback = on

wal_receiver_status_interval — *частота обратной связи (10 секунд)*



Для настройки обратной связи необходимо в файле postgresql.conf на реплике установить параметр hot_standby_feedback = on.

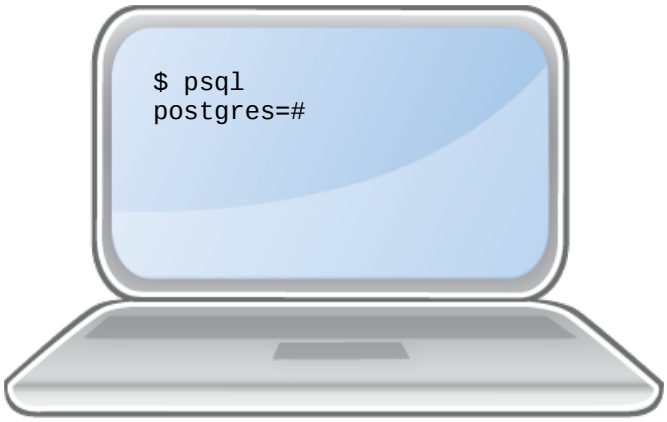
Кроме того, может понадобиться изменение параметра max_receiver_status_interval, задающего максимальный интервал между обновлениями информации. По умолчанию этот параметр равен 10 секундам.

На мастере никаких дополнительных настроек не требуется. Информацию обратной связи можно увидеть:

- в представлении pg_stat_replication (поле backend_xmin), если слот не используется,

- в представлении pg_replication_slots (поле xmin), если слот используется.

# Демонстрация



# Итоги

Потоковая репликация позволяет уменьшить отставание реплики до минимума

Мастер может учитывать интересы реплики: слоты репликации и обратная связь

Механизм слотов позволяет обходиться без архива, но требуется мониторинг

# Практика

1. Выполните необходимую настройку мастера для потоковой репликации с использованием слота, без непрерывного архивирования.
2. Создайте базовую резервную копию на месте каталога с данными реплики (сервер установлен под пользователем postgres2).
3. Выполните настройку реплики для восстановления в режиме горячего резерва с использованием слота и обратной связи.
4. Запустите реплику, создайте на мастере таблицу в базе данных DB11 и убедитесь, что она появилась на реплике.
5. Остановите реплику и вставьте много данных в таблицу. Измерьте размер сгенерированных при этом журнальных записей.
6. Снова запустите реплику и посмотрите, как изменяется задержка репликации.