



Файловая репликация

Задачи и виды репликации

Повторение: базовая резервная копия
и непрерывное архивирование

Резервный сервер с трансляцией журнальных файлов

Особенности и ограничения репликации

Использование реплики в качестве горячего резерва

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

По роли серверов

мастер-реплика

поток данных в одну сторону

мастер-мастер

поток данных в обе стороны

ВОЗМОЖНЫ КОНФЛИКТЫ

По передаваемым данным

физическая

сегменты WAL (файловая)
или записи WAL (потокковая)

двоичная совместимость

репликация всего кластера

логическая

команды SQL

совместимость на уровне команд

возможна выборочная репликация

Резервная копия

базовая резервная копия — `pg_basebackup`

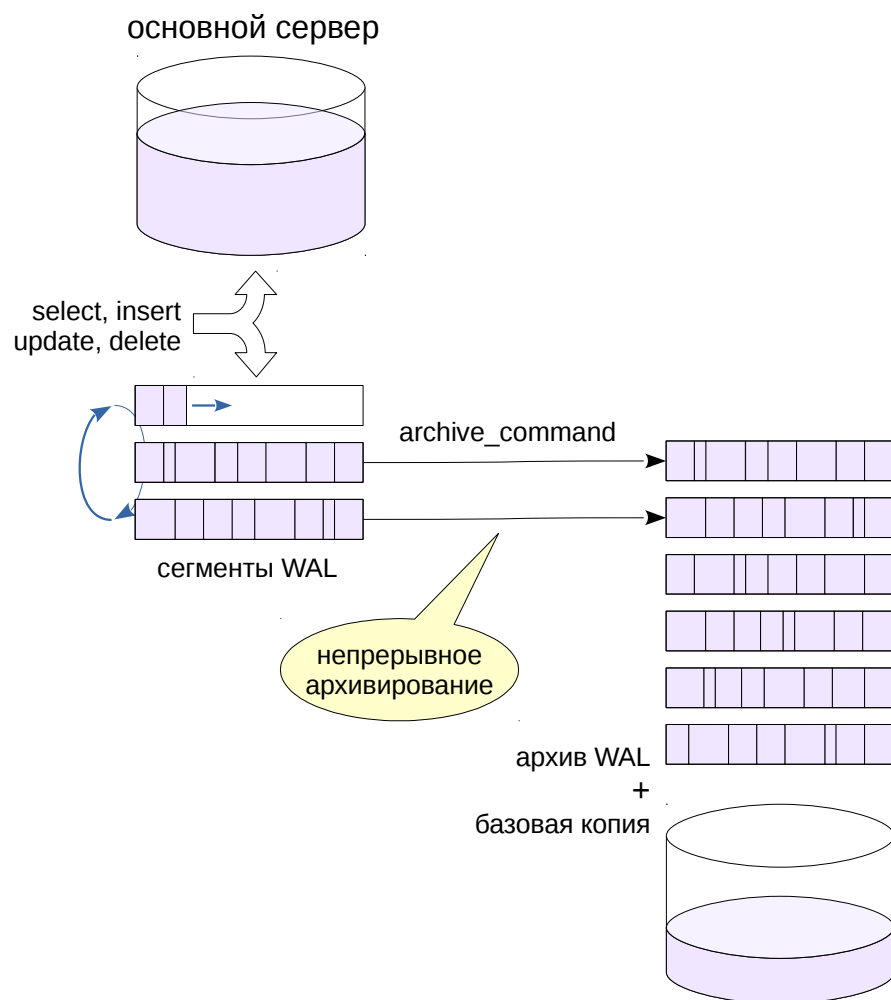
журналы упреждающей записи — непрерывное архивирование

Восстановление

разворачиваем резервную копию,
создаем управляющий файл `recovery.conf`
и запускаем сервер

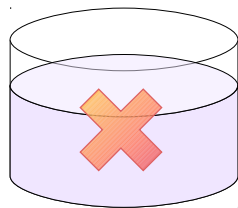
сервер восстанавливает согласованность,
применяет остальные журналы
и переходит в обычный режим работы

Резервное копирование

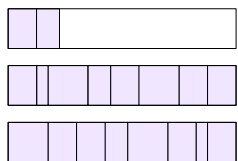


Резервное копирование

основной сервер

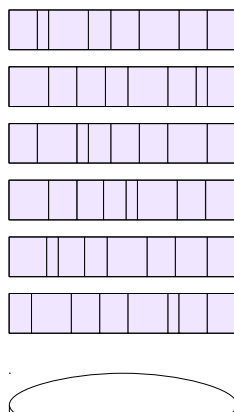


не попал
в архив



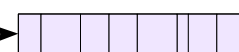
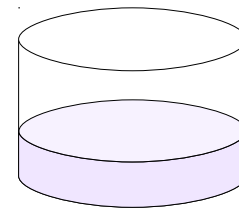
сегменты WAL

архив WAL
+
базовая копия



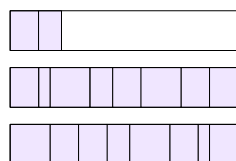
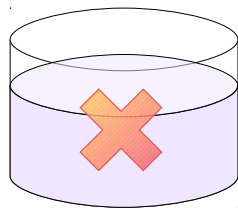
restore_command

резервный сервер



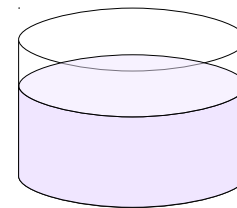
Резервное копирование

основной сервер

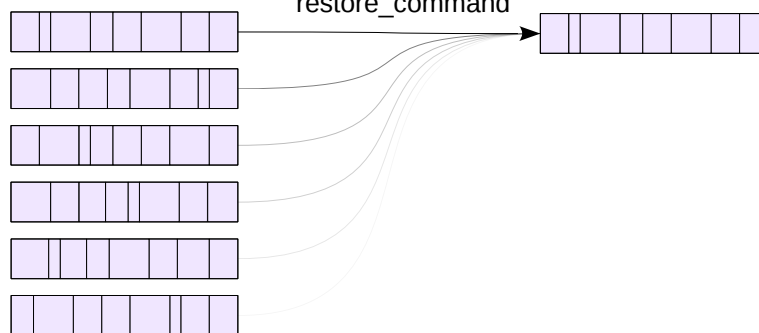


сегменты WAL

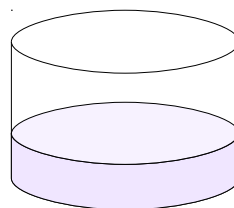
резервный сервер



restore_command

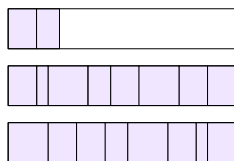
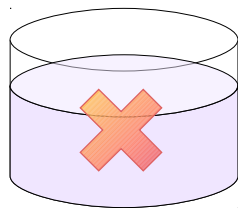


архив WAL
+
базовая копия



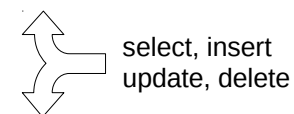
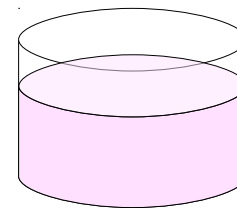
Резервное копирование

основной сервер

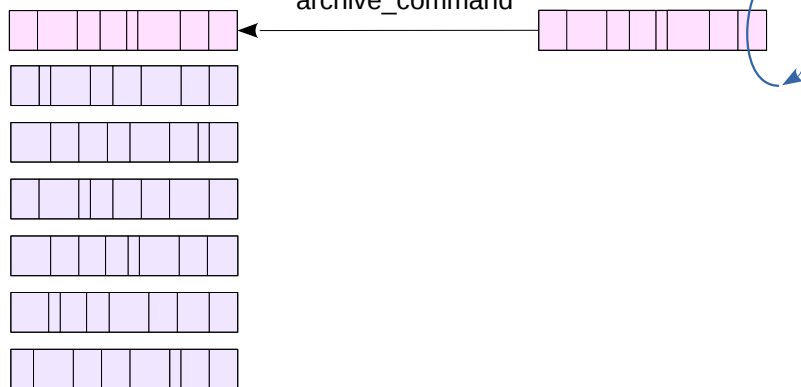


сегменты WAL

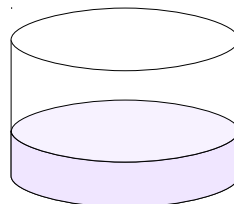
резервный сервер



archive_command



архив WAL
+
базовая копия



Резервная копия

базовая резервная копия — `pg_basebackup`

журналы упреждающей записи — непрерывное архивирование

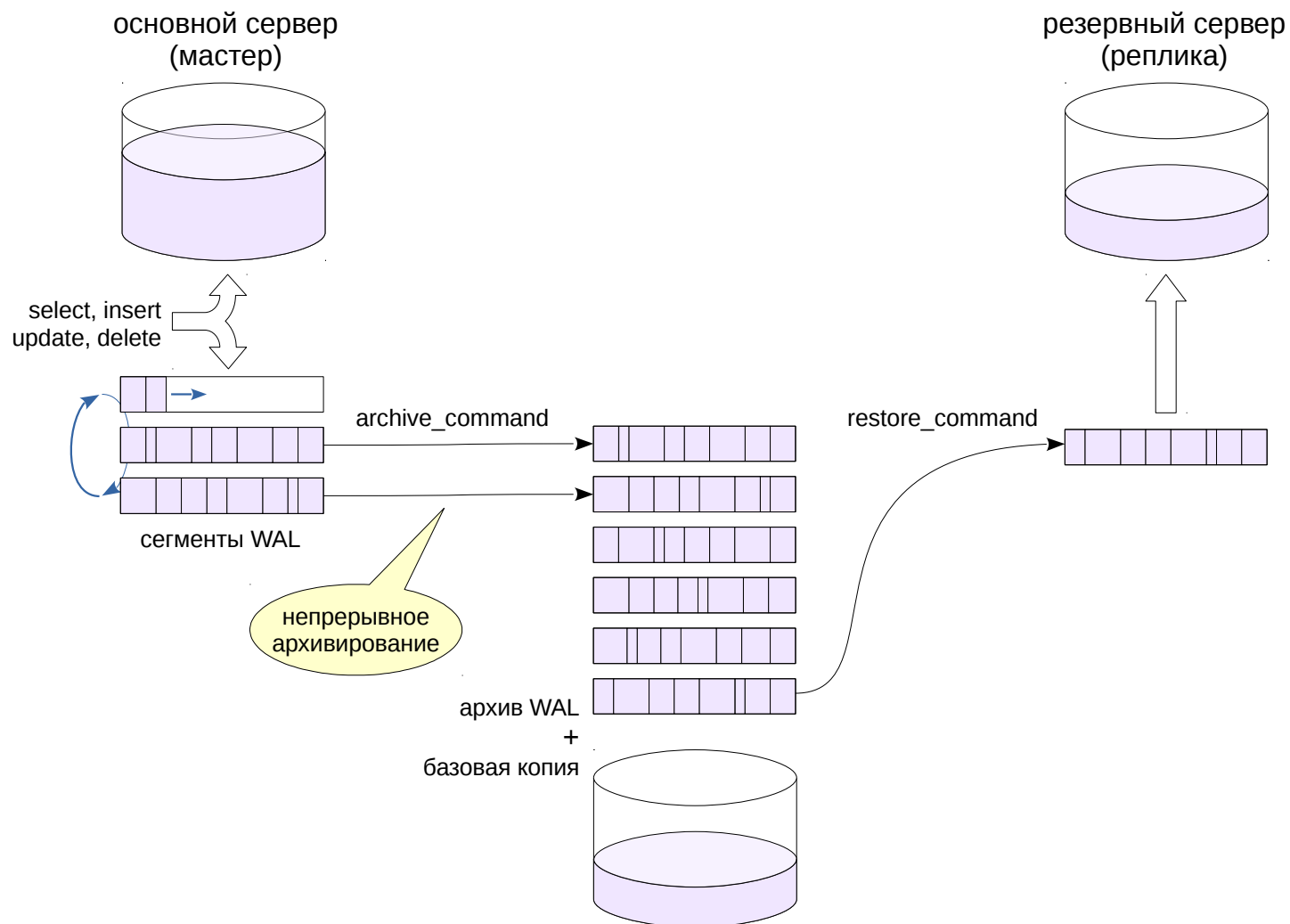
Непрерывное восстановление

разворачиваем резервную копию,
создаем управляющий файл `recovery.conf`
и запускаем сервер

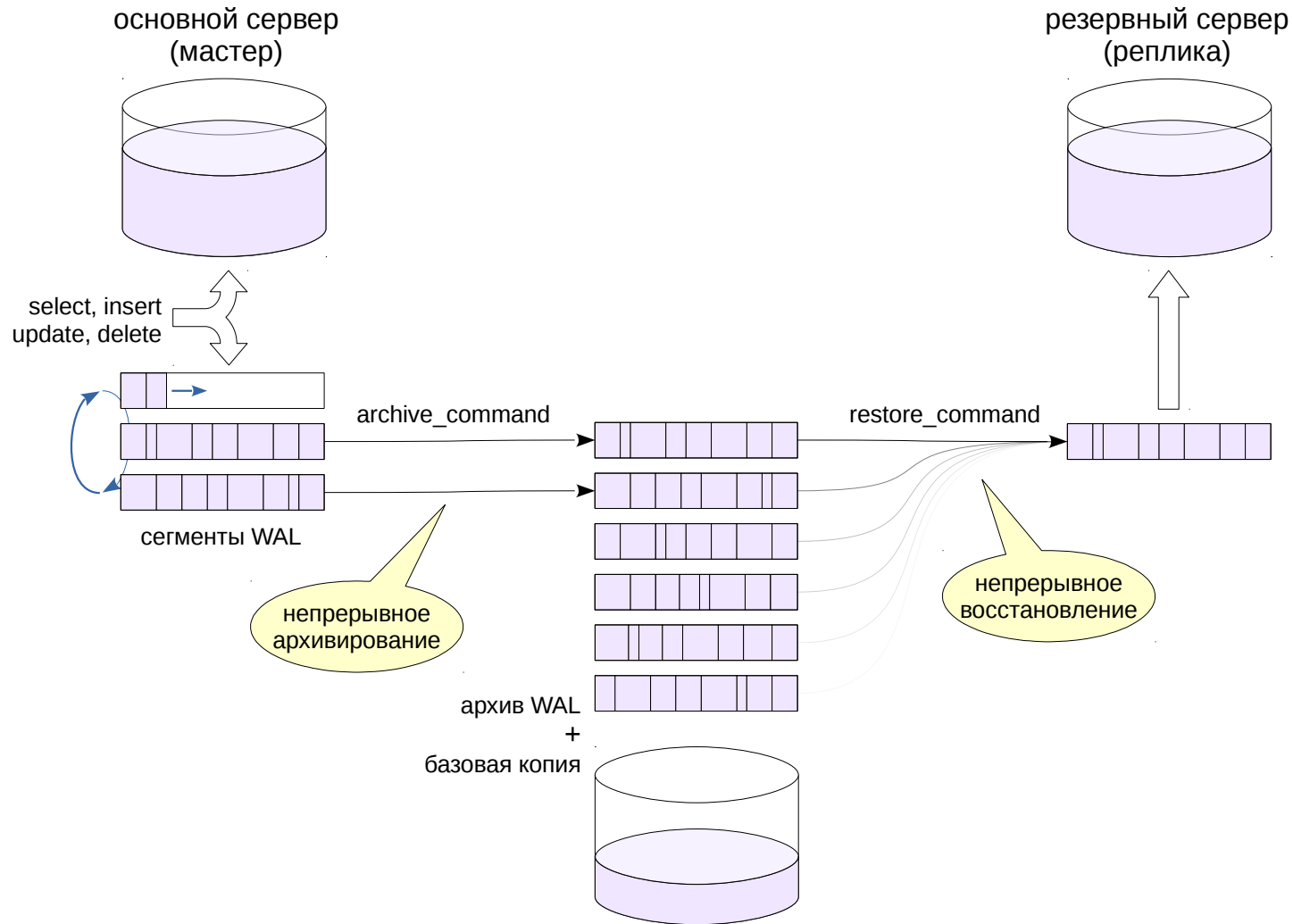
сервер восстанавливает согласованность
и продолжает применять поступающие журналы

подключения (только для чтения) разрешаются
сразу после восстановления согласованности

Репликация



Репликация



Разный набор процессов

мастер

wal writer

autovacuum launcher

archiver

(архивация)

checkpointer

(контрольные точки)

writer

stats collector

реплика

startup process

(восстановление)

checkpointer

(точки рестарта)

writer

stats collector

Непрерывное архивирование

postgresql.conf

wal_level = hot_standby

archive_mode = on

archive_command = ...

archive_timeout = ... — определяет отставание реплики

Базовая резервная копия

postgresql.conf

max_wal_senders \geq 1 (или 2 для потокового получения сегментов)

pg_hba.conf

разрешение для базы replication

Непрерывное восстановление

recovery.conf

```
standby_mode          = on
restore_command       = ...
recovery_target_timeline = 'latest'
```

Горячий резерв

postgresql.conf

```
hot_standby = on
```

Подключение

postgresql.conf

```
port = ... (если две СУБД на одном сервере)
```

Требования к командам

завершаются со статусом 0 только при успехе

используют символы подстановки:

%p	полный путь к сегменту WAL
%f	имя файла для сегмента WAL (<> basename %p)
%%	символ %

Пример

```
archive_command = 'test ! -f /архив/%f && cp %p /архив/%f &&  
chmod g+r /архив/%f'
```

```
restore_command = 'cp /архив/%f %p'
```

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...)

Ограничения

хэш-индексы не попадают в WAL

транзакции с числом вложенных транзакций больше 64 будут задерживать выполнение запросов из-за невозможности сделать снимок
не поддерживается уровень изоляции serializable

Особенности

не работают процессы autovacuum, wal writer

происходит запись на диск (временные файлы, служебная информация)

триггеры не срабатывают

пользовательские блокировки (advisory locks) не работают

Конфликты

эксклюзивная блокировка (например, drop table)

очистка удалила версию строки, нужную для снимка

Варианты

отменить запрос

— если важна высокая доступность

приостановить применение WAL и постараться завершить запрос

— если важно распределение нагрузки

не допустить конфликт

— если готовы отложить очистку на мастере

Параметры

`max_standby_archive_delay` (*реплика*)

— максимальное время ожидания запросов;
отсчет идет от момента получения WAL

`vacuum_defer_cleanup_age` (*мастер*)

— откладывание очистки на определенное число транзакций;
уменьшает вероятность конфликта, но без гарантий

Мониторинг

представление `pg_stat_database_conflicts` на реплике

Архив используется для резервного копирования

нужен полный архив с момента базовой резервной копии

Архив используется для «догона» мастера репликой

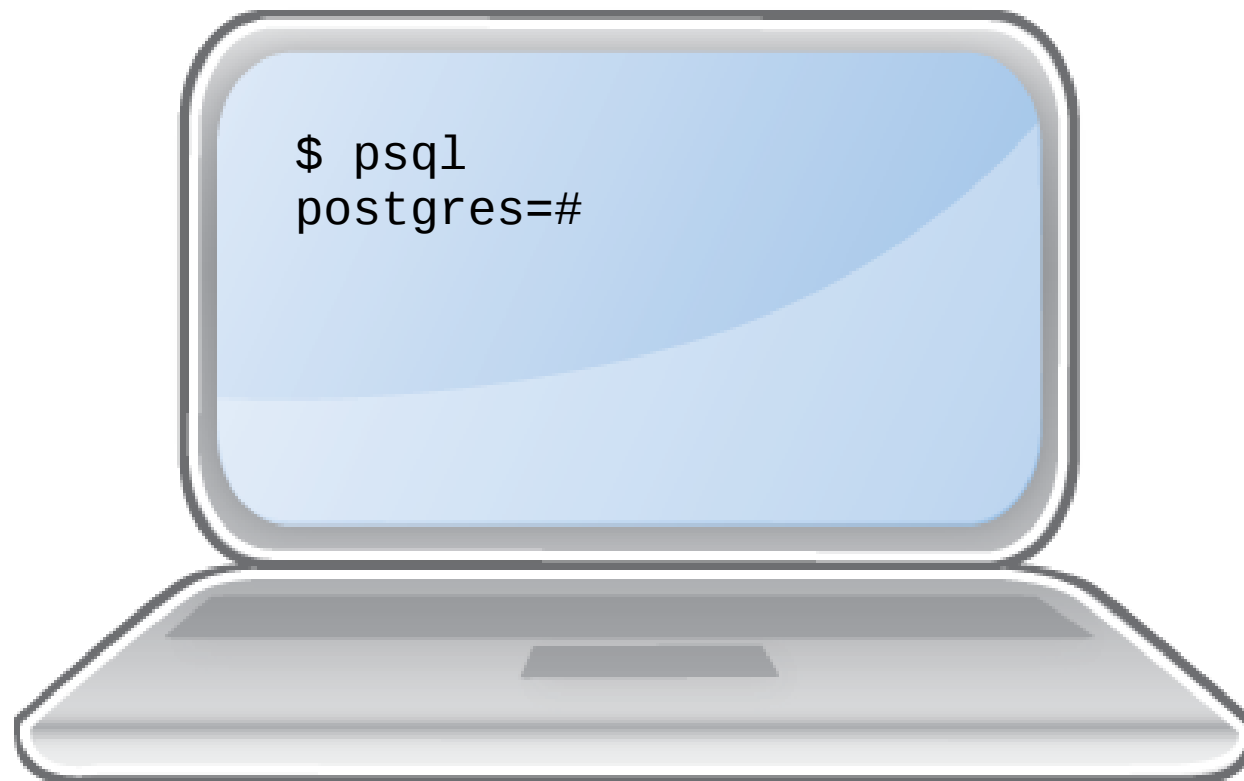
после применения всех сегментов архив нужен только для
восстановления реплики после сбоев (с момента точки рестарта)

специальная команда вызывается после каждой точки рестарта

recovery.conf

```
archive_cleanup_command = 'pg_archivecleanup /архив %r'
```

Демонстрация



Из разных видов репликации рассматриваем физическую

Простая схема с трансляцией журнальных файлов почти не отличается от резервного копирования

Реплика может использоваться для горячего резерва

Есть ряд проблем из-за конфликтов с мастером

1. Выполните необходимую настройку мастера для репликации, включая непрерывное архивирование.
Установите параметр `archive_timeout` в 10 секунд.
2. Создайте на мастере базы данных DB10 и таблицу.
3. Создайте базовую резервную копию и используйте ее для замены каталога с данными реплики (сервер установлен под пользователем `postgres2`).
4. Выполните настройку реплики для непрерывного восстановления в режиме горячего резерва.
5. Запустите реплику.
6. Выполните на мастере вставку строки в таблицу и проверьте, с какой задержкой это изменение реплицируется.



Авторские права

Курс «Администрирование PostgreSQL 9.5. Расширенный курс»

© Postgres Professional, 2016 год.

Авторы: Егор Рогов, Павел Лузанов

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Задачи и виды репликации

Повторение: базовая резервная копия
и непрерывное архивирование

Резервный сервер с трансляцией журнальных файлов

Особенности и ограничения репликации

Использование реплики в качестве горячего резерва

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 апгрейду 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.

По роли серверов

мастер-реплика

поток данных в одну сторону

мастер-мастер

поток данных в обе стороны
возможны конфликты

По передаваемым данным

физическая

сегменты WAL (файловая)
или записи WAL (потокковая)

двоичная совместимость

репликация всего кластера

логическая

команды SQL

совместимость на уровне команд

возможна выборочная репликация

Взаимодействие нескольких серверов можно организовать (по крайней мере, теоретически) совершенно разным образом. Соответственно будет отличаться и репликация.

Репликацию можно разделить по роли серверов. Сервер, выполняющий изменения данных, называется *мастером*. Сервер, позволяющий только читать данные, называется *репликой*. Простая репликация подразумевает один мастер и одну или несколько реплик. Поток данных в этом случае идет в одну сторону. Если допускается несколько мастеров, репликация становится существенно более сложной, так как при этом возможны конфликты и необходим механизм их разрешения.

Репликацию также можно разделить по виду данных, которые передаются между серверами. При физической репликации передаются двоичные данные — сегменты или отдельные записи журнала упреждающей записи (требуется двоичная совместимость; реплицируется весь кластер целиком). При логической репликации передаются команды SQL (требуется совместимость только на уровне команд; можно реплицировать часть объектов).

В этом курсе будет рассматриваться только физическая репликация с одним мастером, так как только она в настоящее время поддерживается без сторонних расширений. Есть надежда, что в ближайших новых версиях появится и логическая репликация на базе WAL (пока существует только в виде расширения [pg_logical](#)).

В этой теме мы познакомимся с репликацией, основанной на трансляции сегментов WAL.

Резервная копия

базовая резервная копия — `pg_basebackup`

журналы упреждающей записи — непрерывное архивирование

Восстановление

разворачиваем резервную копию,
создаем управляющий файл `recovery.conf`
и запускаем сервер

сервер восстанавливает согласованность,
применяет остальные журналы
и переходит в обычный режим работы

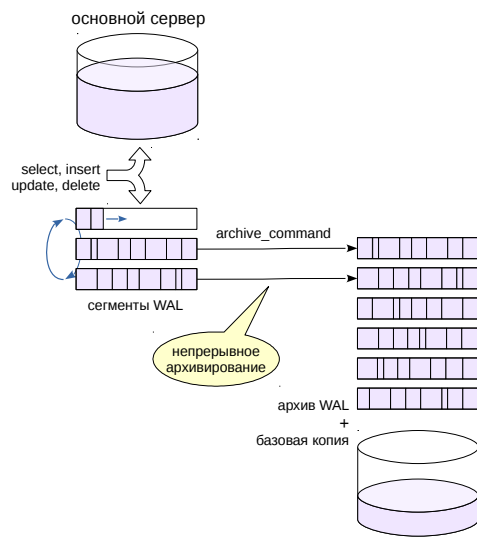
Репликация в PostgreSQL основана на тех же принципах, что и резервное копирование. Эта тема подробно раскрывалась в курсе DBA1, а сейчас вспомним, как это устроено.

Резервная копия состоит из:

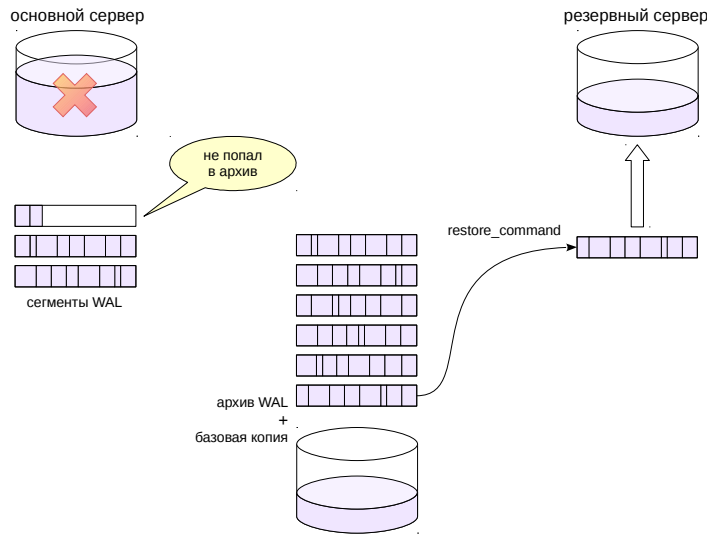
- базовой горячей копии файловой системы, содержащей несогласованные данные,
- архива журналов упреждающей записи.

Для восстановления требуется развернуть резервную копию файловой системы, создать управляющий файл `recovery.conf` (в котором указать как минимум команду получения сегментов WAL из архива) и запустить сервер.

Сервер переключается в режим восстановления согласованности данных, в котором он применяет все (по умолчанию) имеющиеся сегменты WAL, восстанавливая согласованность и «догоняя» состояние баз данных до актуального. После этого сервер переходит в обычный режим работы.

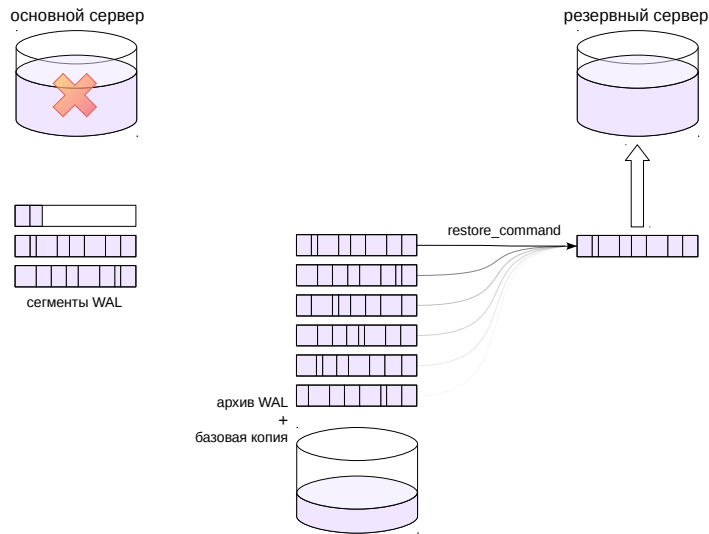


На этом рисунке представлен основной сервер. Он обрабатывает поступающие запросы; при этом формируются записи WAL и изменяется состояние баз данных (сначала в буферном кэше, потом на диске). Сегменты WAL циклически перезаписываются (точнее, удаляются старые сегменты, так как имена файлов уникальны). При этом настроено непрерывное архивирование: заполненные сегменты копируются в отдельный архив, где также заготовлена базовая резервная копия файловой системы.

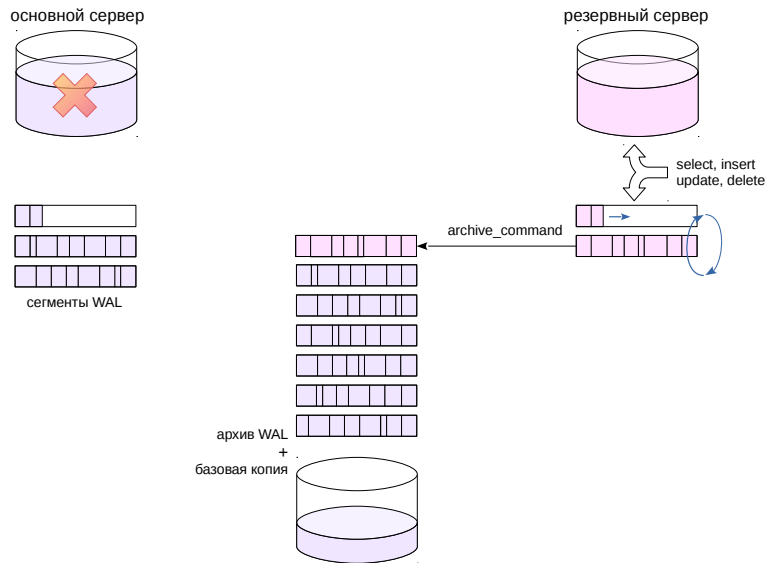


При выходе сервера из строя выполняется процедура восстановления. Для этого на другом (или на том же) сервере разворачивается базовая резервная копия. Сервер запускается и начинает чтение и применение сегментов WAL из архива.

Следует обратить внимание, что незаполненный сегмент WAL, не попавший в архив, следует подложить резервному серверу в каталог `pg_xlog` (если есть такая возможность). Конечно, таких сегментов может оказаться и несколько, но как следствие какого-то сбоя при архивировании.



Прочитав все сегменты WAL из архива (а после этого — все дополнительные сегменты WAL из каталога pg_xlog), резервный сервер «догоняет» состояние баз данных до актуального. Максимально возможная потеря — незаполненный сегмент WAL, не попавший в архив, если его по каким-то причинам невозможно скопировать.



После этого резервный сервер переходит в обычный режим работы, принимает запросы, записывает сегменты WAL в архив и так далее, выступая в качестве нового полноценного основного сервера.

По этой причине резервный сервер имеет смысл располагать на таком же, или, по крайней мере, на сравнимом по характеристикам аппаратуре сервере.

Резервная копия

базовая резервная копия — `pg_basebackup`

журналы упреждающей записи — непрерывное архивирование

Непрерывное восстановление

разворачиваем резервную копию,
создаем управляющий файл `recovery.conf`
и запускаем сервер

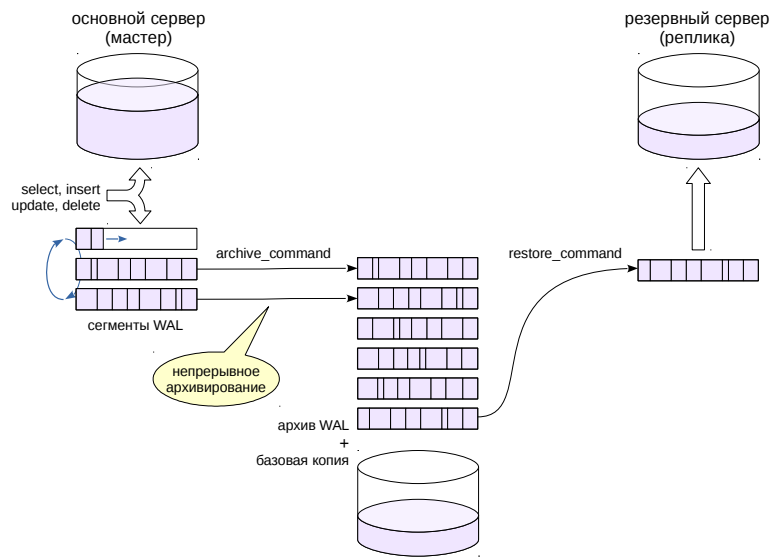
сервер восстанавливает согласованность
и продолжает применять поступающие журналы

подключения (только для чтения) разрешаются
сразу после восстановления согласованности

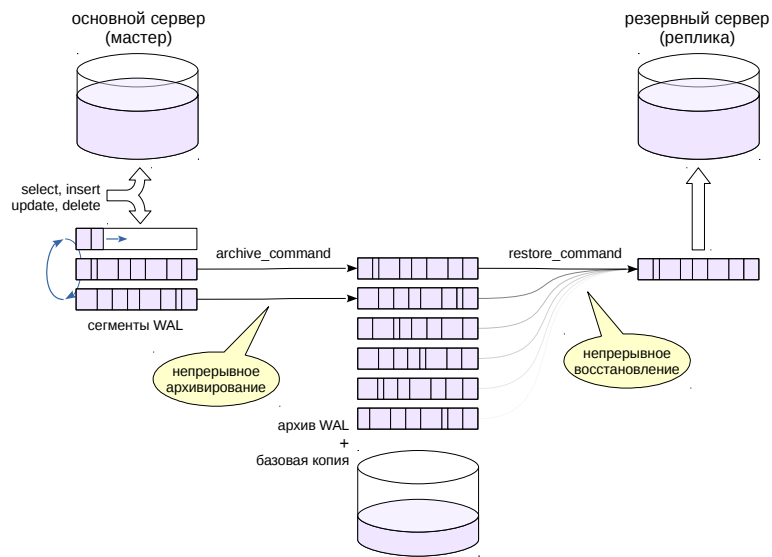
Чем отличается настройка репликации? По большому счету тем, что резервная копия поднимается сразу, не дожидаясь поломки основного сервера. При этом она работает в режиме постоянного восстановления, все время читая и применяя новые сегменты WAL, приходящие с мастера. Таким образом, реплика постоянно поддерживается в почти актуальном состоянии, отставая от мастера на время заполнения сегмента WAL. И в случае сбоя мы имеем сервер, готовый подхватить работу.

Если реплика не допускает подключений, она называется «теплым резервом». Однако можно сделать и «горячий резерв» — в процессе восстановления реплика будет допускать подключения для чтения данных (как только будет восстановлена согласованность данных).

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).



На этом рисунке показан основной сервер (мастер) и развернутая из базовой резервной копии реплика, которая приступает к восстановлению.



После того, как реплика прочитает и применит все сегменты WAL, база данных будет поддерживаться в (почти) актуальном состоянии.

Разный набор процессов

<i>мастер</i>	<i>реплика</i>
wal writer	
autovacuum launcher	
archiver (архивация)	startup process (восстановление)
checkpointer (контрольные точки)	checkpointer (точки рестарта)
writer	writer
stats collector	stats collector

На мастере и на реплике выполняется разный набор процессов.

Общие процессы:

- background writer — записывает грязные страницы из буферного кэша на диск. На реплике страницы тоже могут меняться (не сами данные, а биты-подсказки) и тоже будут записываться на диск.
- stats collector — статистика не передается через WAL, а собирается на реплике отдельно.
- checkpointer — на реплике выполняется похожая на контрольную точку процедура — точка рестарта. Если в процессе восстановления случится сбой, реплика сможет продолжить с последней точки рестарта, а не с самого начала.

Процессы только на мастере:

- wal writer — реплика не создает журналы упреждающей записи, а только получает их с мастера.
- autovacuum launcher — реплика не выполняет очистку, ее выполнение на мастере передается реплике через WAL.
- archiver — реплика не записывает сегменты WAL в архив.

Процесс только на реплике:

- startup process — занимается восстановлением и в каком-то смысле является парой к процессу архивации на мастере. При обычной работе этот процесс отвечает за восстановление после сбоя и быстро завершается.

Непрерывное архивирование

postgresql.conf

wal_level = hot_standby

archive_mode = on

archive_command = ...

archive_timeout = ... — определяет отставание реплики

Базовая резервная копия

postgresql.conf

max_wal_senders \geq 1 (или 2 для потокового получения сегментов)

pg_hba.conf

разрешение для базы replication

Для того, чтобы использовать сервер в качестве мастера репликации, необходимо настроить непрерывное архивирование и иметь возможность сделать базовую резервную копию.

Настройки в *postgresql.conf*:

- wal_level = hot_standby (дополнительно к уровню archive включает информацию о статусе транзакций; если не нужен горячий резерв, то достаточно уровня archive)
- archive_mode = on (включаем непрерывное архивирование)
- archive_command задает команду для копирования сегментов WAL
- archive_timeout определяет максимальный интервал времени между переключениями на новый сегмент WAL (при высокой активности сегменты будут переключаться чаще при заполнении). Этот параметр определяет максимальное время, на которое реплика будет отставать от мастера.
- max_wal_senders должно быть больше 1 для работы pg_basebackup (или 2, если используется --xlog-method=stream)

Кроме того, для pg_basebackup надо разрешить подключение по протоколу репликации в *pg_hba.conf*.

Непрерывное восстановление

```
recovery.conf
standby_mode          = on
restore_command       = ...
recovery_target_timeline = 'latest'
```

Горячий резерв

```
postgresql.conf
hot_standby = on
```

Подключение

```
postgresql.conf
port = ... (если две СУБД на одном сервере)
```

После развертывания резервной копии мастера на месте реплики, в настройки необходимо внести изменения:

- hot_standby = on (если нужен горячий резерв)
- port надо изменить, если реплика будет работать на том же физическом сервере (это актуально для учебного примера, но маловероятно на практике)

Файл recovery.conf, управляющий восстановлением, должен содержать следующие параметры:

- standby_mode = on (включает режим непрерывного восстановления)
- restore_command определяет команду для копирования сегментов WAL из архива
- recovery_target_timeline = 'latest' (восстановление должно следовать за мастером, а не быть в другой ветви времени)

Требования к командам

завершаются со статусом 0 только при успехе

используют символы подстановки:

%p	полный путь к сегменту WAL
%f	имя файла для сегмента WAL (<> basename %p)
%%	символ %

Пример

```
archive_command = 'test ! -f /архив/%f && cp %p /архив/%f &&
chmod g+r /архив/%f'
restore_command = 'cp /архив/%f %p'
```

16

Команды архивирования и восстановления могут быть произвольными, но они должны завершаться с нулевым статусом только в случае успеха.

Команда архивирования не должна перезаписывать уже существующие файлы, так как это скорее всего означает какую-то ошибку и перезапись файла может погубить архив.

Удобно включить сбор сообщений с помощью `logging_collector`, так как в этом случае сообщения об ошибках будут попадать в журнал сервера.

В командах используются символы подстановки `%f` (имя файла сегмента WAL) и `%p` (полный путь к этому файлу). Чтобы получить `%`, надо использовать удвоенный символ: `%%`.

На слайде приведены простые примеры команд, достаточные для учебных целей (подробно эта тема освещалась в курсе DBA1). Команда `chmod` добавлена для того, чтобы сегменты были доступны двум пользователям, из-под которых работает мастер и реплика. По умолчанию сегменты доступны только владельцу (`rw- --- ---`).

В реальной жизни архив обычно располагается на отдельном сервере, куда сегменты WAL копируются по сети.

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...)

В режиме горячего резерва на реплике не допускаются любые изменения данных (включая последовательности), блокировки, команды DDL, такие 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).

Ограничения

- хэш-индексы не попадают в WAL
- транзакции с числом вложенных транзакций больше 64 будут задерживать выполнение запросов из-за невозможности сделать снимок
- не поддерживается уровень изоляции serializable

Особенности

- не работают процессы autovacuum, wal writer
- происходит запись на диск (временные файлы, служебная информация)
- триггеры не срабатывают
- пользовательские блокировки (advisory locks) не работают

У репликации есть ряд ограничений и особенностей.

Хэш-индексы не попадают в WAL и соответственно не передаются на реплику (их вообще не стоит использовать).

Если на мастере выполняется транзакция с большим числом вложенных транзакций (то есть активно используются точки сохранения), создание снимка на реплике будет приостановлено — это означает невозможность некоторое время выполнять запросы.

Не поддерживается уровень изоляции serializable на реплике.

Надо понимать, что на реплике происходит не только чтение данных при выполнении запросов, но и запись на диск — в процессе точек рестарта, в процессе выполнения запросов (временные файлы).

Триггеры не будут срабатывать на реплике (поскольку репликация физическая, а не логическая).

Не будут работать и пользовательские (advisory) блокировки.

Конфликты

- эксклюзивная блокировка (например, drop table)
- очистка удалила версию строки, нужную для снимка

Варианты

- отменить запрос
 - если важна высокая доступность
- приостановить применение WAL и постараться завершить запрос
 - если важно распределение нагрузки
- не допустить конфликт
 - если готовы отложить очистку на мастере

В ходе работы с мастера могут прийти записи WAL, конфликтующие с выполняющимися запросами.

Возможны разные ситуации. Примером является эксклюзивная блокировка (например, при удалении таблицы). Другой пример — удаление процессом очистки на мастере версии строки, которая на мастере уже не нужна, а на реплике еще нужна. Проблема здесь в том, что мастер не в курсе запросов, выполняющихся на реплике, и не учитывает их при очистке.

Вариантов несколько.

Можно отменить на реплике конфликтующий запрос. Это разумно, если высокая доступность реплики важнее отдельных запросов.

Можно отложить применение сегментов WAL на некоторое время в надежде на то, что запрос успеет завершиться. При этом отставание реплики от мастера может увеличиться.

Можно постараться не допустить конфликт, откладывая очистку на мастере (с блокировками это не проходит).

Параметры

`max_standby_archive_delay` (*реплика*)

— максимальное время ожидания запросов;
отсчет идет от момента получения WAL

`vacuum_defer_cleanup_age` (*мастер*)

— откладывание очистки на определенное число транзакций;
уменьшает вероятность конфликта, но без гарантий

Мониторинг

представление `pg_stat_database_conflicts` на реплике

Для управления конфликтами могут использоваться два параметра.

- `max_standby_archive_delay` на реплике. Определяет максимальное время, на которое может приостанавливаться применение сегментов WAL. Это время отсчитывается от момента получения сегмента, а не от момента начала запроса. По умолчанию — 30 секунд.

- `vacuum_defer_cleanup_age` на мастере. Откладывает выполнение очистки на заданное число транзакций (не на время). За счет увеличения этого параметра можно уменьшить вероятность конфликта, но без гарантии его предотвращения. Обратная сторона — возможно увеличение таблиц в размере из-за отложенной очистки. По умолчанию этот параметр равен нулю (откладывание не происходит).

Количество возникающих конфликтов можно посмотреть на реплике с помощью представления `pg_stat_database_conflicts`.

Архив используется для резервного копирования

нужен полный архив с момента базовой резервной копии

Архив используется для «догона» мастера репликой

после применения всех сегментов архив нужен только для восстановления реплики после сбоев (с момента точки рестарта) специальная команда вызывается после каждой точки рестарта

recovery.conf

```
archive_cleanup_command = 'pg_archivecleanup /архив %r'
```

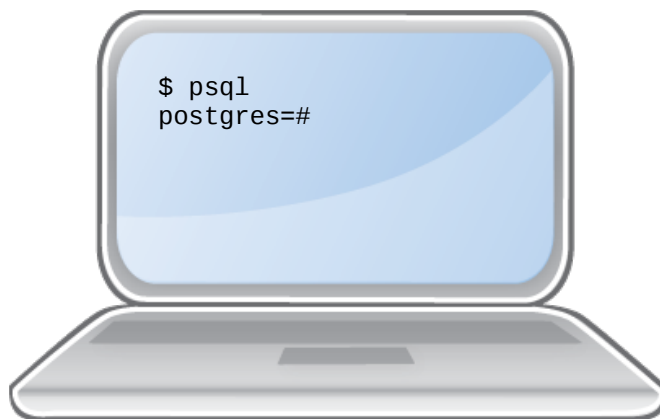
Если архив используется для поддержки резервного копирования, то он должен сохраняться полностью с момента создания наиболее ранней базовой резервной копии, которая еще нужна.

Если же архив нужен только для одноразового «разгона» реплики, то из него нужны только файлы с момента последней точки рестарта (чтобы реплика могла восстановиться после сбоя).

Для такого случая в конфигурационном файле *recovery.conf* можно прописать специальную команду, которая будет выполняться после каждой точки рестарта. В команде можно использовать символ подстановки *%r*, равный первому еще нужному номеру сегмента WAL (то есть более ранние сегменты можно удалять). Удаление удобно делать с помощью расширения *pg_archivecleanup*, специально созданного для этого.

<http://www.postgresql.org/docs/current/static/archive-recovery-settings.html>

<http://www.postgresql.org/docs/current/static/pgarchivecleanup.html>



Из разных видов репликации рассматриваем физическую

Простая схема с трансляцией журнальных файлов
почти не отличается от резервного копирования

Реплика может использоваться для горячего резерва

Есть ряд проблем из-за конфликтов с мастером

1. Выполните необходимую настройку мастера для репликации, включая непрерывное архивирование. Установите параметр `archive_timeout` в 10 секунд.
2. Создайте на мастере базы данных DB10 и таблицу.
3. Создайте базовую резервную копию и используйте ее для замены каталога с данными реплики (сервер установлен под пользователем `postgres2`).
4. Выполните настройку реплики для непрерывного восстановления в режиме горячего резерва.
5. Запустите реплику.
6. Выполните на мастере вставку строки в таблицу и проверьте, с какой задержкой это изменение реплицируется.