DIASOFT
всё по-настоящему

Q.DATABASE, УПРАВЛЕНИЕ БД

Руководство администратора

Диасофт
2022

Содержание

1. Общие положения 4
2. Термины и сокращения 5
3. Администрирование сервера 6
3.1 Подготовка к работе и сопровождение сервера 6
3.1.1 Учетная запись пользователя PostgreDB 6
3.1.2 Создание кластера баз данных 6
3.1.3 Запуск сервера базы данных 7
3.1.4 Выключение сервера 7
3.1.5 Возможности шифрования 7
3.2 Настройка сервера 8
3.2.1 Имена и значения параметров 8
3.2.2 Определение параметров в файле конфигурации 9
3.2.3 Управление параметрами через SQL 10
3.2.4 Управление параметрами в командной строке 12
3.2.5 Упорядочение содержимого файлов конфигурации 12
3.2.6 Репликация 26
3.2.7 Планирование запросов 32
3.2.8 Регистрация ошибок и протоколирование работы сервера 34
3.2.9 Параметры клиентских сеансов по умолчанию 47
3.2.10 Предопределённые параметры 49
3.3 Аутентификация клиентского приложения 53
3.3.1 Файл pg_hba.conf 53
3.4 Роли базы данных 59
3.5 Управление базами данных 60
3.5.1 Создание базы данных 60
3.5.2 Удаление базы данных 61
3.5.3 Локализация 61
3.5.4 Регламентные задачи обслуживания базы данных 61
3.5.5 Резервное копирование и восстановление 63
3.5.6 Мониторинг работы СУБД 84

1. Общие положения

В данном руководстве рассматриваются пользователи и их полномочия, использование главных записей, полномочий суперпользователя, генератора профилей, генерации меню предприятия и другие вопросы, непосредственно относящиеся к безопасности информации.

2. Термины и сокращения

Термин/Сокращение	Определение
БД	База данных
СУБД PostgreDB	Система управления базами данных PostgreDB

3. Администрирование сервера

3.1 Подготовка к работе и сопровождение сервера

3.1.1 Учетная запись пользователя PostgreDB

В случае, если установка программного компонента «Система управления базами данных PostgreDB» выполнялась стандартным образом, учётная запись Linux будет создана автоматически (если пользователь postgres отсутствует в системе).

Эта учётная запись имеет доступ только к файлам с данными кластера БД и к журнальным файлам. Устанавливать пользователя postgres владельцем исполняемых программ не рекомендуется.

3.1.2 Создание кластера баз данных

Кластер баз данных представляет собой набор баз, управляемых одним экземпляром работающего сервера. После инициализации кластер будет содержать базу данных с именем postgres, предназначенную для использования по умолчанию утилитами, пользователями и сторонними приложениями. Сам сервер баз данных не требует наличия базы postgres, но многие внешние вспомогательные программы рассчитывают на её существование. При инициализации в каждом кластере создаётся ещё одна база, с именем template1. Она применяется впоследствии в качестве шаблона создаваемых баз данных; использовать её в качестве рабочей не следует.

С точки зрения файловой системы кластер баз данных представляет собой один каталог, в котором будут храниться все данные (каталог данных или область данных).

Местом для хранения данных может быть любой каталог файловой системы. Чаще всего данные размещают в `/usr/local/pgsql/data` или в `/var/lib/pgsql/data`.

Для инициализации кластера выполните команду: `$ initdb -D /usr/local/pgsql/data`

Команду нужно выполнять от имени пользователя PostgreDB. Программа initdb устанавливается в составе PostgreDB.

3.1.3 Запуск сервера базы данных

Для возможности обращения к базе данных необходимо запустить сервер базы данных.

Запуск сервера БД выполняется привилегированными пользователем root.

Для запуска сервера базы данных выполнить следующие команды: `systemctl enable postgresql`, `systemctl start postgresql`.

3.1.4 Выключение сервера

Сервер баз данных можно отключить несколькими способами. На практике они все сводятся к отправке сигнала управляющему процессу postgres. Удобную возможность отправлять эти сигналы, отключающие сервер, предоставляет программа `pg_ctl`. Кроме того, соответствующий сигнал можно отправить с помощью команды `kill`. PID основного процесса postgres можно узнать, воспользовавшись программой `ps`, либо прочитав файл `postmaster.pid` в каталоге данных.

3.1.5 Возможности шифрования

PostgreDB обеспечивает шифрование на разных уровнях и даёт гибкость в выборе средств защиты данных в случае кражи сервера, от недобросовестных администраторов или в небезопасных сетях. Шифрование может также требоваться для защиты конфиденциальных данных, например, медицинских сведений или финансовых транзакций.

3.1.5.1 Шифрование паролей

Пароли пользователей базы данных хранятся в виде хешей (алгоритм хеширования определяется параметром `password_encryption`), так что администратор не может узнать, какой именно пароль имеет пользователь. Если шифрование SCRAM или MD5 применяется и при проверке подлинности, пароль не присутствует на сервере в открытом виде даже кратковременно, так как клиент шифрует его перед тем как передавать по сети. Предпочтительным методом является SCRAM, так как это стандарт, принятый в Интернете, и он более безопасен, чем собственный протокол проверки MD5 в PostgreSQL.

3.2 Настройка сервера

3.2.1 Имена и значения параметров

Имена всех параметров являются регистронезависимыми. Каждый параметр принимает значение одного из пяти типов: логический, строка, целое, число с плавающей точкой или перечисление. От типа значения зависит синтаксис установки этого параметра:

- *Логический:* значения могут задаваться строками `on`, `off`, `true`, `false`, `yes`, `no`, `1`, `0` (регистр не имеет значения), либо как достаточно однозначный префикс одной из этих строк.
- *Строка:* Обычно строковое значение заключается в апострофы (при этом внутренние апострофы дублируются). Однако если значение является простым числом или идентификатором, апострофы обычно можно опустить. (Значения, совпадающие с ключевыми словами SQL, всё же требуют заключения в апострофы в некоторых контекстах.)
- *Число (целое или с плавающей точкой):* Значения числовых параметров могут задаваться в обычных форматах, принятых для целых чисел или чисел с плавающей точкой; если параметр целочисленный, дробные значения округляются до ближайшего целого. Кроме того, целочисленные параметры принимают значения в шестнадцатеричном (с префиксом `0x`) и восьмеричном (с префиксом `0`) виде, но дробная часть в таких случаях исключена. Разделители разрядов в значениях использовать нельзя. Заключать в кавычки требуется только значения в шестнадцатеричном виде.
- *Число с единицей измерения:* Некоторые числовые параметры задаются с единицами измерения, так как они описывают количества информации или времени. Единицами могут быть байты, килобайты, блоки (обычно восемь килобайт), миллисекунды, секунды или минуты. При указании только числового значения для такого параметра единицей измерения будет считаться установленная для него единица по умолчанию, которая указывается в `pg_settings.unit`. Для удобства параметры также можно задавать, указывая единицу измерения явно, например, задать `'120 ms'` для значения времени. При этом такое значение будет переведено в основную единицу измерения параметра. Заметьте, что для этого значение должно записываться в виде

строки (в апострофах). Имя единицы является регистронезависимым, а между ним и числом допускаются пробельные символы.

- Допустимые единицы информации: `B` (байты), `kB` (килобайты), `MB` (мегабайты), `GB` (гигабайты) и `TB` (терабайты). Множителем единиц информации считается 1024, не 1000.
- Допустимые единицы времени: `us` (микросекунды), `ms` (миллисекунды), `s` (секунды), `min` (минуты), `h` (часы) и `d` (дни).

Если с единицей измерения задаётся дробное значение, оно будет округлено до следующей меньшей единицы, если такая имеется.

Если параметр имеет целочисленный тип, после преобразования единицы измерения значение окончательно округляется до целого.

Перечисление: Параметры, имеющие тип перечисление, записываются так же, как строковые параметры, но могут иметь только ограниченный набор значений. Список допустимых значений такого параметра задаётся в `pg_settings.enumvals`. В значениях перечислений регистр не учитывается.

3.2.2 Определение параметров в файле конфигурации

Самый основной способ установки этих параметров — определение их значений в файле `postgresql.conf`, который обычно находится в каталоге данных. При инициализации каталога кластера БД в этот каталог помещается копия стандартного файла.

Основной процесс сервера перечитывает файл конфигурации заново, получая сигнал SIGHUP; послать его проще всего можно, запустив `pg_ctl reload` в командной строке или вызвав SQL-функцию `pg_reload_conf()`. Основной процесс сервера передаёт этот сигнал всем остальным запущенным серверным процессам, так что существующие сеансы тоже получают новые значения (после того, как завершится выполнение текущей команды клиента). Также возможно послать этот сигнал напрямую одному из серверных процессов. Некоторые параметры можно установить только при запуске сервера; любые изменения их значений в файле конфигурации не будут

учитываться до перезапуска сервера. Более того, при обработке SIGHUP игнорируются неверные значения параметров (но об этом сообщается в журнале).

В дополнение к `postgresql.conf` в каталоге данных PostgreDB содержится файл `postgresql.auto.conf`, который имеет тот же формат, что и `postgresql.conf`, но предназначен для автоматического изменения, а не для редактирования вручную. Этот файл содержит параметры, задаваемые командой `ALTER SYSTEM`. Он считывается одновременно с `postgresql.conf` и заданные в нём параметры действуют таким же образом. Параметры в `postgresql.auto.conf` переопределяют те, что указаны в `postgresql.conf`.

Вносить изменения в `postgresql.auto.conf` можно и с использованием внешних средств. Однако это не рекомендуется делать в процессе работы сервера, так эти изменения могут быть потеряны при параллельном выполнении команды `ALTER SYSTEM`. Внешние программы могут просто добавлять новые определения параметров в конец файла или удалять повторяющиеся определения и/или комментарии (как делает `ALTER SYSTEM`).

Системное представление `pg_file_settings` может быть полезным для предварительной проверки изменений в файлах конфигурации или для диагностики проблем, если сигнал `SIGHUP` не даёт желаемого эффекта.

3.2.3 Управление параметрами через SQL

В PostgreDB есть три SQL-команды, задающие для параметров значения по умолчанию. Уже упомянутая команда `ALTER SYSTEM` даёт возможность изменять глобальные значения средствами SQL; она функционально равнозначна редактированию `postgresql.conf`. Кроме того, есть ещё две команды, которые позволяют задавать значения по умолчанию на уровне баз данных и ролей:

- Команда `ALTER DATABASE` позволяет переопределить глобальные параметры на уровне базы данных.

- Команда `ALTER ROLE` позволяет переопределить для конкретного пользователя как глобальные, так и локальные для базы данных параметры.

Значения, установленные командами `ALTER DATABASE` и `ALTER ROLE`, применяются только при новом подключении к базе данных. Они переопределяют значения, полученные из файлов конфигурации или командной строки сервера, и применяются по умолчанию в рамках сеанса. Заметьте, что некоторые параметры невозможно изменить после запуска сервера, поэтому их нельзя установить этими командами (или командами, перечисленными ниже).

Когда клиент подключён к базе данных, он может воспользоваться двумя дополнительными командами SQL (и равнозначными функциями), которые предоставляет PostgreDB для управления параметрами конфигурации:

- Команда `SHOW` позволяет узнать текущее значение любого параметра. Ей соответствует SQL-функция `current_setting(setting_name text)`.
- Команда `SET` позволяет изменить текущее значение тех параметров, которые устанавливаются локально в рамках сеанса; на другие сеансы она не влияет. Ей соответствует SQL-функция `set_config(setting_name, new_value, is_local)`.

Кроме того, просмотреть и изменить значения параметров для текущего сеанса можно в системном представлении pg_settings:

- Запрос на чтение представления выдаёт ту же информацию, что и `SHOW ALL`, но более подробно. Этот подход и более гибкий, так как в нём можно указать условия фильтра или связать результат с другими отношениями.

Выполнение `UPDATE` для этого представления, а именно присваивание значения столбцу `setting`, равносильно выполнению команды `SET`.

3.2.4 Управление параметрами в командной строке

Помимо изменения глобальных значений по умолчанию и переопределения их на уровне базы данных или роли, параметры PostgreDB можно изменить, используя средства командной строки. Управление через командную строку поддерживают и сервер, и клиентская библиотека `libpq`.

При запуске сервера, значения параметров можно передать команде postgres в аргументе командной строки `-c`.

При запуске клиентского сеанса, использующего `libpq`, значения параметров можно указать в переменной окружения `PGOPTIONS`. Заданные таким образом параметры будут определять значения по умолчанию на время сеанса, но никак не влияют на другие сеансы. Формат `PGOPTIONS` похож на тот, что применяется при запуске команды postgres; в частности, в нём должен присутствовать флаг `-c`.

3.2.5 Упорядочение содержимого файлов конфигурации

PostgreDB предоставляет несколько возможностей для разделения сложных файлов `postgresql.conf` на вложенные файлы. Эти возможности особенно полезны при управлении множеством серверов с похожими, но не одинаковыми конфигурациями.

Помимо присваиваний значений параметров, `postgresql.conf` может содержать *директивы включения* файлов, которые будут прочитаны и обработаны, как если бы их содержимое было вставлено в данном месте файла конфигурации. Это позволяет разбивать файл конфигурации на физически отдельные части. Директивы включения записываются в формате: `include 'имя_файла'`

Если имя файла задаётся не абсолютным путём, оно рассматривается относительно каталога, в котором находится включающий файл конфигурации. Включения файлов могут быть вложенными. Кроме того, есть директива `include_if_exists`, которая работает подобно `include`, за исключением случаев, когда включаемый файл не существует или не может быть прочитан. Обычная директива `include` считает это критической ошибкой, но `include_if_exists` просто выводит сообщение и продолжает обрабатывать текущий файл конфигурации.

Файл `postgresql.conf` может также содержать директивы `include_dir`, позволяющие подключать целые каталоги с файлами конфигурации. Они записываются в следующем формате: `include_dir 'каталог'`

Имена, заданные не абсолютным путём, рассматриваются относительно каталога, содержащего текущий файл конфигурации.

Включение файлов или каталогов позволяет разделить конфигурацию базы данных на логические части, а не вести один большой файл `postgresql.conf`.

3.2.5.1 Расположение файлов

По умолчанию файлы конфигурации размещаются в каталоге данных кластера БД. Параметры, описанные в этом разделе, позволяют разместить их и в любом другом месте. (Это позволяет упростить администрирование, в частности, выполнять резервное копирование этих файлов обычно проще, когда они хранятся отдельно.)

`data_directory (string)`

Задаёт каталог, в котором хранятся данные. Этот параметр можно задать только при запуске сервера.

`config_file (string)`

Задаёт основной файл конфигурации сервера (его стандартное имя - `postgresql.conf`). Этот параметр можно задать только в командной строке postgres.

`hba_file (string)`

Задаёт файл конфигурации для аутентификации по сетевым узлам (его стандартное имя – `pg_hba.conf`). Этот параметр можно задать только при старте сервера.

`ident_file (string)`

Задаёт файл конфигурации для сопоставлений имён пользователей (его стандартное имя - `pg_ident.conf`). Этот параметр можно задать только при запуске сервера.

`external_pid_file (string)`

Задаёт имя дополнительного файла с идентификатором процесса (PID), который будет создавать сервер для использования программами администрирования. Этот параметр можно задать только при запуске сервера.

При стандартной установке ни один из этих параметров не задаётся явно. Вместо них задаётся только каталог данных, аргументом командной строки –D или переменной окружения `PGDATA`, и все необходимые файлы конфигурации загружаются из этого каталога.

Чтобы разместить файлы конфигурации не в каталоге данных, то аргумент командной строки postgres –D или переменная окружения `PGDATA` должны указывать на каталог, содержащий файлы конфигурации, а в `postgresql.conf` (или в командной строке) должен задаваться параметр `data_directory`, указывающий, где фактически находятся данные.

Параметр `data_directory` переопределяет путь, задаваемый в –D или `PGDATA` как путь каталога данных, но не расположение файлов конфигурации.

Во всех этих параметрах относительный путь должен задаваться от каталога, в котором запускается postgres.

3.2.5.2 Параметры подключений

`listen_addresses (string)`

Задаёт адреса TCP/IP, по которым сервер будет принимать подключения клиентских приложений. Это значение принимает форму списка, разделённого запятыми, из имён и/или числовых IP-адресов компьютеров. Особый элемент, *, обозначает все имеющиеся IP-интерфейсы. Запись 0.0.0.0 позволяет задействовать все адреса IPv4, а :: — все адреса IPv6. Если список пуст, сервер не будет привязываться ни к какому IP-интерфейсу, а значит, подключиться к нему можно

будет только через Unix-сокеты. По умолчанию этот параметр содержит localhost, что допускает подключение к серверу по TCP/IP только через локальный интерфейс «замыкания». Параметр `listen_addresses` может ограничить интерфейсы, через которые будут приниматься соединения. Этот параметр можно задать только при запуске сервера.

port (integer)

TCP-порт, открываемый сервером; по умолчанию, 5432, он используется для всех IP-адресов, через которые сервер принимает подключения. Этот параметр можно задать только при запуске сервера.

max_connections (integer)

Определяет максимальное число одновременных подключений к серверу БД. По умолчанию обычно это 100 подключений, но это число может быть меньше, если ядро накладывает свои ограничения (это определяется в процессе `initdb`). Этот параметр можно задать только при запуске сервера.

Для ведомого сервера значение этого параметра должно быть больше или равно значению на ведущем. В противном случае на ведомом сервере не будут разрешены запросы.

3.2.5.3 Аутентификация

password_encryption (enum)

Когда в `CREATE ROLE` или `ALTER ROLE` задаётся пароль, этот параметр определяет, каким алгоритмом его шифровать. Возможные значения данного параметра — scram-sha-256 (пароль будет шифроваться алгоритмом SCRAM-SHA-256) и md5 (пароль сохраняется в виде хеша MD5). Значение по умолчанию — scram-sha-256

3.2.5.4 Память

shared_buffers (integer)

Задаёт объём памяти, который будет использовать сервер баз данных для буферов в разделяемой памяти. По умолчанию это обычно 128 мегабайт (128MB), но может быть и меньше, если конфигурация вашего ядра накладывает дополнительные ограничения (это определяется в процессе `initdb`). Это значение не должно быть меньше 128 килобайт. Однако для хорошей производительности обычно требуются гораздо большие значения. Если это значение задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Минимальное допустимое значение зависит от величины BLCKSZ. Задать этот параметр можно только при запуске сервера.

В системах с объёмом ОЗУ меньше 1 ГБ стоит ограничиться меньшим процентом ОЗУ, чтобы оставить достаточно места операционной системе.

work_mem (integer)

Задаёт базовый максимальный объём памяти, который будет использоваться во внутренних операциях при обработке запросов (например, для сортировки или хеш-таблиц), прежде чем будут задействованы временные файлы на диске. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение по умолчанию - четыре мегабайта (4MB).

maintenance_work_mem (integer)

Задаёт максимальный объём памяти для операций обслуживания БД, в частности VACUUM, CREATE INDEX и ALTER TABLE ADD FOREIGN KEY. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение по умолчанию - 64 мегабайта (64MB). Так как в один момент времени в сеансе может выполняться только одна такая операция и обычно они не запускаются параллельно, это значение вполне может быть гораздо больше work_mem. Увеличение этого значения может привести к ускорению операций очистки и восстановления БД из копии.

3.2.5.5 Асинхронное поведение

max_worker_processes (integer)

Задаёт максимальное число фоновых процессов, которое можно запустить в текущей системе. Этот параметр можно задать только при запуске сервера. Значение по умолчанию - 8.

Для ведомого сервера значение этого параметра должно быть больше или равно значению на ведущем. В противном случае на ведомом сервере не будут разрешены запросы. Одновременно с изменением этого значения также может быть полезно изменить `max_parallel_workers`.

max_parallel_workers (integer)

Задаёт максимальное число рабочих процессов, которое система сможет поддерживать для параллельных операций. Значение по умолчанию - 8. Значение данного параметра, превышающее `max_worker_processes`, не будет действовать, так как параллельные рабочие процессы берутся из пула рабочих процессов, ограничиваемого этим параметром.

3.2.5.6 Журнал предзаписи

wal_level (enum)

Задаёт, как много информации записывается в WAL. Со значением replica (по умолчанию) в журнал записываются данные, необходимые для поддержки архивирования WAL и репликации, включая запросы только на чтение на ведомом сервере. Вариант minimal оставляет только информацию, необходимую для восстановления после сбоя или аварийного отключения. Вариант logical добавляет информацию, требующуюся для поддержки логического декодирования. Каждый последующий уровень включает информацию, записываемую на всех уровнях ниже. Задать этот параметр можно только при запуске сервера.

fsync (boolean)

Если этот параметр установлен, сервер PostgreDB старается добиться, чтобы изменения были записаны на диск физически, выполняя системные вызовы fsync(). Это даёт гарантию, что кластер баз данных сможет вернуться в согласованное состояние после сбоя оборудования или операционной системы.

Хотя отключение fsync часто даёт выигрыш в скорости, это может привести к неисправимой порче данных в случае отключения питания или сбоя системы. В качестве примеров, когда отключение fsync неопасно, можно привести начальное наполнение нового кластера данными из копии, обработку массива данных, после которой базу данных можно удалить и создать заново, либо эксплуатацию копии базы данных только для чтения, которая регулярно пересоздаётся и не используется для отработки отказа. Качественное оборудование само по себе не является достаточной причиной для отключения fsync.

synchronous_commit (enum)

Определяет, после завершения какого уровня обработки WAL сервер будет сообщать об успешном выполнении операции. Допустимые значения: remote_apply (применено удалённо), on (вкл., по умолчанию), remote_write (записано удалённо), local (локально) и off (выкл.).

wal_sync_method (enum)

Метод, применяемый для принудительного сохранения изменений WAL на диске. Если режим fsync отключён, данный параметр не действует, так как принудительное сохранение изменений WAL не производится вовсе. Возможные значения этого параметра:

`open_datasync` (для сохранения файлов WAL открывать их функцией open() с параметром O_DSYNC)

`fdatasync` (вызывать fdatasync() при каждом фиксировании)

`fsync` (вызывать fsync() при каждом фиксировании)

`fsync_writethrough` (вызывать fsync() при каждом фиксировании, форсируя сквозную запись кеша)

`open_sync` (для сохранения файлов WAL открывать их функцией open() с параметром O_SYNC)

full_page_writes (boolean)

Когда этот параметр включён, сервер PostgreDB записывает в WAL всё содержимое каждой страницы при первом изменении этой страницы после контрольной точки. Это необходимо, потому что запись страницы, прерванная при сбое операционной системы, может выполниться частично, и на диске окажется страница, содержащая смесь старых данных с новыми. При этом информации об изменениях на уровне строк, которая обычно сохраняется в WAL, будет недостаточно для получения согласованного содержимого такой страницы при восстановлении после сбоя. Сохранение образа всей страницы гарантирует, что страницу можно восстановить корректно, ценой увеличения объёма данных, которые будут записываться в WAL.

Отключение этого параметра рекомендуется лишь в случаях, когда рекомендовалось отключение параметра fsync.

wal_log_hints (boolean)

Когда этот параметр имеет значение on, сервер PostgreDB записывает в WAL всё содержимое каждой страницы при первом изменении этой страницы после контрольной точки, даже при второстепенных изменениях так называемых вспомогательных битов. Этот параметр можно задать только при запуске сервера. По умолчанию он имеет значение off.

wal_compression (boolean)

Этот параметр позволяет без дополнительных рисков повреждения данных уменьшить объём WAL, ценой дополнительной нагрузки на процессор, связанной

со сжатием данных при записи в WAL и разворачиванием их при воспроизведении WAL.

wal_init_zero (boolean)

Если этот параметр включён (on), создаваемые файлы WAL заполняются нулями. Со значением off в создаваемый файл записывается только последний байт, чтобы файл WAL сразу обрёл желаемый размер.

wal_recycle (boolean)

Если этот параметр имеет значение on (по умолчанию), файлы WAL используются повторно (для этого они переименовываются), что избавляет от необходимости создавать новые файлы.

wal_buffers (integer)

Объём разделяемой памяти, который будет использоваться для буферизации данных WAL, ещё не записанных на диск. Значение по умолчанию, равное -1, задаёт размер, равный 1/32 (около 3%) от `shared_buffers`, но не меньше чем 64 КБ и не больше чем размер одного сегмента WAL (обычно 16 МБ). Содержимое буферов WAL записывается на диск при фиксировании каждой транзакции, так что очень большие значения вряд ли принесут значительную пользу. Однако значение как минимум в несколько мегабайт может увеличить быстродействие при записи на нагруженном сервере, когда сразу множество клиентов фиксируют транзакции.

wal_writer_delay (integer)

Определяет, с какой периодичностью процесс записи WAL будет сбрасывать WAL на диск. После очередного сброса WAL он делает паузу, длительность которой задаётся параметром `wal_writer_delay`, но может быть пробуждён асинхронно фиксируемой транзакцией.

wal_writer_flush_after (integer)

Определяет, при каком объёме процесс записи WAL будет сбрасывать WAL на диск.

wal_skip_threshold (integer)

Когда выбран wal_level minimal и фиксируется транзакция, которая создавала или перезаписывала постоянное отношение, этот параметр определяет, как будут сохраняться новые данные. Если объём данных меньше заданного значения, они будут записываться в журнал WAL; в противном случае затронутые файлы просто синхронизируются с ФС.

commit_delay (integer)

Параметр commit_delay добавляет паузу перед собственно выполнением сохранения WAL. Эта задержка может увеличить быстродействие при фиксировании множества транзакций, позволяя зафиксировать большее число транзакций за одну операцию сохранения WAL, если система нагружена достаточно сильно и за заданное время успевают зафиксироваться другие транзакции.

commit_siblings (integer)

Минимальное число одновременно открытых транзакций, при котором будет добавляться задержка commit_delay. Чем больше это значение, тем больше вероятность, что минимум одна транзакция окажется готовой к фиксированию за время задержки. По умолчанию это число равно пяти.

checkpoint_timeout (integer)

Максимальное время между автоматическими контрольными точками в WAL. Допускаются значения от 30 секунд до одного дня. Значение по умолчанию — пять минут (5min).

checkpoint_completion_target (floating point)

Задаёт целевое время для завершения процедуры контрольной точки, как долю общего времени между контрольными точками.

checkpoint_flush_after (integer)

Когда в процессе контрольной точки записывается больше заданного объёма данных, сервер даёт указание ОС произвести запись этих данных в нижележащее хранилище. Это ограничивает объём «грязных» данных в страничном кеше ядра и уменьшает вероятность затормаживания при выполнении fsync в конце контрольной точки или когда ОС сбрасывает данные на диск большими порциями в фоне. Часто это значительно уменьшает задержки транзакций, но бывают ситуации (особенно когда объём рабочей нагрузки больше `shared_buffers`, но меньше страничного кеша ОС), когда производительность может упасть. Значение по умолчанию — 256kB.

checkpoint_warning (integer)

Записывать в журнал сервера сообщение в случае, если контрольные точки, вызванные заполнением файлов сегментов WAL, выполняются быстрее, чем через заданное время (что говорит о том, что нужно увеличить max_wal_size). Значение по умолчанию равно 30 секундам (30s). При нуле это предупреждение отключается.

max_wal_size (integer)

Максимальный размер, до которого может вырастать WAL во время автоматических контрольных точек. Значение по умолчанию — 1 ГБ.

min_wal_size (integer)

Пока WAL занимает на диске меньше этого объёма, старые файлы WAL в контрольных точках всегда перерабатываются, а не удаляются. Это позволяет

зарезервировать достаточно места для WAL, чтобы справиться с резкими скачками использования WAL, например, при выполнении больших пакетных заданий. Значение по умолчанию — 80 МБ.

archive_mode (enum)

Когда параметр archive_mode включён, полные сегменты WAL передаются в хранилище архива командой archive_command. Помимо значения off (выключающего архивацию) есть ещё два: on (вкл.) и always (всегда). В обычном состоянии эти два режима не различаются, но в режиме always архивация WAL активна и во время восстановления архива, и при использовании ведомого сервера. В этом режиме все файлы, восстановленные из архива или полученные при потоковой репликации, будут архивироваться (снова).

archive_command (string)

Команда, которая будет выполняться для архивации завершённого сегмента WAL. Любое вхождение %p в этой строке заменяется путём архивируемого файла, а вхождение %f заменяется только его именем. Команда должна возвращать нулевой код, если она завершается успешно.

archive_timeout (integer)

Команда archive_command вызывается только для завершённых сегментов WAL. Для ограничения времени существования неархивированных данных можно установить значение archive_timeout, чтобы сервер периодически переключался на новый файл сегмента WAL. Заметьте, что архивируемые файлы, закрываемые досрочно из-за принудительного переключения, всё равно будут иметь тот же размер, что и полностью заполненные.

restore_command (string)

Команда, которая выполняется для извлечения архивного сегмента из набора файлов WAL. Этот параметр требуется для восстановления из архива, но необязателен для потоковой репликации. Любое вхождение %f в строке заменяется именем извлекаемого из архива файла, а %p заменяется на путь назначения на сервере. Любое вхождение %r заменяется на имя файла, в котором содержится последняя действительная точка восстановления. Команда должна возвращать ноль на выходе лишь в случае успешного выполнения. Команде будут поступать имена файлов, отсутствующих в архиве; в этом случае она должна возвращать ненулевой статус.

archive_cleanup_command (string)

Команда, которая будет вызываться при каждой точке перезапуска, для предоставления механизма очистки от старых архивных файлов WAL, которые более не нужны на ведомом сервере. Любое вхождение %r заменяется на имя файла, содержащего последнюю действительную точку перезапуска. Это самый ранний файл, который необходимо хранить для возможности восстановления, а более старые файлы вполне можно удалить. Эта информация может быть использована для усечения архива с целью его минимизации при сохранении возможности последующего восстановления из заданной точки. Стоит обратить внимание, что в конфигурациях с множеством ведомых серверов, использующих общий архивный каталог для восстановления, необходимо контролировать удаление файлов WAL, так как они могут ещё быть нужны некоторым серверам. Поэтому archive_cleanup_command обычно используется при организации тёплого резерва.

recovery_end_command (string)

Команда, которая будет выполнена единожды в конце процесса восстановления. Назначение параметра recovery_end_command — предоставить механизм для очистки после репликации или восстановления. Любое вхождение %r заменяется

именем файла, содержащим последнюю действительную точку восстановления, например, как в archive_cleanup_command.

recovery_target_name (string)

Параметр указывает именованную точку восстановления, до которой будет производиться восстановление.

recovery_target_time (timestamp)

Параметр указывает точку времени, вплоть до которой будет производиться восстановление. Окончательно точка останова определяется в зависимости от значения recovery_target_inclusive.

recovery_target_xid (string)

Параметр указывает идентификатор транзакции, вплоть до которой необходимо произвести процедуру восстановления. Имейте в виду, что числовое значение идентификатора отражает последовательность именно старта транзакций, а фиксироваться они могут в ином порядке. Восстановлению будут подлежать все транзакции, что были зафиксированы до указанной (и, возможно, включая её). Точность точки останова также зависит от recovery_target_inclusive.

recovery_target_lsn (pg_lsn)

Данный параметр указывает LSN позиции в журнале предзаписи, до которой должно выполняться восстановление. Точная позиция остановки зависит также от параметра recovery_target_inclusive.

recovery_target_inclusive (boolean)

Указывает на необходимость остановки сразу после (on) либо до (off) достижения целевой точки. Применяется одновременно с recovery_target_lsn, recovery_target_time или recovery_target_xid. По умолчанию выбирается вариант on.

recovery_target_timeline (string)

Указывает линию времени для восстановления. Значение может задаваться числовым идентификатором линии времени или ключевым словом. С ключевым словом current восстанавливается та линия времени, которая была активной при создании базовой резервной копии. С ключевым словом latest восстанавливаться будет последняя линия времени, найденная в архиве, что полезно для ведомого сервера. По умолчанию подразумевается latest.

recovery_target_action (enum)

Указывает, какое действие должен предпринять сервер после достижения цели восстановления. Вариант по умолчанию — pause, что означает приостановку восстановления. Второй вариант, promote, означает, что процесс восстановления завершится, и сервер начнёт принимать подключения. Наконец, с вариантом shutdown сервер остановится, как только цель восстановления будет достигнута.

3.2.6 Репликация

Параметры, перечисленные в данном разделе, управляют поведением встроенного механизма потоковой репликации. Когда он применяется, один сервер является ведущим, а другие — ведомыми. Ведущий сервер всегда передаёт, а ведомые всегда принимают данные репликации, но, когда настроена каскадная репликация, ведомые серверы могут быть и передающими.

Передающие серверы

Следующие параметры можно задать на любом сервере, который передаёт данные репликации одному или нескольким ведомым. Ведущий сервер всегда является передающим, так что на нём они должны задаваться всегда. Роль и значение этих параметров не меняются после того, как ведомый сервер становится ведущим.

max_wal_senders (integer)

Задаёт максимально допустимое число одновременных подключений ведомых серверов или клиентов потокового копирования (т. е. максимальное количество одновременно работающих процессов передачи WAL). Значение по умолчанию — 10. При значении 0 репликация отключается. В случае неожиданного отключения клиента потоковой передачи слот подключения может оставаться занятым до достижения тайм-аута, так что этот параметр должен быть немного больше максимально допустимого числа клиентов, чтобы отключившиеся клиенты могли переподключиться немедленно. Задать этот параметр можно только при запуске сервера. Чтобы к данному серверу могли подключаться ведомые, уровень `wal_level` должен быть replica или выше. Для ведомого сервера значение этого параметра должно быть больше или равно значению на ведущем. В противном случае на ведомом сервере не будут разрешены запросы.

max_replication_slots (integer)

Задаёт максимальное число слотов репликации, которое сможет поддерживать сервер. Значение по умолчанию — 10. Этот параметр можно задать только при запуске сервера. Если заданное значение данного параметра будет меньше, чем число уже существующих слотов репликации, сервер не запустится. Чтобы слоты репликации можно было использовать, нужно также установить в `wal_level` уровень replica или выше.

wal_keep_size (integer)

Задаёт минимальный объём прошлых сегментов журнала, который будет сохраняться в каталоге `pg_wal`, чтобы ведомый сервер мог выбрать их при потоковой репликации. Если ведомый сервер, подключённый к передающему, отстаёт больше чем на `wal_keep_size` мегабайт, передающий может удалить сегменты WAL, всё ещё необходимые ведомому, и в этом случае соединение репликации прервётся. В результате этого затем также будут прерваны зависимые соединения. (Однако ведомый сервер сможет восстановиться, выбрав этот сегмент из архива, если осуществляется архивация WAL.)

max_slot_wal_keep_size (integer)

Задаёт максимальный размер файлов WAL, который может оставаться в каталоге `pg_wal` для слотов репликации после выполнения контрольной точки.

wal_sender_timeout (integer)

Задаёт период времени, по истечении которого прерываются неактивные соединения репликации. Это помогает передающему серверу обнаружить сбой ведомого или разрывы сети. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию — 60 секунд. При значении, равном нулю, тайм-аут отключается.

track_commit_timestamp (boolean)

Включает запись времени фиксации транзакций. По умолчанию этот параметр имеет значение off.

Главный сервер

Следующие параметры можно задать на главном/ведущем сервере, который должен передавать данные репликации одному или нескольким ведомым. Значения этих параметров на ведомых серверах не важны, хотя их можно подготовить заранее, на случай, если ведомый сервер придётся сделать ведущим.

synchronous_standby_names (string)

Определяет список ведомых серверов, которые могут поддерживать синхронную репликацию. Активных синхронных ведомых серверов может быть один или несколько; транзакции, ожидающие фиксации, будут завершаться только после того, как эти ведомые подтвердят получение их данных. Синхронными ведомыми будут те, имена которых указаны в этом списке и которые подключены к ведущему и принимают поток данных в реальном времени. Указание нескольких имён

ведомых серверов позволяет обеспечить очень высокую степень доступности и защиту от потери данных. Если имена синхронных ведомых серверов не определены, синхронная репликация не включается и фиксируемые транзакции не будут ждать репликации. Это поведение по умолчанию. Даже когда синхронная репликация включена, для отдельных транзакций можно отключить ожидание репликации, задав для параметра `synchronous_commit` значение local или off.

`vacuum_defer_cleanup_age (integer)`

Задаёт число транзакций, на которое будет отложена очистка старых версий строк при `VACUUM` и изменениях `HOT`. По умолчанию это число равно нулю, то есть старые версии строк могут удаляться сразу, как только перестанут быть видимыми в открытых транзакциях.

Ведомые серверы

Следующие параметры управляют поведением ведомого сервера, который будет получать данные репликации.

`primary_conninfo (string)`

Указывает строку подключения резервного сервера к передающему. В строке подключения должно задаваться имя (или адрес) передающего сервера, а также номер порта, если он отличается от подразумеваемого по умолчанию ведущим. Также в ней указывается имя пользователя, соответствующее роли с необходимыми правами на передающем сервере. Если сервер осуществляет аутентификацию по паролю, дополнительно потребуется задать пароль.

`primary_slot_name (string)`

Дополнительно задаёт заранее созданный слот, который будет использоваться при подключении к передающему серверу по протоколу потоковой репликации для управления освобождением ресурсов вышестоящего узла.

promote_trigger_file (string)

Указывает триггерный файл, при появлении которого завершается работа в режиме ведомого.

hot_standby (boolean)

Определяет, можно ли будет подключаться к серверу и выполнять запросы в процессе восстановления. Значение по умолчанию — on (подключения разрешаются).

max_standby_archive_delay (integer)

В режиме горячего резерва этот параметр определяет, как долго должен ждать ведомый сервер, прежде чем отменять запросы, конфликтующие с очередными изменениями в WAL. Применяется при обработке данных WAL, считываемых из архива.

max_standby_streaming_delay (integer)

В режиме горячего резерва этот параметр определяет, как долго должен ждать ведомый сервер, прежде чем отменять запросы, конфликтующие с очередными изменениями в WAL. Применяется при обработке данных WAL, поступающих при потоковой репликации.

wal_receiver_create_temp_slot (boolean)

Определяет, должен ли процесс-приёмник WAL создавать временный слот репликации на удалённом сервере в случаях, когда постоянный слот репликации не настроен. По умолчанию отключён.

wal_receiver_status_interval (integer)

Определяет минимальную частоту, с которой процесс, принимающий WAL на ведомом сервере, будет сообщать о состоянии репликации ведущему или вышестоящему ведомому. Значение по умолчанию равно 10 секундам.

hot_standby_feedback (boolean)

Определяет, будет ли сервер горячего резерва сообщать ведущему или вышестоящему ведомому о запросах, которые он выполняет в данный момент. Если используется каскадная репликация, сообщения о запросах передаются выше, пока в итоге не достигнут ведущего сервера. На промежуточных серверах эта информация больше никак не задействуется.

wal_receiver_timeout (integer)

Задаёт период времени, по истечении которого прерываются неактивные соединения репликации. Это помогает принимающему ведомому серверу обнаружить сбой ведущего или разрыв сети. Значение по умолчанию — 60 секунд.

wal_retrieve_retry_interval (integer)

Определяет, сколько ведомый сервер должен ждать поступления данных WAL из любых источников (потоковая репликация, локальный `pg_wal` или архив WAL), прежде чем повторять попытку получения WAL. Значение по умолчанию — 5 секунд.

recovery_min_apply_delay (integer)

По умолчанию ведомый сервер восстанавливает записи WAL передающего настолько быстро, насколько это возможно. Иногда полезно иметь возможность задать задержку при копировании данных, например, для устранения ошибок, связанных с потерей данных. Этот параметр позволяет отложить восстановление на заданное время. Например, если установить значение 5min, ведомый сервер будет воспроизводить фиксацию транзакции не раньше, чем через 5 минут (судя по его

системным часам) после времени фиксации, сообщённого ведущим. Значение по умолчанию равно нулю, то есть задержка не добавляется.

Логическая репликация

Следующие параметры управляют поведением подписчика логической репликации.

max_logical_replication_workers (int)

Задаёт максимально возможное число рабочих процессов логической репликации. В это число входят и рабочие процессы, применяющие изменения, и процессы, синхронизирующие таблицы. Значение по умолчанию — 4.

max_sync_workers_per_subscription (integer)

Максимальное число рабочих процессов, выполняющих синхронизацию, для одной подписки. Этот параметр управляет степенью распараллеливания копирования начальных данных в процессе инициализации подписки или при добавлении новых таблиц. Значение по умолчанию — 2.

3.2.7 Планирование запросов

Выполняя любой полученный запрос, PostgreDB разрабатывает для него план запроса. Выбор правильного плана, соответствующего структуре запроса и характеристикам данным, крайне важен для хорошей производительности, поэтому в системе работает сложный планировщик, задача которого — подобрать хороший план. Узнать, какой план был выбран для какого-либо запроса, можно с помощью команды `EXPLAIN`.

Структура плана запроса представляет собой дерево узлов плана. Узлы на нижнем уровне дерева — это узлы сканирования, которые возвращают необработанные данные таблицы. Разным типам доступа к таблице соответствуют разные узлы: последовательное сканирование, сканирование индекса и сканирование битовой карты. Источниками строк могут быть не только таблицы, но и, например, предложения `VALUES` и функции, возвращающие множества во `FROM`, и они представляются отдельными типами узлов

сканирования. Если запрос требует объединения, агрегатных вычислений, сортировки или других операций с исходными строками, над узлами сканирования появляются узлы, обозначающие эти операции. И так как обычно операции могут выполняться разными способами, на этом уровне тоже могут быть узлы разных типов. В выводе команды `EXPLAIN` для каждого узла в дереве плана отводится одна строка, где показывается базовый тип узла плюс оценка стоимости выполнения данного узла, которую сделал для него планировщик. Если для узла выводятся дополнительные свойства, в вывод могут добавляться дополнительные строки, с отступом от основной информации узла. В самой первой строке (основной строке самого верхнего узла) выводится общая стоимость выполнения для всего плана; именно это значение планировщик старается минимизировать.

Точность оценок планировщика можно проверить, используя команду `EXPLAIN` с параметром `ANALYZE`. С этим параметром `EXPLAIN` на самом деле выполняет запрос, а затем выводит фактическое число строк и время выполнения, накопленное в каждом узле плана, вместе с теми же оценками, что выдаёт обычная команда `EXPLAIN`.

В некоторых планах запросов некоторый внутренний узел может выполняться неоднократно. Например, внутреннее сканирование индекса будет выполняться для каждой внешней строки во вложенном цикле верхнего уровня. В таких случаях значение loops (циклы) показывает, сколько всего раз выполнялся этот узел, а фактическое время и число строк вычисляется как среднее по всем итерациям. Это делается для того, чтобы полученные значения можно было сравнить с выводимыми приблизительными оценками. Чтобы получить общее время, затраченное на выполнение узла, время одной итерации нужно умножить на значение `loops`.

Под заголовком Planning time (Время планирования) команда `EXPLAIN ANALYZE` выводит время, затраченное на построение плана запроса из разобранного запроса и его оптимизацию. Время собственно разбора или перезаписи запроса в него не включается.

Значение Execution time (Время выполнения), выводимое командой `EXPLAIN ANALYZE`, включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров, но не включает время разбора, перезаписи и планирования запроса. Время, потраченное на выполнение триггеров BEFORE (если такие

имеются) включается во время соответствующих узлов `Insert, Update или Delete node`; но время выполнения триггеров `AFTER` не учитывается, так как триггеры `AFTER` срабатывают после выполнения всего плана. Общее время, проведённое в каждом триггере (`BEFORE` или `AFTER`), также выводится отдельно.

Время выполнения, измеренное командой `EXPLAIN ANALYZE`, может значительно отличаться от времени выполнения того же запроса в обычном режиме. Тому есть две основных причины. Во-первых, так как при анализе никакие строки результата не передаются клиенту, время ввода/вывода и передачи по сети не учитывается. Во-вторых, может быть существенной дополнительная нагрузка, связанная с функциями измерений `EXPLAIN ANALYZE`, особенно в системах, где вызов `gettimeofday()` выполняется медленно.

3.2.8 Регистрация ошибок и протоколирование работы сервера

Результаты `EXPLAIN` не следует распространять на ситуации, значительно отличающиеся от тех, в которых проводится тестирование. В частности, не следует полагать, что выводы, полученные для игрушечной таблицы, будут применимы и для настоящих больших таблиц. Оценки стоимости нелинейны и планировщик может выбирать разные планы в зависимости от размера таблицы. Регистрация ошибок и протоколирование работы сервера

3.2.8.1 Куда протоколировать

`log_destination (string)`

PostgreDB поддерживает несколько методов протоколирования сообщений сервера: `stderr, csvlog и syslog`. В качестве значения `log_destination` указывается один или несколько методов протоколирования, разделённых запятыми. По умолчанию используется `stderr`. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера.

Если в `log_destination` включено значение `csvlog`, то протоколирование ведётся в формате CSV (разделённые запятыми значения). Это удобно для программной обработки журнала.

Если присутствует указание `stderr` или `csvlog`, создаётся файл `current_logfiles`, в который записывается расположение файла(ов) журнала, в настоящее время используемого сборщиком сообщений для соответствующего назначения. Это позволяет легко определить, какие файлы журнала используются в данный момент экземпляром сервера.

`logging_collector (boolean)`

Параметр включает сборщик сообщений (*logging collector*). Это фоновый процесс, который собирает отправленные в `stderr` сообщения и перенаправляет их в журнальные файлы. Такой подход зачастую более полезен чем запись в `syslog`, поскольку некоторые сообщения в `syslog` могут не попасть. Для установки параметра требуется перезапуск сервера.

`log_directory (string)`

При включённом `logging_collector`, определяет каталог, в котором создаются журнальные файлы. Можно задавать как абсолютный путь, так и относительный от каталога данных кластера. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера. Значение по умолчанию - `log`.

`log_filename (string)`

При включённом `logging_collector` задаёт имена журнальных файлов. Значение трактуется как строка формата в функции `strftime`, поэтому в ней можно использовать спецификаторы % для включения в имена файлов информации о дате и времени. (При наличии зависящих от часового пояса спецификаторов % будет использован пояс, заданный в `log_timezone`.). Значение по умолчанию `postgresql-%Y-%m-%d_%H%M%S.log`.

Если для задания имени файлов не используются спецификаторы %, то для избежания переполнения диска, следует использовать утилиты для ротации журнальных файлов.

Если в `log_destination` включён вывод в формате CSV, то к имени журнального файла будет добавлено расширение `.csv`. (Если `log_filename` заканчивается на `.log`, то это расширение заменится на `.csv`.)

Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

`log_file_mode (integer)`

Параметр задаёт права доступа к журнальным файлам, при включённом `logging_collector`. Значение параметра должно быть числовым, в формате команд `chmod` и `umask`. (Для восьмеричного формата, требуется задать лидирующий 0 (ноль).)

Права доступа по умолчанию 0600, т. е. только владелец сервера может читать и писать в журнальные файлы. Также, может быть полезным значение 0640, разрешающее чтение файлов членам группы. Однако, чтобы установить такое значение, нужно каталог для хранения журнальных файлов (`log_directory`) вынести за пределы каталога данных кластера. В любом случае нежелательно открывать для всех доступ на чтение журнальных файлов, так как они могут содержать конфиденциальные данные.

Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

`log_rotation_age (integer)`

При включённом `logging_collector` этот параметр определяет максимальное время жизни отдельного журнального файла, по истечении которого создаётся новый файл. Если это значение задаётся без единиц измерения, оно считается заданным в минутах. Значение по умолчанию 24 часа. При нулевом значении смена

файлов по времени не производится. Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

log_rotation_size (integer)

При включённом `logging_collector` этот параметр определяет максимальный размер отдельного журнального файла. При достижении этого размера создаётся новый файл. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение по умолчанию - 10 мегабайт. При нулевом значении смена файлов по размеру не производится. Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

log_truncate_on_rotation (boolean)

Если параметр `logging_collector` включён, PostgreDB будет перезаписывать существующие журнальные файлы, а не дописывать в них. Однако перезапись при переключении на новый файл возможна только в результате ротации по времени, но не при старте сервера или ротации по размеру файла. При выключенном параметре всегда продолжается запись в существующий файл. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера.

syslog_facility (enum)

При включённом протоколировании в `syslog`, этот параметр определяет значение «`facility`». Допустимые значения LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7. По умолчанию используется LOCAL0. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера.

syslog_ident (string)

При включённом протоколировании в syslog, этот параметр задаёт имя программы, которое будет использоваться в syslog для идентификации сообщений, относящихся к PostgreDB. По умолчанию используется postgres. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

syslog_sequence_numbers (boolean)

Когда сообщения выводятся в syslog и этот параметр включён (по умолчанию), все сообщения будут предваряться последовательно увеличивающимися номерами. Это позволяет обойти подавление повторов «--- последнее сообщение повторилось N раз ---», которое по умолчанию осуществляется во многих реализациях syslog. В более современных реализациях syslog подавление повторных сообщений можно настроить (например, в rsyslog есть директива $RepeatedMsgReduction), так что это может излишне. Если же вы действительно хотите, чтобы повторные сообщения подавлялись, вы можете отключить этот параметр.

Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

syslog_split_messages (boolean)

Когда активен вывод сообщений в syslog, этот параметр определяет, как будут доставляться сообщения. Если он включён (по умолчанию), сообщения разделяются по строкам, а длинные строки разбиваются на строки не длиннее 1024 байт, что составляет типичное ограничение размера для традиционных реализаций syslog. Когда он отключён, сообщения сервера PostgreDB передаются службе syslog как есть, и она должна сама корректно воспринять потенциально длинные сообщения.

Если syslog в итоге выводит сообщения в текстовый файл, результат будет тем же и лучше оставить этот параметр включённым, так как многие реализации

syslog не способны обрабатывать большие сообщения или их нужно специально настраивать для этого. Но если syslog направляет сообщения в некоторую другую среду, может потребоваться или будет удобнее сохранять логическую целостность сообщений.

Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

event_source (string)

При включённом протоколировании в event log, этот параметр задаёт имя программы, которое будет использоваться в журнале событий для идентификации сообщений относящихся к PostgreDB. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера

3.2.8.2 Когда протоколировать

log_min_messages (enum)

Управляет минимальным уровнем сообщений, записываемых в журнал сервера. Допустимые значения DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL и PANIC. Каждый из перечисленных уровней включает все идущие после него. Чем дальше в этом списке уровень сообщения, тем меньше сообщений будет записано в журнал сервера. По умолчанию используется WARNING. Позиция LOG здесь отличается от принятой в client_min_messages. Только суперпользователи могут изменить этот параметр.

log_min_error_statement (enum)

Управляет тем, какие SQL-операторы, завершившиеся ошибкой, записываются в журнал сервера. SQL-оператор будет записан в журнал, если он завершится ошибкой с указанным уровнем важности или выше. Допустимые значения: DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL и PANIC. По умолчанию используется

ERROR. Это означает, что в журнал сервера будут записаны все операторы, завершившиеся сообщением с уровнем важности ERROR, LOG, FATAL и PANIC. Чтобы фактически отключить запись операторов с ошибками, установите для этого параметра значение PANIC. Изменить этот параметр могут только суперпользователи.

3.2.8.3 Что протоколировать

application_name (string)

application_name это любая строка, не превышающая NAMEDATALEN символов (64 символа при стандартной сборке). Обычно устанавливается приложением при подключении к серверу. Значение отображается в представлении pg_stat_activity и добавляется в журнал сервера, при использовании формата CSV. Для прочих форматов, application_name можно добавить в журнал через параметр log_line_prefix. Значение application_name может содержать только печатные ASCII символы. Остальные символы будут заменены знаками вопроса (?).

debug_print_parse (boolean)
debug_print_rewritten (boolean)
debug_print_plan (boolean)

Эти параметры включают вывод различной отладочной информации. А именно: вывод дерева запроса, дерево запроса после применения правил или плана выполнения запроса, соответственно. Все эти сообщения имеют уровень LOG. Поэтому, по умолчанию, они записываются в журнал сервера, но не отправляются клиенту. Отправку клиенту можно настроить через client_min_messages и/или log_min_messages. По умолчанию параметры выключены.

debug_pretty_print (boolean)

Включает выравнивание сообщений, выводимых debug_print_parse, debug_print_rewritten или debug_print_plan. В результате сообщения

легче читать, но они значительно длиннее, чем в формате «compact», который используется при выключенном значении. По умолчанию включён.

log_autovacuum_min_duration (integer)

Задаёт время выполнения действия автоочистки, при превышении которого информация об этом действии записывается в журнал. При нулевом значении в журнале фиксируются все действия автоочистки. Значение -1 (по умолчанию) отключает журналирование действий автоочистки. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Когда этот параметр имеет любое значение, отличное от-1, в журнал будет записываться сообщение в случае пропуска действия автоочистки из-за конфликтующей блокировки или параллельного удаления отношения. Таким образом, включение этого параметра позволяет отслеживать активность автоочистки. Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера. Однако его можно переопределить для отдельных таблиц, изменив их параметры хранения.

log_checkpoints (boolean)

Включает протоколирование выполнения контрольных точек и точек перезапуска сервера. При этом записывается некоторая статистическая информация. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера. По умолчанию выключен.

log_connections (boolean)

Включает протоколирование всех попыток подключения к серверу, в том числе успешного завершения как аутентификации (если она требуется), так и авторизации клиентов. Изменить его можно только в начале сеанса и сделать это могут только суперпользователи. Значение по умолчанию - off.

log_disconnections (boolean)

Включает протоколирование завершения сеанса. В журнал выводится примерно та же информация, что и с `log_connections`, плюс длительность сеанса. Изменить этот параметр можно только в начале сеанса и сделать это могут только суперпользователи. Значение по умолчанию - off.

log_duration (boolean)

Записывает продолжительность каждой завершённой команды. По умолчанию выключен. Только суперпользователи могут изменить этот параметр.

Для клиентов, использующих расширенный протокол запросов, будет записываться продолжительность фаз: разбор, связывание и выполнение.

log_error_verbosity (enum)

Управляет количеством детальной информации, записываемой в журнал сервера для каждого сообщения. Допустимые значения: `TERSE`, `DEFAULT` и `VERBOSE`. Каждое последующее значение добавляет больше полей в выводимое сообщение. Для `TERSE` из сообщения об ошибке исключаются поля `DETAIL`, `HINT`, `QUERY` и `CONTEXT`. Для `VERBOSE` в сообщение включается код ошибки `SQLSTATE`, а также имя файла с исходным кодом, имя функции и номер строки, сгенерировавшей ошибку. Только суперпользователи могут изменить этот параметр.

log_hostname (boolean)

По умолчанию, сообщения журнала содержат лишь IP-адрес подключившегося клиента. При включении этого параметра, дополнительно будет фиксироваться и имя сервера. Обратите внимание, что в зависимости от применяемого способа разрешения имён, это может отрицательно сказаться на производительности. Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

log_line_prefix (string)

Строка, в стиле функции printf, которая выводится в начале каждой строки журнала сообщений. С символов % начинаются управляющие последовательности, которые заменяются статусной информацией, описанной ниже. Неизвестные управляющие последовательности игнорируются. Все остальные символы напрямую копируются в выводимую строку. Некоторые управляющие последовательности используются только для пользовательских процессов и будут игнорироваться фоновыми процессами, например, основным процессом сервера. Статусная информация может быть выровнена по ширине влево или вправо указанием числа после % и перед кодом последовательности. Отрицательное число дополняет значение пробелами справа до заданной ширины, а положительное число - слева. Выравнивание может быть полезно для улучшения читаемости.

Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. По умолчанию этот параметр имеет значение %m [%p] ', с которым в журнал выводится время и идентификатор процесса (PID).

Спецсимвол	Назначение	Только для пользовательского процесса
%a	Имя приложения (application_name)	да
%u	Имя пользователя	да
%d	Имя базы данных	да
%r	Имя удалённого узла или IP-адрес, а также номер порта	да
%h	Имя удалённого узла или IP-адрес	да
%b	Тип обслуживающего процесса	нет
%p	Идентификатор процесса	нет
%P	Идентификатор ведущего процесса группы, если текущий процесс является исполнителем параллельного запроса	нет

Спецсимвол	Назначение	Только для пользовательского процесса
%t	Штамп времени, без миллисекунд	нет
%m	Штамп времени, с миллисекундами	нет
%n	Штамп времени, с миллисекундами (в виде времени Unix)	нет
%i	Тег команды: тип текущей команды в сессии	да
%e	Код ошибки SQLSTATE	нет
%c	Идентификатор сессии. Подробности ниже	нет
%l	Номер строки журнала для каждой сессии или процесса. Начинается с 1	нет
%s	Штамп времени начала процесса	нет
%v	Идентификатор виртуальной транзакции (backendID/localXID)	нет
%x	Идентификатор транзакции (0 если не присвоен)	нет
%q	Ничего не выводит. Непользовательские процессы останавливаются в этой точке. Игнорируется пользовательскими процессами	нет
%Q	Идентификатор текущего запроса. Вычисление идентификаторов запроса по умолчанию отключено, поэтому в этом поле будет выводиться значение 0, если не включён параметр или не настроен дополнительный модуль, вычисляющий идентификаторы запросов.	да
%%	Выводит %	нет

Тип обслуживающего процесса соответствует содержимому столбца `backend_type`, но в журнале могут фигурировать и другие типы, которые не показываются в этом представлении. Спецпоследовательность %c заменяется на

практически уникальный идентификатор сеанса, содержащий из 4 шестнадцатеричных чисел (без ведущих нулей), разделённых точками. Эти числа представляют время запуска процесса и его PID, так что %c можно использовать как более компактную альтернативу совокупности этих двух элементов.

log_lock_waits (boolean)

Определяет, нужно ли фиксировать в журнале события, когда сеанс ожидает получения блокировки дольше, чем указано в `deadlock_timeout`. Это позволяет выяснить, не связана ли низкая производительность с ожиданием блокировок. По умолчанию отключено. Только суперпользователи могут изменить этот параметр.

log_recovery_conflict_waits (boolean)

Определяет, нужно ли фиксировать в журнале события, когда сеанс ожидает получения блокировки дольше, чем указано в `deadlock_timeout` для конфликтов восстановления. Это позволяет выяснить, препятствуют ли конфликты восстановления применению WAL. Значение по умолчанию - off (выкл.). Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

log_parameter_max_length (integer)

Положительное значение устанавливает количество байт, до которого будут усекаться значения привязанных SQL-параметров, выводимые в сообщениях вместе с SQL-операторами (это не относится к регистрации ошибок). Ноль отключает вывод значений привязанных параметров в таких сообщениях. Значение -1 (по умолчанию) позволяет получить в журнале привязанные параметры в полном объёме. Если значение этого параметра задаётся без единиц измерения, оно считается заданным в байтах. Изменить этот параметр могут только суперпользователи. Этот параметр влияет только на сообщения, выводимые в журнал по критериям, которые устанавливают параметры `log_statement`, `log_duration` и связанные с ними.

log_parameter_max_length_on_error (integer)

Положительное значение устанавливает количество байт, до которого будут усекаться значения привязанных SQL-параметров, выводимые вместе с SQL-операторами при регистрации ошибок. Нулевое значение отключает вывод значений привязанных операторов в сообщениях об ошибках. Значение -1 (по умолчанию) позволяет получить в журнале привязанные параметры в полном объёме. Если значение этого параметра задаётся без единиц измерения, оно считается заданным в байтах.

log_statement (enum)

Управляет тем, какие SQL-команды записывать в журнал. Допустимые значения: `none` (отключено), `ddl`, `mod` и `all` (все команды). `Ddl` записывает все команды определения данных, такие как `CREATE`, `ALTER`, `DROP`. `Mod` описывает все команды `ddl`, а также команды изменяющие данные, такие как `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE` и `COPY FROM PREPARE`, `EXECUTE` и `EXPLAIN ANALYZE` также записываются, если вызваны для команды соответствующего типа. Если клиент использует расширенный протокол запросов, то запись происходит на фазе выполнения и содержит значения всех связанных переменных (если есть символы одиночных кавычек, то они дублируются).

По умолчанию `none`. Только суперпользователи могут изменить этот параметр.

log_replication_commands (boolean)

Включает запись в журнал сервера всех команд репликации.

Значение по умолчанию - off. Изменить этот параметр могут только суперпользователи.

log_temp_files (integer)

Управляет регистрацией в журнале имён и размеров временных файлов. Временные файлы могут использоваться для сортировки, хеширования и временного хранения результатов запросов. Когда этот параметр включён, при удалении временного файла информация о нём может записываться в журнал. При нулевом значении записывается информация обо всех файлах, а при положительном - о файлах, размер которых не меньше заданной величины. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение по умолчанию равно -1, то есть запись такой информации отключена. Изменить этот параметр могут только суперпользователи.

log_timezone (string)

Устанавливает часовой пояс для штампов времени при записи в журнал сервера. В отличие от TimeZone, это значение одинаково для всех баз данных кластера, поэтому для всех сессий используются согласованные значения штампов времени. Встроенное значение по умолчанию GMT, но оно переопределяется в `postgresql.conf`: `initdb` записывает в него значение, соответствующее системной среде. Задать этот параметр можно только в `postgresql.conf` или в командной строке при запуске сервера.

3.2.9 Параметры клиентских сеансов по умолчанию

3.2.9.1 Поведение команд

client_min_messages (enum)

Управляет минимальным уровнем сообщений, посылаемых клиенту. Допустимые значения `DEBUG5`, `DEBUG4`, `DEBUG3`, `DEBUG2`, `DEBUG1`, `LOG`, `NOTICE`, `WARNING` и `ERROR`. Каждый из перечисленных уровней включает все идущие после него. Чем дальше в этом списке уровень сообщения, тем меньше сообщений будет посылаться клиенту. По умолчанию используется `NOTICE`. Обратите внимание, позиция `LOG` здесь отличается от принятой в `log_min_messages`.

Сообщения уровня `INFO` передаются клиенту всегда.

search_path (string)

Эта переменная определяет порядок, в котором будут просматриваться схемы при поиске объекта (таблицы, типа данных, функции и т. д.), к которому обращаются просто по имени, без указания схемы. Если объекты с одинаковым именем находятся в нескольких схемах, использоваться будет тот, что встретится первым при просмотре пути поиска. К объекту, который не относится к схемам, перечисленным в пути поиска, можно обратиться только по полному имени (с точкой), с указанием содержащей его схемы.

Значением `search_path` должен быть список имён схем через запятую. Если для имени, указанного в этом списке, не находится существующая схема, либо пользователь не имеет права `USAGE` для схемы с этим именем, такое имя просто игнорируется.

Если список содержит специальный элемент `$user`, вместо него подставляется схема с именем, возвращаемым функцией `CURRENT_USER`, если такая схема существует, и пользователь имеет право `USAGE` для неё. (В противном случае элемент `$user` игнорируется.)

Схема системных каталогов, `pg_catalog`, просматривается всегда, независимо от того, указана она в пути или нет. Если она указана в пути, она просматривается в заданном порядке. Если же `pg_catalog` отсутствует в пути, эта схема будет просматриваться перед остальными элементами пути.

Аналогично всегда просматривается схема временных таблиц текущего сеанса, `pg_temp_nnn`, если она существует. Её можно включить в путь поиска, указав её псевдоним `pg_temp`. Если она отсутствует в пути, она будет просматриваться первой (даже перед `pg_catalog`). Временная схема просматривается только при поиске отношений (таблиц, представлений, последовательностей и т. д.) и типов данных, но никогда при поиске функций и операторов.

Когда объекты создаются без указания определённой целевой схемы, они помещаются в первую пригодную схему, указанную в `search_path`. Если путь поиска схем пуст, выдаётся ошибка.

По умолчанию этот параметр имеет значение "$user", public. При таком значении поддерживается совместное использование базы данных (когда пользователи не имеют личных схем, все используют схему public), использование личных схем, а также комбинация обоих вариантов. Другие подходы можно реализовать, изменяя значение пути по умолчанию, либо глобально, либо индивидуально для каждого пользователя.

default_transaction_isolation (enum)

Для каждой транзакции в SQL устанавливается уровень изоляции: «read uncommitted», «read committed», «repeatable read» или «serializable». Этот параметр задаёт уровень изоляции, который будет устанавливаться по умолчанию для новых транзакций. Значение этого параметра по умолчанию - «read committed».

transaction_isolation (enum)

Данный параметр отражает уровень изоляции текущей транзакции. В начале каждой транзакции ему присваивается текущее значение default_transaction_isolation. Последующая попытка изменить значение этого параметра равнозначна команде SET TRANSACTION.

3.2.9.2 Другие параметры по умолчанию

gin_fuzzy_search_limit (integer)

Задаёт мягкий верхний лимит для размера набора, возвращаемого при сканировании индексов GIN.

3.2.10 Предопределённые параметры

Следующие «параметры» доступны только для чтения. По этой причине они отсутствуют в примере файла postgresql.conf. Эти параметры отражают различные аспекты поведения PostgreDB, которые могут быть полезны в определённых приложениях,

например, клиентских средствах администрирования. Большинство из них задаётся при компиляции или при установке PostgreDB.

block_size (integer)

Сообщает размер блока на диске. Он определяется значением BLCKSZ при сборке сервера. Значение по умолчанию — 8192 байта. Значение block_size влияет на некоторые другие переменные конфигурации .

data_checksums (boolean)

Сообщает, включён ли в этом кластере контроль целостности данных.

data_directory_mode (integer)

Показывает разрешения (определённые data_directory), которые были установлены для каталога данных на момент запуска сервера.

debug_assertions (boolean)

Сообщает, был ли PostgreDB собран с проверочными утверждениями. Это имеет место, когда при сборке PostgreDB определяется макрос USE_ASSERT_CHECKING (например, при выполнении configure с флагом --enable-cassert). По умолчанию PostgreDB собирается без проверочных утверждений.

integer_datetimes (boolean)

Сообщает, был ли PostgreDB собран с поддержкой даты и времени в 64-битных целых.

in_hot_standby (boolean)

Сообщает, находится ли сервер в настоящий момент в режиме горячего резерва. Когда этот параметр имеет значение on (вкл.), все транзакции ограничивается

режимом «только чтение». В рамках сеанса это может измениться только в том случае, если сервер повышается до ведущего.

lc_collate (string)

Сообщает локаль, по правилам которой выполняется сортировка текстовых данных. Это значение определяется при создании базы данных.

lc_ctype (string)

Сообщает локаль, определяющую классификацию символов. Это значение определяется при создании базы данных. Обычно оно не отличается от `lc_collate`, но для некоторых приложений оно может быть определено по-другому.

max_function_args (integer)

Сообщает верхний предел для числа аргументов функции. Он определяется константой `FUNC_MAX_ARGS` при сборке сервера. По умолчанию установлен предел в 100 аргументов.

max_identifier_length (integer)

Сообщает максимальную длину идентификатора. Она определяется числом на 1 меньше, чем `NAMEDATALEN`, при сборке сервера. По умолчанию константа `NAMEDATALEN` равна 64; следовательно, `max_identifier_length` по умолчанию равна 63 байтам, но число символов в многобайтной кодировке будет меньше.

max_index_keys (integer)

Сообщает верхний предел для числа ключей индекса. Он определяется константой `INDEX_MAX_KEYS` при сборке сервера. По умолчанию установлен предел в 32 ключа.

segment_size (integer)

Сообщает, сколько блоков (страниц) можно сохранить в одном файловом сегменте. Это число определяется константой `RELSEG_SIZE` при сборке сервера. Максимальный размер сегмента в файлах равен произведению `segment_size` и `block_size`; по умолчанию это 1 гигабайт.

server_encoding (string)

Сообщает кодировку базы данных (набор символов). Она определяется при создании базы данных. Обычно клиентов должно интересовать только значение `client_encoding`.

server_version (string)

Сообщает номер версии сервера. Она определяется константой `PG_VERSION` при сборке сервера.

server_version_num (integer)

Сообщает номер версии сервера в виде целого числа. Она определяется константой `PG_VERSION_NUM` при сборке сервера.

ssl_library (string)

Сообщает имя библиотеки SSL, с которой был собран данный сервер PostgreDB (даже если SSL для данного экземпляра не настроен или не используется).

wal_block_size (integer)

Сообщает размер блока WAL на диске. Он определяется константой `XLOG_BLCKSZ` при сборке сервера. Значение по умолчанию — 8192 байта.

wal_segment_size (integer)

Сообщает размер сегментов журнала предзаписи. Значение по умолчанию — 16 МБ.

3.3 Аутентификация клиентского приложения

При подключении к серверу базы данных, клиентское приложение указывает имя пользователя СУБД, так же как и при обычном входе пользователя на компьютер с ОС Unix. При работе в среде SQL по имени пользователя определяется, какие у него есть права доступа к объектам базы данных. Следовательно, важно указать на этом этапе, к каким базам пользователь имеет право подключиться.

Аутентификация - это процесс идентификации клиента сервером базы данных, а также определение того, может ли клиентское приложение (или пользователь запустивший приложение) подключиться с указанным именем пользователя.

PostgreDB предлагает несколько различных методов аутентификации клиентов. Метод аутентификации конкретного клиентского соединения может основываться на адресе компьютера клиента, имени базы данных, имени пользователя.

Имена пользователей базы данных PostgreDB не имеют прямой связи с пользователями операционной системы на которой запущен сервер. Если у всех пользователей базы данных заведена учётная запись в операционной системе сервера, то имеет смысл назначить им точно такие же имена для входа в PostgreDB. Однако сервер, принимающий удалённые подключения, может иметь большое количество пользователей базы данных, у которых нет учётной записи в ОС. В таких случаях не требуется соответствие между именами пользователей базы данных и именами пользователей операционной системы.

3.3.1 Файл pg_hba.conf

Аутентификация клиентов управляется конфигурационным файлом, который традиционно называется `pg_hba.conf` и расположен в каталоге с данными кластера базы данных. (HBA расшифровывается как `host-based authentication` — аутентификации по имени узла). Файл `pg_hba.conf`, со стандартным содержимым,

создаётся командой `initdb` при инициализации каталога с данными. Однако его можно разместить в любом другом месте; см. конфигурационный параметр `hba_file`.

Общий формат файла `pg_hba.conf` — набор записей, по одной на строку. Пустые строки игнорируются, как и любой текст комментария после знака #. Запись может быть продолжена на следующей строке, для этого нужно завершить строку обратной косой чертой. (Обратная косая черта является спецсимволом только в конце строки). Запись состоит из нескольких полей, разделённых пробелами и/или табуляциями. Поля могут содержать пробелы, если содержимое этих полей заключено в кавычки. Если в кавычки берётся какое-либо ключевое слово в поле базы данных, пользователя или адресации (например, all или replication), то слово теряет своё особое значение и просто обозначает базу данных, пользователя или сервер с данным именем. Обратная косая черта обозначает перенос строки даже в тексте в кавычках или комментариях.

Каждая запись обозначает тип соединения, диапазон IP-адресов клиента (если он соотносится с типом соединения), имя базы данных, имя пользователя, и способ аутентификации, который будет использован для соединения в соответствии с этими параметрами. Первая запись с соответствующим типом соединения, адресом клиента, указанной базой данных и именем пользователя применяется для аутентификации. Процедур «`fall-through`» или «`backup`» не предусмотрено: если выбрана запись и аутентификация не прошла, последующие записи не рассматриваются. Если же ни одна из записей не подошла, в доступе будет отказано.

Записи могут иметь следующие форматы:

```
local      база  пользователь  метод-аутентификации [параметры-аутентификации]
host       база  пользователь  адрес  метод-аутентификации [параметры-аутентификации]
hostgssenc база  пользователь  адрес  метод-аутентификации [параметры-аутентификации]
hostnogssenc база пользователь адрес  метод-аутентификации [параметры-аутентификации]
```

Значения полей описаны ниже:

- **`local`** Управляет подключениями через Unix-сокеты. Без подобной записи подключения через Unix-сокеты невозможны.
- **`host`** Управляет подключениями, устанавливаемыми по TCP/IP. Записи host соответствуют подключениям с SSL и без SSL.
- **`база`** Определяет, каким именам баз данных соответствует эта запись. Значение all определяет, что подходят все базы данных.

Значение `sameuser` определяет, что данная запись соответствует, только если имя запрашиваемой базы данных совпадает с именем запрашиваемого пользователя. Значение `samerole` определяет, что запрашиваемый пользователь должен быть членом роли с таким же именем, как и у запрашиваемой базы данных. Суперпользователи, несмотря на то, что они имеют полные права, считаются включёнными в `samerole`, только когда они явно входят в такую роль, непосредственно или косвенно. Значение `replication` показывает, что запись соответствует, когда запрашивается подключение для физической репликации, но не когда запрашивается подключение для логической. Имейте в виду, что для подключений физической репликации не указывается какая-то конкретная база данных, в то время как для подключений логической репликации должна указываться конкретная база. Любое другое значение воспринимается как имя определённой базы данных. Несколько имён баз данных можно указать, разделяя их запятыми. Также можно задать отдельный файл с именами баз данных, написав имя файла после знака @.

`пользователь` указывает, какому имени (или именам) пользователя базы данных соответствует эта запись. Значение `all` указывает, что запись соответствует всем пользователям. Любое другое значение задаёт либо имя конкретного пользователя базы данных, либо имя группы (если это значение начинается с +). (Напомним, что в СУБД нет никакой разницы между пользователем и группой; знак + означает «совпадение любых ролей, которые прямо или косвенно являются членами роли», тогда как имя без знака + является подходящим только для этой конкретной роли). В связи с этим, суперпользователь рассматривается как член роли, только если он явно является членом этой роли, прямо или косвенно, а не только потому, что он

является суперпользователем. Несколько имён пользователей можно указать, разделяя их запятыми. Файл, содержащий имена пользователей, можно указать, поставив знак @ в начале его имени.

- **адрес** указывает адрес (или адреса) клиентской машины, которым соответствует данная запись. Это поле может содержать или имя компьютера, или диапазон IP-адресов, или одно из нижеупомянутых ключевых слов.

  Диапазон IP-адресов указывается в виде начального адреса диапазона, дополненного косой чертой (/) и длиной маски CIDR. Длина маски задаёт количество старших битов клиентского IP-адреса, которые должны совпадать с битами IP-адреса диапазона. Биты, находящиеся правее, в указанном IP-адресе должны быть нулевыми. Между IP-адресом, знаком / и длиной маски CIDR не должно быть пробельных символов.

  Типичные примеры диапазонов адресов IPv4, указанных таким образом: 172.20.143.89/32 для одного компьютера, 172.20.143.0/24 для небольшой и 10.6.0.0/16 для крупной сети. Диапазон адресов IPv6 может выглядеть как ::1/128 для одного компьютера (это адрес замыкания IPv6) или как fe80::7a31:c1ff:0000:0000/96 для небольшой сети. 0.0.0.0/0 представляет все адреса IPv4, а ::0/0 — все адреса IPv6. Чтобы указать один компьютер, используйте длину маски 32 для IPv4 или 128 для IPv6. Опускать замыкающие нули в сетевом адресе нельзя.

  Запись, сделанная в формате IPv4, подойдёт только для подключений по IPv4, а запись в формате IPv6 подойдёт только для подключений по IPv6, даже если представленный адрес находится в диапазоне IPv4-в-IPv6. Имейте в виду, что записи в формате IPv6 не будут приниматься, если системная библиотека C не поддерживает адреса IPv6.

  Вы также можете прописать значение `all`, чтобы указать любой IP-адрес, `samehost`, чтобы указать любые IP-адреса данного сервера, или `samenet`, чтобы указать любой адрес любой подсети, к которой сервер подключён напрямую.

Если определено имя компьютера (всё, что не является диапазоном IP-адресов или специальным ключевым словом, воспринимается как имя компьютера), то оно сравнивается с результатом обратного преобразования IP-адреса клиента (например, обратного DNS-запроса, если используется DNS). При сравнении имён компьютеров регистр не учитывается. Если имена совпали, выполняется прямое преобразование имени (например, прямой DNS-запрос) для проверки, относится ли клиентский IP-адрес к адресам, соответствующим имени. Если двусторонняя проверка пройдена, запись считается соответствующей компьютеру. (В качестве имени узла в файле `pg_hba.conf` должно указываться то, что возвращается при преобразовании IP-адреса клиента в имя, иначе строка не будет соответствовать узлу. Некоторые базы данных имён позволяют связать с одним IP-адресом несколько имён узлов, но операционная система при попытке разрешить IP-адрес возвращает только одно имя.)

Указание имени, начинающееся с точки (.), соответствует суффиксу актуального имени узла. Так, diasoft.ru будет соответствовать qdbhost.diasoft.ru (а не только diasoft.ru).

Когда в `pg_hba.conf` указываются имена узлов, следует добиться, чтобы разрешение имён выполнялось достаточно быстро. Для этого может быть полезен локальный кеш разрешения имён, например, `nscd`. Вы также можете включить конфигурационный параметр `log_hostname`, чтобы видеть в журналах имя компьютера клиента вместо IP-адреса.

Эти поля не применимы к записям `local`.

- **`IP-адрес IP-маска`** Эти два поля могут быть использованы как альтернатива записи IP-адрес/длина-маски. Вместо того, чтобы указывать длину маски, в отдельном столбце указывается сама маска. Например, 255.0.0.0 представляет собой маску CIDR для IPv4 длиной 8 бит, а 255.255.255.255 представляет маску CIDR длиной 32 бита.

  Эти поля не применимы к записям `local`.

- **`метод-аутентификации`** Указывает метод аутентификации, когда подключение соответствует этой записи.

- **trust** Разрешает безусловное подключение. Этот метод позволяет тому, кто может подключиться к серверу с базой данных, войти под любым желаемым пользователем СУБД без введения пароля и без какой-либо другой аутентификации.
- **reject** Отклоняет подключение безусловно. Эта возможность полезна для «фильтрации» некоторых серверов группы, например, строка reject может отклонить попытку подключения одного компьютера, при этом следующая строка позволяет подключиться остальным компьютерам в той же сети.
- **scram-sha-256** Проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256.
- **md5** Проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256 или MD5.
- **password** Требует для аутентификации введения клиентом незашифрованного пароля. Поскольку пароль посылается простым текстом через сеть, такой способ не стоит использовать, если сеть не вызывает доверия.
- **ident** Получает имя пользователя операционной системы клиента, связываясь с сервером `Ident`, и проверяет, соответствует ли оно имени пользователя базы данных. Аутентификация `ident` может использоваться только для подключений по TCP/IP. Для локальных подключений применяется аутентификация `peer`.
- **peer** Получает имя пользователя операционной системы клиента из операционной системы и проверяет, соответствует ли оно имени пользователя запрашиваемой базы данных. Доступно только для локальных подключений.
- **cert** Проводит аутентификацию, используя клиентский сертификат SSL.
- **параметры-аутентификации** После поля метод-аутентификации может идти поле (поля) вида имя=значение, определяющее параметры метода аутентификации. Подробнее о параметрах, доступных для различных методов аутентификации, рассказывается ниже.

Файлы, включённые в конструкции, начинающиеся с @, читаются, как список имён, разделённых запятыми или пробелами. Комментарии предваряются знаком #, как и в файле pg_hba.conf, и вложенные @ конструкции допустимы. Если только имя файла, начинающегося с @ не является абсолютным путём.

Поскольку записи файла `pg_hba.conf` рассматриваются последовательно для каждого подключения, порядок записей имеет большое значение. Обычно более ранние записи определяют чёткие критерии для соответствия параметров подключения, но для методов аутентификации допускают послабления. Напротив, записи более поздние смягчают требования к соответствию параметров подключения, но усиливают их в отношении методов аутентификации. Например, некто желает использовать trust аутентификацию для локального подключения по TCP/IP, но при этом запрашивать пароль для удалённых подключений по TCP/IP. В этом случае запись, устанавливающая аутентификацию trust для подключения адреса 127.0.0.1, должна предшествовать записи, определяющей аутентификацию по паролю для более широкого диапазона клиентских IP-адресов.

Файл `pg_hba.conf` прочитывается при запуске системы, а также в тот момент, когда основной сервер получает сигнал `SIGHUP`. Если вы редактируете файл во время работы системы, необходимо послать сигнал процессу `postmaster` (используя `pg_ctl reload`, вызвав SQL-функцию `pg_reload_conf()` или выполнив `kill -HUP`), чтобы он прочел обновлённый файл.

3.4 Роли базы данных

Роли базы данных концептуально полностью отличаются от пользователей операционной системы. На практике поддержание соответствия между ними может быть удобным, но не является обязательным. Роли базы данных являются глобальными для всей установки кластера базы данных (не для отдельной базы данных).

Для создания роли используется команда `SQL CREATE ROLE :`

```
CREATE ROLE имя;
```

где ***имя*** подчиняется правилам именования идентификаторов SQL.

Для удаления роли используется команда `DROP ROLE:`

```
DROP ROLE имя;
```

Для удобства поставляются программы `createuser` и `dropuser`, которые являются обёртками для этих команд SQL и вызываются из командной строки оболочки ОС: `createuser` ***имя***

`dropuser` ***имя***

Для получения списка существующих ролей, используйте `pg_roles` системного каталога или метакоманду `du` программы `psql`.

Для начальной настройки кластера базы данных, система сразу после инициализации всегда содержит одну предопределённую роль. Эта роль является суперпользователем («`superuser`») и по умолчанию (если не изменено при запуске `initdb`) имеет такое же имя, как и пользователь операционной системы, инициализирующий кластер баз данных. Обычно эта роль называется postgres. Для создания других ролей, вначале нужно подключиться с этой ролью.

Каждое подключение к серверу базы данных выполняется под именем конкретной роли, и эта роль определяет начальные права доступа для команд, выполняемых в этом соединении. Имя роли для конкретного подключения к базе данных указывается клиентской программой характерным для неё способом, таким образом инициируя запрос на подключение.

Список доступных для подключения ролей, который могут использовать клиенты, определяется конфигурацией аутентификации.

3.5 Управление базами данных

3.5.1 Создание базы данных

Для создания базы данных сервер PostgreDB должен быть развёрнут и запущен.

База данных создаётся SQL-командой `CREATE DATABASE`

`CREATE DATABASE` ***имя;***

где ***имя*** подчиняется правилам именования идентификаторов SQL. Текущий пользователь автоматически назначается владельцем. Владелец может удалить свою базу, что также приведёт к удалению всех её объектов, в том числе, имеющих других владельцев.

3.5.2 Удаление базы данных

Базы данных удаляются командой `DROP DATABASE`:

```
DROP DATABASE имя;
```

Лишь владелец базы данных или суперпользователь могут удалить базу. При удалении также удаляются все её объекты. Удаление базы данных это необратимая операция

3.5.3 Локализация

PostgreDB поддерживает два средства локализации:

Использование средств локализации операционной системы для обеспечения определяемых локалью порядка правил сортировки, форматирования чисел, перевода сообщений и прочих аспектов;

Обеспечение возможностей использования различных кодировок для хранения текста на разных языках и перевода кодировок между клиентом и сервером.

3.5.4 Регламентные задачи обслуживания базы данных

Задачи, которые рассматриваются в этой главе, являются обязательными, но они по природе своей повторяющиеся и легко поддаются автоматизации с использованием стандартных средств, таких как задания `cron`. Создание соответствующих заданий и контроль над их успешным выполнением входят в обязанности администратора базы данных.

Одной из очевидных задач обслуживания СУБД является регулярное создание резервных копий данных. При отсутствии свежей резервной копии у вас не будет шанса восстановить систему после катастрофы (сбой диска, пожар, удаление важной таблицы по ошибке и т. д.).

Другое важное направление обслуживания СУБД — периодическая «очистка» базы данных. С ней тесно связано обновление статистики, которая будет использоваться планировщиком запросов.

Ещё одной задачей, требующей периодического выполнения, является управление файлами журнала.

Для контроля состояния базы данных и для отслеживания нестандартных ситуаций можно использовать `check_postgres`.

Регламентная очистка

Базы данных PostgreDB требуют периодического проведения процедуры обслуживания, которая называется очисткой. Во многих случаях очистку достаточно выполнять с помощью демона автоочистки. Можно дополнить или заменить действие этого демона командами `VACUUM`.

Команды `VACUUM` должны обрабатывать каждую таблицу по следующим причинам:

Для высвобождения или повторного использования дискового пространства, занятого изменёнными или удалёнными строками.

Для обновления статистики по данным, используемой планировщиком запросов.

Для обновления карты видимости, которая ускоряет сканирование только индекса.

Для предотвращения потери очень старых данных из-за зацикливания идентификаторов транзакций или мультитранзакций.

Регулярная переиндексация

В некоторых ситуациях стоит периодически перестраивать индексы, выполняя команду `REINDEX` или последовательность отдельных шагов по восстановлению индексов.

Страницы индексов на основе B-деревьев, которые стали абсолютно пустыми, могут быть использованы повторно. Однако возможность неэффективного использования пространства всё же остаётся: если со страницы были удалены почти все, но не все ключи индекса, страница всё равно остаётся занятой. Следовательно, шаблон использования, при котором со временем удаляются многие, но не все ключи в каждом диапазоне, приведёт к неэффективному расходованию пространства. В таких случаях рекомендуется периодически проводить переиндексацию.

Обслуживание журнала

Журнал бесценен при диагностике проблем. Однако он может быть очень объёмным, так что хранить его неограниченно долго вы вряд ли захотите. Поэтому необходимо организовать ротацию журнальных файлов так, чтобы новые файлы создавались, а старые удалялись через разумный промежуток времени.

Существует встроенное средство ротации журнальных файлов, которое можно использовать, установив для параметра `logging_collector` значение true в `postgresql.conf`.

Можно также использовать внешнюю программу для ротации журнальных файлов. Один из вариантов — направить вывод `stderr` сервера в желаемую программу.

Ещё одно решение заключается в передаче журнала в `syslog`, чтобы ротацией файлов занималась уже служба `syslog`. Для этого присвойте параметру конфигурации `log_destination` значение `syslog` (для вывода журнала только в `syslog`) в `postgresql.conf`. Однако во многих системах, а особенно с большими сообщениями, `syslog` работает не очень надёжно; он может обрезать или терять сообщения как раз тогда, когда они вам нужны. Кроме того, в Linux, `syslog`> сбрасывает каждое сообщение на диск, от чего страдает производительность. (Для отключения этой синхронной записи можно добавить «-» перед именем файла в файле конфигурации `syslog`.)

3.5.5 Резервное копирование и восстановление

Как и всё, что содержит важные данные, базы данных PostgreDB следует регулярно сохранять в резервной копии. Хотя эта процедура по существу проста, важно чётко понимать лежащие в её основе приёмы и положения.

Существует три фундаментально разных подхода к резервному копированию данных в PostgreDB:

- Выгрузка в SQL;
- Копирование на уровне файлов;
- Непрерывное архивирование.

3.5.5.1 Выгрузка в SQL

Идея, стоящая за этим методом, заключается в генерации текстового файла с командами SQL, которые при выполнении на сервере пересоздадут базу данных в том же самом состоянии, в котором она была на момент выгрузки. PostgreDB предоставляет для этой цели вспомогательную программу pg_dump. Применение этой программы выглядит так:

```
pg_dump имя_базы > файл_дампа
```

В то время как вышеупомянутая команда создаёт текстовый файл, pg_dump может создать файлы и в других форматах, которые допускают параллельную обработку и более гибкое управление восстановлением объектов.

Программа pg_dump является для PostgreDB обычным клиентским приложением. Это означает, что вы можете выполнять процедуру резервного копирования с любого удалённого компьютера, если имеете доступ к нужной базе данных. Но необходимо помнить, что pg_dump не использует для своей работы какие-то специальные привилегии. В частности, ей обычно требуется доступ на чтение всех таблиц, которые вы хотите выгрузить, так что для копирования всей базы данных практически всегда её нужно запускать с правами суперпользователя СУБД. (Если у вас нет достаточных прав для резервного копирования всей базы данных, вы тем не менее можете сделать резервную копию той части базы, доступ к которой у вас есть, используя такие параметры, как -n схема или -t таблица.)

Указать, к какому серверу должна подключаться программа pg_dump, можно с помощью аргументов командной строки -h сервер и -p порт. По умолчанию в качестве сервера выбирается localhost или значение, указанное в переменной окружения PGHOST. Подобным образом, по умолчанию используется порт, заданный в переменной окружения PGPORT, а если она не задана, то порт, указанный по умолчанию при компиляции.

Как и любое другое клиентское приложение PostgreDB, pg_dump по умолчанию будет подключаться к базе данных с именем пользователя, совпадающим с именем текущего пользователя операционной системы. Чтобы переопределить имя, либо добавьте параметр -U, либо установите переменную окружения PGUSER. Помните, что pg_dump подключается к серверу через обычные механизмы проверки подлинности клиента.

Важное преимущество pg_dump в сравнении с другими методами резервного копирования, описанными далее, состоит в том, что вывод pg_dump обычно можно загрузить в более новые версии PostgreDB, в то время как резервная копия на уровне файловой системы и непрерывное архивирование жёстко зависят от версии сервера. Также, только метод с применением pg_dump будет работать при переносе базы данных на другую машинную архитектуру, например, при переносе с 32-битной на 64-битную версию сервера.

Дампы, создаваемые pg_dump, являются внутренне согласованными, то есть, дамп представляет собой снимок базы данных на момент начала запуска pg_dump. pg_dump не блокирует другие операции с базой данных во время своей работы. (Исключение составляют операции, которым нужна исключительная блокировка, как например, большинство форм команды ALTER TABLE.)

3.5.5.2 Восстановление дампа

Текстовые файлы, созданные pg_dump, предназначаются для последующего чтения программой psql. Общий вид команды для восстановления дампа:

```
psql имя_базы < файл_дампа
```

где файл_дампа — это файл, содержащий вывод команды pg_dump. База данных, заданная параметром имя_базы, не будет создана данной командой, так что вы должны создать её сами из базы template0 перед запуском psql (например, с помощью команды createdb -T template0 имя_базы). Программа psql принимает параметры, указывающие сервер, к которому осуществляется подключение, и имя пользователя, подобно pg_dump. Дампы, выгруженные не в текстовом формате, восстанавливаются утилитой pg_restore.

Перед восстановлением SQL-дампа все пользователи, которые владели объектами или имели права на объекты в выгруженной базе данных, должны уже существовать. Если их нет, при восстановлении будут ошибки пересоздания объектов с изначальными владельцами и/или правами. (Иногда это желаемый результат, но обычно нет).

По умолчанию, если происходит ошибка SQL, программа psql продолжает выполнение. Если же запустить psql с установленной переменной ON_ERROR_STOP, это поведение поменяется и psql завершится с кодом 3 в случае возникновения ошибки SQL:

```
psql --set ON_ERROR_STOP=on имя_базы < файл_дампа
```

В любом случае вы получите только частично восстановленную базу данных. В качестве альтернативы можно указать, что весь дамп должен быть восстановлен в одной транзакции, так что восстановление либо полностью выполнится, либо полностью отменится. Включить данный режим можно, передав psql аргумент -1 или --single-transaction. Выбирая этот режим, учтите, что даже незначительная ошибка может привести к откату восстановления, которое могло продолжаться несколько часов. Однако это всё же может быть предпочтительней, чем вручную вычищать сложную базу данных после частично восстановленного дампа.

После восстановления резервной копии имеет смысл запустить ANALYZE для каждой базы данных, чтобы оптимизатор запросов получил полезную статистику.

3.5.5.3 Использование pg_dumpall

Программа pg_dump выгружает только одну базу данных в один момент времени и не включает в дамп информацию о ролях и табличных пространствах (так как это информация уровня кластера, а не самой базы данных). Для удобства создания дампа всего содержимого кластера баз данных предоставляется программа pg_dumpall, которая делает резервную копию всех баз данных кластера, а также сохраняет данные уровня кластера, такие как роли и определения табличных пространств. Простое использование этой команды:

```
pg_dumpall > файл_дампа
```

Полученную копию можно восстановить с помощью psql:

```
psql -f файл_дампа postgres
```

Восстанавливать дамп, который выдала pg_dumpall, всегда необходимо с правами суперпользователя, так как они требуются для восстановления информации о ролях и

табличных пространствах. Если вы используете табличные пространства, убедитесь, что пути к табличным пространствам в дампе соответствуют новой среде.

pg_dumpall выдаёт команды, которые заново создают роли, табличные пространства и пустые базы данных, а затем вызывает для каждой базы pg_dump. Таким образом, хотя каждая база данных будет внутренне согласованной, состояние разных баз не будет синхронным.

Только глобальные данные кластера можно выгрузить, передав pg_dumpall ключ --globals-only. Это необходимо, чтобы полностью скопировать кластер, когда pg_dump выполняется для отдельных баз данных.

3.5.5.4 Управление большими базами данных

Некоторые операционные системы накладывают ограничение на максимальный размер файла, что приводит к проблемам при создании больших файлов с помощью pg_dump. pg_dump может писать в стандартный вывод, так что вы можете использовать стандартные инструменты Unix для того, чтобы избежать потенциальных проблем. Вот несколько возможных методов:

- Используйте сжатые дампы. Вы можете использовать предпочитаемую программу сжатия, например gzip:
  pg_dump имя_базы | gzip > имя_файла.gz
  Затем загрузить сжатый дамп можно командой:
  gunzip -c имя_файла.gz | psql имя_базы или **cat имя_файла.gz | gunzip | psql имя_базы;**

- Используйте split. Команда split может разбивать выводимые данные на небольшие файлы, размер которых удовлетворяет ограничению нижележащей файловой системы. Например, чтобы получить части по 2 гигабайта:
  pg_dump имя_базы | split -b 2G - имя_файла
  Восстановить их можно так:
  cat имя_файла* | psql имя_базы

Использовать GNU `split` можно вместе с gzip:

```
pg_dump имя_базы | split -b 2G --filter='gzip > $FILE.gz'
```

Восстановить данные после такого разбиения можно с помощью команды zcat;

- Используйте специальный формат дампа `pg_dump`. Если при сборке PostgreDB была подключена библиотека `zlib`, дамп в специальном формате будет записываться в файл в сжатом виде. В таком формате размер файла дампа будет близок к размеру, полученному с применением `gzip`, но он лучше тем, что позволяет восстанавливать таблицы выборочно. Следующая команда выгружает базу данных в специальном формате:

  ```
  pg_dump -Fc имя_базы > имя_файла
  ```

  Дамп в специальном формате не является скриптом для `psql` и должен восстанавливаться с помощью команды `pg_restore`, например:

  ```
  pg_restore -d имя_базы имя_файла;
  ```

- Используйте возможность параллельной выгрузки в `pg_dump`. Чтобы ускорить выгрузку большой БД, вы можете использовать режим параллельной выгрузки в `pg_dump`. При этом одновременно будут выгружаться несколько таблиц. Управлять числом параллельных заданий позволяет параметр -j. Параллельная выгрузка поддерживается только для формата архива в каталоге.

  ```
  pg_dump -j число -F d -f выходной_каталог имя_базы
  ```

  Вы также можете восстановить копию в параллельном режиме с помощью `pg_restore -j`. Это поддерживается для любого архива в формате каталога или специальном формате, даже если архив создавался не командой `pg_dump -j`.

3.5.5.5 Резервное копирование на уровне файлов

Альтернативной стратегией резервного копирования является непосредственное копирование файлов, в которых PostgreDB хранит содержимое базы данных. Вы можете использовать любой способ копирования файлов по желанию, например:

```
tar -cf backup.tar /usr/local/pgsql/data
```

Однако существуют два ограничения, которые делают этот метод непрактичным или как минимум менее предпочтительным по сравнению с `pg_dump`:

- Чтобы полученная резервная копия была годной, сервер баз данных должен быть остановлен. Такие полумеры, как запрещение всех подключений к серверу, работать не будут (отчасти потому что `tar` и подобные средства не получают мгновенный снимок состояния файловой системы, но ещё и потому, что в сервере есть внутренние буферы). Необходимо отметить, что сервер нужно будет остановить и перед восстановлением данных;
- Если вы ознакомились с внутренней организацией базы данных в файловой системе, у вас может возникнуть соблазн скопировать или восстановить только отдельные таблицы или базы данных в соответствующих файлах или каталогах. Это не будет работать, потому что информацию, содержащуюся в этих файлах, нельзя использовать без файлов журналов транзакций, `pg_xact/*`, которые содержат состояние всех транзакций. Без этих данных файлы таблиц непригодны к использованию. Разумеется, также невозможно восстановить только одну таблицу и соответствующие данные `pg_xact`, потому что в результате нерабочими станут все другие таблицы в кластере баз данных. Таким образом, копирование на уровне файловой системы будет работать, только если выполняется полное копирование и восстановление всего кластера баз данных.

Ещё один подход к резервному копированию файловой системы заключается в создании «целостного снимка» каталога с данными, если это поддерживает файловая система. Типичная процедура включает создание «замороженного снимка» тома, содержащего базу данных, затем копирование всего каталога с данными из этого снимка на устройство резервного копирования, и наконец освобождение замороженного снимка. При этом

сервер базы данных может не прекращать свою работу. Однако резервная копия, созданная таким способом, содержит файлы базы данных в таком состоянии, как если бы сервер баз данных не был остановлен штатным образом; таким образом, когда вы запустите сервер баз данных с сохранёнными данными, он будет считать, что до этого процесс сервера был прерван аварийно, и будет накатывать журнал WAL. Это не проблема, просто имейте это в виду (и обязательно включите файлы WAL в резервную копию). Чтобы сократить время восстановления, можно выполнить команду `CHECKPOINT` перед созданием снимка.

Если ваша база данных размещена в нескольких файловых системах, получить в точности одновременно замороженные снимки всех томов может быть невозможно. Например, если файлы данных и журналы WAL находятся на разных дисках или табличные пространства расположены в разных файловых системах, резервное копирование со снимками может быть неприменимо, потому что снимки должны быть одновременными. В таких ситуациях очень внимательно изучите документацию по вашей файловой системе, прежде чем довериться технологии согласованных снимков.

Если одновременные снимки невозможны, остаётся вариант с остановкой сервера баз данных на время, достаточное для получения всех замороженных снимков. Другое возможное решение — получить базовую копию путём непрерывного архивирования, такие резервные копии не могут пострадать от изменений файловой системы в процессе резервного копирования. Для этого требуется включить непрерывное архивирование только на время резервного копирования; для восстановления применяется процедура восстановления из непрерывного архива.

Ещё один вариант — копировать содержимое файловой системы с помощью `rsync`. Для этого `rsync` запускается сначала во время работы сервера баз данных, а затем сервер останавливается на время, достаточное для запуска `rsync --checksum`. (Ключ `--checksum` необходим, потому что `rsync` различает время только с точностью до секунд.) Во второй раз `rsync` отработает быстрее, чем в первый, потому что скопировать надо будет относительно немного данных; и в итоге будет получен согласованный результат, так как сервер был остановлен. Данный метод позволяет получить копию на уровне файловой системы с минимальным временем простоя.

Обратите внимание, что размер копии на уровне файлов обычно больше, чем дампа SQL. (Программе `pg_dump` не нужно, например, записывать содержимое индексов, достаточно команд для их пересоздания). Однако копирование на уровне файлов может выполняться быстрее.

3.5.5.6 Непрерывное архивирование и восстановление на момент времени (Point-in-Time Recovery, PITR)

Всё время в процессе работы PostgreDB ведёт журнал предзаписи (WAL), который расположен в подкаталоге `pg_wal/` каталога с данными кластера баз данных. В этот журнал записываются все изменения, вносимые в файлы данных. Прежде всего, журнал существует для безопасного восстановления после краха сервера: если происходит крах, целостность СУБД может быть восстановлена в результате «воспроизведения» записей, зафиксированных после последней контрольной точки. Однако наличие журнала делает возможным использование третьей стратегии копирования баз данных: можно сочетать резервное копирование на уровне файловой системы с копированием файлов WAL. Если потребуется восстановить данные, мы можем восстановить копию файлов, а затем воспроизвести журнал из скопированных файлов WAL, и таким образом привести систему в нужное состояние. Такой подход более сложен для администрирования, чем любой из описанных выше, но он имеет значительные преимущества:

- В качестве начальной точки для восстановления необязательно иметь полностью согласованную копию на уровне файлов. Внутренняя несогласованность копии будет исправлена при воспроизведении журнала (практически то же самое происходит при восстановлении после краха). Таким образом, согласованный снимок файловой системы не требуется, вполне можно использовать tar или похожие средства архивации;
- Поскольку при воспроизведении можно обрабатывать неограниченную последовательность файлов WAL, непрерывную резервную копию можно получить, просто продолжая архивировать файлы WAL. Это особенно ценно для больших баз данных, полные резервные копии которых делать как минимум неудобно;

- Воспроизводить все записи WAL до самого конца нет необходимости. Воспроизведение можно остановить в любой точке и получить целостный снимок базы данных на этот момент времени. Таким образом, данная технология поддерживает восстановление на момент времени: можно восстановить состояние базы данных на любое время с момента создания резервной копии;
- Если непрерывно передавать последовательность файлов WAL другому серверу, получившему данные из базовой копии того же кластера, получается система тёплого резерва: в любой момент мы можем запустить второй сервер и он будет иметь практически текущую копию баз данных.

Как и обычное резервное копирование файловой системы, этот метод позволяет восстанавливать только весь кластер баз данных целиком, но не его части. Кроме того, для архивов требуется большое хранилище: базовая резервная копия может быть объёмной, а нагруженные системы будут генерировать многие мегабайты трафика WAL, который необходимо архивировать. Тем не менее этот метод резервного копирования предпочитается во многих ситуациях, где необходима высокая надёжность.

Для успешного восстановления с применением непрерывного архивирования (также называемого «оперативным резервным копированием» многими разработчиками СУБД), вам необходима непрерывная последовательность заархивированных файлов WAL, начинающаяся не позже, чем с момента начала копирования. Так что для начала вы должны настроить и протестировать процедуру архивирования файлов WAL до того, как получите первую базовую копию. Соответственно, сначала мы обсудим механику архивирования файлов WAL.

3.5.5.7 Настройка архивирования WAL

В абстрактном смысле, запущенная СУБД PostgreDB производит неограниченно длинную последовательность записей WAL. СУБД физически делит эту последовательность на файлы сегментов WAL, которые обычно имеют размер 16 МБ (хотя размер сегмента может быть изменён при `initdb`). Файлы сегментов получают цифровые имена, которые отражают их позицию в абстрактной последовательности WAL. Когда архивирование WAL не применяется, система обычно создаёт только несколько файлов сегментов и

затем «перерабатывает» их, меняя номер в имени ставшего ненужным файла на больший. Предполагается, что файлы сегментов, содержимое которых предшествует последней контрольной точке, уже не представляют интереса и могут быть переработаны.

При архивировании данных WAL необходимо считывать содержимое каждого файла-сегмента, как только он заполняется, и сохранять эти данные куда-то, прежде чем файл-сегмент будет переработан и использован повторно. В зависимости от применения и доступного аппаратного обеспечения, возможны разные способы «сохранить данные куда-то»: можно скопировать файлы-сегменты в смонтированный по NFS каталог на другую машину, записать их на ленту (убедившись, что у вас есть способ идентифицировать исходное имя каждого файла) или собрать их в пакет и записать на CD, либо какие-то совсем другие варианты. Чтобы у администратора баз данных была гибкость в этом плане, PostgreDB пытается не делать каких-либо предположений о том, как будет выполняться архивация. Вместо этого, PostgreDB позволяет администратору указать команду оболочки, которая будет запускаться для копирования завершённого файла-сегмента в нужное место. Эта команда может быть простой как cp, а может вызывать сложный скрипт оболочки — это решать вам.

Чтобы включить архивирование WAL, установите в параметре конфигурации `wal_level` уровень `replica` (или выше), в `archive_mode` — значение on, и задайте желаемую команду оболочки в параметре `archive_command`. На практике эти параметры всегда задаются в файле `postgresql.conf`. В `archive_command` символы %p заменяются полным путём к файлу, подлежащему архивации, а %f заменяются только именем файла. (Путь задаётся относительно текущего рабочего каталога, т. е. каталога данных кластера). Если в команду нужно включить сам символ %, запишите %%. Простейшая команда, которая может быть полезна:

Windows:

```
archive_command = 'copy "%p" "C:\\server\\archivedir\\%f"'
```

Unix:

```
archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
```

Вышеуказанные команды будут копировать архивируемые сегменты WAL в каталог `/mnt/server/archivedir`.

Команда архивирования будет запущена от имени того же пользователя, от имени которого работает сервер PostgreDB. Поскольку архивируемые последовательности файлов WAL фактически содержат всё, что есть в вашей базе данных, вам нужно будет защитить архивируемые данные от посторонних глаз; например, сохраните архив в каталог, чтение которого запрещено для группы и остальных пользователей.

Важно, чтобы команда архивирования возвращала нулевой код завершения, если и только если она завершилась успешно. Получив нулевой результат, PostgreDB будет полагать, что файл успешно заархивирован и удалит его или переработает. Однако ненулевой код состояния скажет PostgreDB, что файл не заархивирован; попытки заархивировать его будут периодически повторяться, пока это не удастся.

Когда команда архивирования завершается сигналом (отличным от SIGTERM, получаемым при штатном отключении сервера) или при возникновении ошибки оболочки (например, если команда не найдена), процесс архиватора прерывается и перезапускается управляющим процессом `postmaster`. В таких случаях в `pg_stat_archiver` не сообщается об ошибке.

Команда архивирования обычно разрабатывается так, чтобы не допускать перезаписи любых существующих архивных файлов. Это важная мера безопасности, позволяющая сохранить целостность архива в случае ошибки администратора (например, если архивируемые данные двух разных серверов будут сохраняться в одном каталоге).

Рекомендуется протестировать команду архивирования, чтобы убедиться, что она действительно не перезаписывает существующие файлы, и что она возвращает ненулевое состояние в этом случае. В показанной выше команде для Unix для этого добавлен отдельный шаг `test`. На некоторых платформах Unix у cp есть ключ -i, который позволяет сделать то же, но менее явно; но не проверив, какой код состояния при этом возвращается, полагаться на этот ключ не следует.

Разрабатывая схему архивирования, продумайте, что произойдёт, если команда архивирования начнёт постоянно выдавать ошибку, потому что требуется вмешательство оператора или для архивирования не хватает места. Например, это может произойти, если

вы записываете архивы на ленточное устройство без механизма автозамены; когда лента заполняется полностью, больше ничего архивироваться не будет, пока вы не замените кассету. Вы должны убедиться, что любые возникающие ошибки или обращения к человеку (оператору), обрабатываются так, чтобы проблема решалась достаточно быстро. Пока она не разрешится, каталог `pg_wal/` продолжит наполняться файлами-сегментами WAL.

Не важно, с какой скоростью работает команда архивирования, если только она не ниже средней скорости, с которой сервер генерирует записи WAL. Обычно работа продолжается, даже если процесс архивирования немного отстаёт. Если же архивирование отстаёт значительно, это приводит к увеличению объёма данных, которые могут быть потеряны в случае аварии. При этом каталог `pg_wal/` будет содержать большое количество ещё не заархивированных файлов-сегментов, которые в конце концов могут занять всё доступное дисковое пространство. Поэтому рекомендуется контролировать процесс архивации и следить за тем, чтобы он выполнялся как задумано.

При написании команды архивирования вы должны иметь в виду, что имена файлов для архивирования могут иметь длину до 64 символов и содержать любые комбинации из цифр, точек и букв ASCII. Сохранять исходный относительный путь (%p) необязательно, но необходимо сохранять имя файла (%f).

Обратите внимание, что архивирование WAL позволяет сохранить любые изменения данных, произведённые в базе данных PostgreDB, оно не затрагивает изменения, внесённые в конфигурационные файлы (такие как `postgresql.conf`, `pg_hba.conf` и `pg_ident.conf`), поскольку эти изменения выполняются вручную, а не через SQL. Поэтому имеет смысл разместить конфигурационные файлы там, где они будут заархивированы обычными процедурами копирования файлов. Как перемещать конфигурационные файлы.

Команда архивирования вызывается, только когда сегмент WAL заполнен до конца. Таким образом, если сервер постоянно генерирует небольшой трафик WAL (или есть продолжительные периоды, когда это происходит), между завершением транзакций и их безопасным сохранением в архиве может образоваться большая задержка. Чтобы ограничить время жизни неархивированных данных, можно установить

archive_timeout, чтобы сервер переключался на новый файл сегмента WAL как минимум с заданной частотой. Заметьте, что неполные файлы, архивируемые досрочно из-за принудительного переключения по тайм-ауту, будут иметь тот же размер, что и заполненные файлы. Таким образом, устанавливать очень маленький archive_timeout — неразумно; это приведёт к неэффективному заполнению архива. Обычно подходящее значение archive_timeout — минута или около того.

Также вы можете принудительно переключить сегмент WAL вручную с помощью pg_switch_wal, если хотите, чтобы только что завершённая транзакция заархивировалась как можно скорее.

Когда wal_level имеет значение minimal, некоторые команды SQL выполняются в обход журнала WAL. Если архивирование или потоковая репликация были включены во время выполнения таких операторов, WAL не будет содержать информацию, необходимую для восстановления. (На восстановление после краха это не распространяется). Поэтому wal_level можно изменить только при запуске сервера. Однако для изменения команды archive_command достаточно перезагрузить файл конфигурации. Если вы хотите на время остановить архивирование, это можно сделать, например, задав в качестве значения archive_command пустую строку (''). В результате файлы WAL будут накапливаться в каталоге pg_wal/, пока не будет восстановлена действующая команда archive_command.

3.5.5.8 Создание базовой резервной копии

Проще всего получить базовую резервную копию, используя программу pg_basebackup. Эта программа сохраняет базовую копию в виде обычных файлов или в архиве tar. Если гибкости pg_basebackup не хватает, вы также можете получить базовую резервную копию, используя низкоуровневый API.

Продолжительность создания резервной копии обычно не имеет большого значения. Однако если вы эксплуатируете сервер с отключённым режимом full_page_writes, вы можете заметить падение производительности в процессе резервного копирования, так как режим full_page_writes включается принудительно на время резервного копирования.

Чтобы резервной копией можно было пользоваться, нужно сохранить все файлы сегментов WAL, сгенерированные во время и после копирования файлов. Для облегчения этой задачи, процесс создания базовой резервной копии записывает файл истории резервного копирования, который немедленно сохраняется в области архивации WAL. Данный файл получает имя по имени файла первого сегмента WAL, который потребуется для восстановления скопированных файлов. Например, если начальный файл WAL назывался 000000010000123400000055CD, файл истории резервного копирования получит имя 000000010000123400000055CD.007C9330.backup. (Вторая часть имени файла обозначает точную позицию внутри файла WAL и обычно может быть проигнорирована.) Как только вы заархивировали копии файлов данных и файлов сегментов WAL, полученных в процессе копирования (по сведениям в файле истории резервного копирования), все заархивированные сегменты WAL с именами, меньшими по номеру, становятся ненужными для восстановления файловой копии и могут быть удалены. Но всё же рассмотрите возможность хранения нескольких наборов резервных копий, чтобы быть абсолютно уверенными, что вы сможете восстановить ваши данные.

Файл истории резервного копирования — это просто небольшой текстовый файл. В него записывается метка, которая была передана `pg_basebackup`, а также время и текущие сегменты WAL в момент начала и завершения резервной копии. Если вы связали с данной меткой соответствующий файл дампа, то заархивированного файла истории достаточно, чтобы найти файл дампа, нужный для восстановления.

Поскольку необходимо хранить все заархивированные файлы WAL с момента последней базовой резервной копии, интервал базового резервного копирования обычно выбирается в зависимости от того, сколько места может быть выделено для архива файлов WAL. Также стоит отталкиваться от того, сколько вы готовы ожидать восстановления, если оно понадобится — системе придётся воспроизвести все эти сегменты WAL, а этот процесс может быть долгим, если с момента последней базовой копии прошло много времени.

3.5.5.9 Создание базовой резервной копии через низкоуровневый API

Процедура создания базовой резервной копии с использованием низкоуровневого API содержит чуть больше шагов, чем метод `pg_basebackup`, но всё же относительно

проста. Очень важно, чтобы эти шаги выполнялись по порядку, и следующий шаг выполнялся, только если предыдущий успешен.

Резервное копирование на низком уровне можно произвести в монопольном или немонопольном режиме. Рекомендуется применять немонопольный метод, а монопольный считается устаревшим и в конце концов будет ликвидирован.

Немонопольное резервное копирование позволяет параллельно запускать другие процессы копирования (используя тот же API или `pg_basebackup`).

Инструкция по запуску немонопольного копирования:

- Убедитесь, что архивирование WAL включено и работает.
- Подключитесь к серверу (к любой базе данных) как пользователь с правами на выполнение `pg_start_backup` (суперпользователь или пользователь, которому дано право EXECUTE для этой функции) и выполните команду: `SELECT pg_start_backup('label', false, false);`

  где `label` — любая метка, по которой можно однозначно идентифицировать данную операцию резервного копирования. Соединение, через которое вызывается `pg_start_backup`, должно поддерживаться до окончания резервного копирования, иначе этот процесс будет автоматически прерван.

  По умолчанию `pg_start_backup` может выполняться длительное время. Это объясняется тем, что функция выполняет контрольную точку, а операции ввода/вывода, требуемые для этого, распределяются в интервале времени, по умолчанию равном половине интервала между контрольными точками (см. параметр `checkpoint_completion_target`). Обычно это вполне приемлемо, так как при этом минимизируется влияние на выполнение других запросов. Если же вы хотите начать резервное копирование максимально быстро, передайте во втором параметре true. В этом случае контрольная точка будет выполнена немедленно без ограничения объёма ввода/вывода. Третий параметр, имеющий значение false, указывает `pg_start_backup` начать немонопольное базовое копирование.

- Скопируйте файлы, используя любое удобное средство резервного копирования, например, `tar` или `cpio` (не `pg_dump` или `pg_dumpall`). В процессе копирования останавливать работу базы данных не требуется, это ничего не даёт.
- Через то же подключение, что и раньше, выполните команду: `SELECT * FROM pg_stop_backup(false, true);`
- При этом сервер выйдет из режима резервного копирования. Ведущий сервер вместе с этим автоматически переключится на следующий сегмент WAL. На ведомом автоматическое переключение сегментов WAL невозможно, поэтому вы можете выполнить `pg_switch_wal` на ведущем, чтобы произвести переключение вручную. Такое переключение позволяет получить готовый к архивированию последний сегмент WAL, записанный в процессе резервного копирования. Функция `pg_stop_backup` возвратит одну строку с тремя значениями. Второе из них нужно записать в файл `backup_label` в корневой каталог резервной копии. Третье значение, если оно не пустое, должно быть записано в файл `tablespace_map`. Эти файлы крайне важны для восстановления копии и должны записываться байт за байтом без изменений, для чего может потребоваться открыть файл в двоичном редакторе.
- После этого останется заархивировать файлы сегментов WAL, активных во время создания резервной копии, и процедура резервного копирования будет завершена. Функция `pg_stop_backup` в первом значении результата указывает, какой последний сегмент требуется для формирования полного набора файлов резервной копии. На ведущем сервере, если включён режим архивации (параметр `archive_mode`) и аргумент `wait_for_archive` равен true, функция `pg_stop_backup` не завершится, пока не будет заархивирован последний сегмент. На ведомом значением `archive_mode` должно быть always, чтобы `pg_stop_backup` ожидала архивации. Эти файлы будут заархивированы автоматически, поскольку также должна быть настроена команда `archive_command`. Чаще всего это происходит быстро, но мы советуем наблюдать за системой архивации и проверять, не возникают ли

задержки. Если архивирование остановится из-за ошибок команды архивации, попытки архивации будут продолжаться до успешного завершения, и только тогда резервное копирование окончится. Если вы хотите ограничить время выполнения `pg_stop_backup`, установите соответствующее значение в `statement_timeout`, но заметьте, что в случае прерывания `pg_stop_backup` по времени резервная копия может оказаться негодной. Если в процедуре резервного копирования предусмотрено отслеживание и архивация всех файлов сегментов WAL, необходимых для резервной копии, то в аргументе `wait_for_archive` (по умолчанию равном true) можно передать false, чтобы функция `pg_stop_backup` завершилась сразу, как только в WAL будет помещена запись о завершении копирования. По умолчанию `pg_stop_backup` будет ждать окончания архивации всех файлов WAL, что может занять некоторое время. Использовать этот параметр следует с осторожностью: если архивация WAL не контролируется, в резервной копии могут оказаться не все необходимые файлы WAL и её нельзя будет восстановить.

Монопольное резервное копирование считается устаревшим, так что от него следует отказаться. Ранее это был единственный возможный метод низкоуровневого копирования, но сейчас пользователям рекомендуется по возможности подкорректировать свои скрипты и перейти к использованию немонопольного варианта.

3.5.5.10 Восстановление непрерывной архивной копии

Допустим, вам необходимо восстановить базу данных из резервной копии. Порядок действий таков:

1. Остановите сервер баз данных, если он запущен;

2. Если у вас есть место для этого, скопируйте весь текущий каталог кластера баз данных и все табличные пространства во временный каталог на случай, если они вам понадобятся. Учтите, что эта мера предосторожности требует, чтобы свободного места на диске было достаточно для размещения двух копий существующих данных. Если места недостаточно, необходимо сохранить как минимум содержимое подкаталога pg_wal

каталога кластера, так как он может содержать журналы, не попавшие в архив перед остановкой системы.

3. Удалите все существующие файлы и подкаталоги из каталога кластера и из корневых каталогов используемых табличных пространств.

4. Восстановите файлы базы данных из резервной копии файлов. Важно, чтобы у восстановленных файлов были правильные разрешения и правильный владелец (пользователь, запускающий сервер, а не root!). Если вы используете табличные пространства, убедитесь также, что символьные ссылки в `pg_tblspc/` восстановились корректно.

5. Удалите все файлы из `pg_wal/;` они восстановились из резервной копии файлов и поэтому, скорее всего, будут старее текущих. Если вы вовсе не архивировали pg_wal/, создайте этот каталог с правильными правами доступа, но если это была символьная ссылка, восстановите её.

6. Если на шаге 2 вы сохранили незаархивированные файлы с сегментами WAL, скопируйте их в `pg_wal/`. (Лучше всего именно копировать, а не перемещать их, чтобы у вас остались неизменённые файлы на случай, если возникнет проблема и всё придётся начинать сначала.)

7. Установите параметры восстановления в `postgresql.conf` и создайте файл `recovery.signal` в каталоге данных кластера. Вы можете также временно изменить `pg_hba.conf`, чтобы обычные пользователи не могли подключиться, пока вы не будете уверены, что восстановление завершилось успешно.

8. Запустите сервер. Сервер запустится в режиме восстановления и начнёт считывать необходимые ему архивные файлы WAL. Если восстановление будет прервано из-за внешней ошибки, сервер можно просто перезапустить и он продолжит восстановление. По завершении процесса восстановления сервер удалит файл `recovery.signal` (чтобы предотвратить повторный запуск режима восстановления), а затем перейдёт к обычной работе с базой данных.

9. Просмотрите содержимое базы данных, чтобы убедиться, что вы вернули её к желаемому состоянию. Если это не так, вернитесь к шагу 1. Если всё хорошо, разрешите пользователям подключаться к серверу, восстановив обычный файл `pg_hba.conf`.

Ключевой момент этой процедуры заключается в создании конфигурации восстановления, описывающей, как будет выполняться восстановление и до какой точки. Единственное, что совершенно необходимо задать — это команду `restore_command`, которая говорит PostgreDB, как получать из архива файл-сегменты WAL. Как и `archive_command`, это командная строка для оболочки. Она может содержать символы %f, которые заменятся именем требующегося файла журнала, и %p, которые заменятся целевым путём для копирования этого файла. (Путь задаётся относительно текущего рабочего каталога, т. е. каталога кластера данных.) Если вам нужно включить в команду сам символ %, напишите %%. Простейшая команда, которая может быть полезна: **`restore_command = 'cp /mnt/server/archivedir/%f %p'`**

Эта команда копирует заархивированные ранее сегменты WAL из каталога `/mnt/server/archivedir`. Разумеется, вы можете использовать что-то более сложное, возможно, даже скрипт оболочки, который укажет оператору установить соответствующую ленту.

Важно, чтобы данная команда возвращала ненулевой код возврата в случае ошибки. Эта команда будет вызываться и с запросом файлов, отсутствующих в архиве; в этом случае она должна вернуть ненулевое значение и это считается штатной ситуацией. В исключительной ситуации, когда команда была прервана сигналом (кроме `SIGTERM`, который применяется в процессе остановки сервера базы данных) или произошла ошибка оболочки (например, команда не найдена), восстановление будет прервано и сервер не запустится.

Обычно при восстановлении обрабатываются все доступные сегменты WAL и, таким образом, база данных восстанавливается до последнего момента времени (или максимально близкого к нему, в зависимости от наличия сегментов WAL). Таким образом, восстановление обычно завершается с сообщением «файл не найден»; точный текст сообщения об ошибке зависит от того, что делает `restore_command`. Вы также можете увидеть сообщение об ошибке в начале восстановления для файла с именем типа

00000001.history. Это также нормально и обычно не говорит о какой-либо проблеме при восстановлении в простых ситуациях.

Если вы хотите восстановить базу на какой-то момент времени (скажем, до момента, когда неопытный администратор базы данных удалил основную таблицу транзакций), просто укажите требуемую точку остановки. Вы можете задать эту точку, иначе называемую «целью восстановления», по дате/времени, именованной точке восстановления или определённому идентификатору транзакции.

Если при восстановлении обнаруживаются повреждённые данные WAL, восстановление прерывается в этом месте и сервер не запускается. В этом случае процесс восстановления можно перезапустить с начала, указав «цель восстановления» до точки повреждения, чтобы восстановление могло завершиться нормально. Если восстановление завершается ошибкой из-за внешней причины, например, из-за краха системы или недоступности архива WAL, его можно просто перезапустить, и оно продолжится с того места, где было прервано. Перезапуск восстановления реализован по тому же принципу, что и контрольные точки при обычной работе: сервер периодически сохраняет всё текущее состояние на диске и отражает это в файле `pg_control`, чтобы уже обработанные данные WAL не приходилось сканировать снова.

На момент написания документации методика непрерывного архивирования имеет несколько ограничений:

- Если во время создания базовой резервной копии выполняется команда `CREATE DATABASE`, а затем база-шаблон, задействованная в `CREATE DATABASE`, изменяется, пока продолжается копирование, возможно, что при восстановлении эти изменения распространятся также и на созданную базу данных. Конечно, это нежелательно. Во избежание подобных рисков, лучше всего не изменять никакие базы-шаблоны во время получения базовой резервной копии.
- Команды `CREATE TABLESPACE` записываются в WAL с абсолютным путём и, таким образом, при воспроизведении WAL будут выполнены с тем же абсолютным путём. Это может быть нежелательно, если журнал воспроизводится на другой машине.

Но опасность есть, даже если журнал воспроизводится на той же машине, но в другом каталоге данных: при воспроизведении будет так же перезаписано содержимое исходных табличных пространств. Во избежание потенциальных проблем такого рода лучше всего делать новую базовую резервную копию после создания или удаления табличных пространств.

Также следует заметить, что стандартный формат WAL не очень компактный, так как включает много снимков дисковых страниц. Эти снимки страниц предназначены для поддержки восстановления после сбоя, на случай, если понадобится исправить страницы, записанные на диск частично. В зависимости от аппаратного и программного обеспечения вашей системы, риск частичной записи может быть достаточно мал, так что его можно игнорировать, и в этом случае можно существенно уменьшить общий объём архивируемых журналов, выключив снимки страниц с помощью параметра `full_page_writes`. Выключение снимков страниц не препятствует использованию журналов для восстановления PITR. Одним из направлений разработки в будущем является сжатие архивируемых данных WAL, путём удаления ненужных копий страниц даже при включённом режиме `full_page_writes`. Тем временем администраторы могут сократить количество снимков страниц, включаемых в WAL, увеличив параметры интервала контрольных точек в разумных пределах

3.5.6 Мониторинг работы СУБД

3.5.6.1 Сборщик статистики

Сборщик статистики в PostgreDB представляет собой подсистему, которая собирает и отображает информацию о работе сервера. В настоящее время сборщик может подсчитывать количество обращений к таблицам и индексам — в виде количества прочитанных блоков или строк с диска. Кроме того, он отслеживает общее число строк в каждой таблице, информацию о выполнении очистки и сбора статистики для каждой таблицы. Он также может подсчитывать вызовы пользовательских функций и общее время, затраченное на выполнение каждой из них.

Кроме того, PostgreDB может предоставить динамическую информацию о том, что происходит в системе прямо сейчас, в частности, сообщить, какие именно команды

выполняются другими серверными процессами и какие другие соединения существуют в системе. Эта возможность не зависит от процесса сборщика

3.5.6.2 Конфигурация системы сбора статистики

Поскольку сбор статистики несколько увеличивает накладные расходы при выполнении запроса, есть возможность настроить СУБД так, чтобы выполнять или не выполнять сбор статистической информации. Это контролируется конфигурационными параметрами, которые обычно устанавливаются в файле `postgresql.conf`.

Параметр `track_activities` включает мониторинг текущих команд, выполняемой любым серверным процессом.

Параметр `track_counts` определяет необходимость сбора статистики по обращениям к таблицам и индексам.

Параметр `track_functions` включает отслеживание использования пользовательских функций.

Параметр `track_io_timing` включает мониторинг времени чтения и записи блоков.

Параметр `track_wal_io_timing` включает мониторинг времени записи WAL.

Обычно эти параметры устанавливаются в `postgresql.conf`, поэтому они применяются ко всем серверным процессам, однако, используя команду SET, их можно включать и выключать в отдельных сессиях.

Сборщик статистики использует временные файлы для передачи собранной информации другим процессам PostgreDB. Имя каталога, в котором хранятся эти файлы, задаётся параметром `stats_temp_directory`, по умолчанию он называется `pg_stat_tmp`. Для повышения производительности `stats_temp_directory` может указывать на каталог, расположенный в оперативной памяти, что сокращает время физического ввода/вывода. При остановке сервера постоянная копия статистической информации сохраняется в подкаталоге `pg_stat`, поэтому статистику можно хранить на протяжении нескольких перезапусков сервера. Когда восстановление выполняется при запуске сервера (например, после непосредственного завершения работы, катастрофического отказа сервера, и восстановлении на заданную точку во времени), все статистические данные счётчиков сбрасываются.

3.5.6.3 Просмотр статистики

Для просмотра текущего состояния системы предназначены несколько предопределённых представлений.

Наблюдая собранные данные в сборщике статистики, важно понимать, что эта информация обновляется не сразу. Каждый серверный процесс передаёт новые статистические данные сборщику статистики непосредственно перед переходом в режим ожидания; то есть запрос или транзакция в процессе выполнения не влияют на отображаемые данные статистики. К тому же, сам сборщик статистики формирует новый отчёт не чаще, чем раз в `PGSTAT_STAT_INTERVAL` миллисекунд (500 мс, если этот параметр не изменялся при компиляции сервера). Так что отображаемая информация отстаёт от того, что происходит в настоящий момент. Однако информация о текущем запросе, собираемая с параметром `track_activities`, всегда актуальна.

Ещё одним важным моментом является то, что когда в серверном процессе запрашивают какую-либо статистику, сначала он получает наиболее свежий моментальный снимок от сборщика статистики и затем до окончания текущей транзакции использует этот снимок для всех статистических представлений и функций. Так что на протяжении одной транзакции статистическая информация меняться не будет. Подобным же образом информация о текущих запросах во всех сессиях собирается в тот момент, когда она впервые запрашивается в рамках транзакции, и эта же самая информация будет отображаться на протяжении всей транзакции. Это не ошибка, а полезное свойство СУБД, поскольку оно позволяет выполнять запросы к статистическим данным и сравнивать результаты, не беспокоясь о том, что статистические данные изменяются. Но если для каждого запроса вам нужны новые результаты, то их следует выполнять вне любых транзакционных блоков. Или же можно вызывать функцию `pg_stat_clear_snapshot()`, которая сбросит ранее полученный снимок статистики в текущей транзакции (если он был). При следующем обращении к статистической информации будет сформирован новый моментальный снимок.

Через представления `pg_stat_xact_all_tables`, `pg_stat_xact_sys_tables`, `pg_stat_xact_user_tables`, и `pg_stat_xact_user_functions` транзакции также доступна её собственная статистика (ещё не переданная сборщику статистики).

Данные в этих представлениях ведут себя не так, как описано выше; наоборот, в течение транзакции они постоянно обновляются.

Таблица 5 – Динамические статистические представления

Имя представления	Описание
pg_stat_activity	Одна строка для каждого серверного процесса с информацией о текущей активности процесса, включая его состояние и текущий запрос.
pg_stat_replication	По одной строке для каждого процесса-передатчика WAL со статистикой по репликации на ведомом сервере, к которому подключён этот процесс.
pg_stat_wal_receiver	Только одна строка со статистикой приёмника WAL, полученной с сервера, на котором работает приёмник.
pg_stat_subscription	Как минимум одна строка для подписки, сообщающая о рабочих процессах подписки.
pg_stat_ssl	Одна строка для каждого подключения (обычного и реплицирующего), в которой показывается информация об использовании SSL для данного подключения.
pg_stat_gssapi	Одна строка для каждого подключения (обычного и реплицирующего), в которой показывается информация об использовании аутентификации и шифровании GSSAPI для данного подключения.
pg_stat_progress_analyze	По одной строке с текущим состоянием для каждого обслуживающего процесса (включая рабочие процессы автоочистки), в котором работает ANALYZE.
pg_stat_progress_create_index	По одной строке с текущим состоянием для каждого обслуживающего процесса, в котором

	выполняется CREATE INDEX или REINDEX.
pg_stat_progress_vacuum	По одной строке с текущим состоянием для каждого обслуживающего процесса (включая рабочие процессы автоочистки), в котором работает VACUUM.
pg_stat_progress_cluster	По одной строке с текущим состоянием для каждого обслуживающего процесса, в котором выполняется CLUSTER или VACUUM FULL.
pg_stat_progress_basebackup	По одной строке с текущим состоянием для каждого передающего WAL процесса, транслирующего базовую копию.
pg_stat_progress_copy	По одной строке с текущим состоянием для каждого обслуживающего процесса, в котором выполняется COPY.

Таблица 6 – Представления собранной статистики

Имя представления	Описание
pg_stat_archiver	Только одна строка со статистикой работы процесса архивации WAL.
pg_stat_bgwriter	Только одна строка со статистикой работы фонового процесса записи.
pg_stat_wal	Только одна строка со статистикой работы WAL.
pg_stat_database	Одна строка для каждой базы данных со статистикой на уровне базы.
pg_stat_database_conflicts	По одной строке на каждую базу данных со статистикой по отменам запросов, выполненным вследствие конфликта с процессами восстановления на ведомых серверах.

pg_stat_all_tables	По одной строке на каждую таблицу в текущей базе данных со статистикой по обращениям к этой таблице.
pg_stat_sys_tables	Аналогично pg_stat_all_tables, за исключением того, что отображаются только системные таблицы.
pg_stat_user_tables	Аналогично pg_stat_all_tables, за исключением того, что отображаются только пользовательские таблицы.
pg_stat_xact_all_tables	Подобно pg_stat_all_tables, но подсчитывает действия, выполненные в текущей транзакции к настоящему моменту (которые ещё не вошли в pg_stat_all_tables и связанные представления). Столбцы для числа живых и мёртвых строк, а также количества операций очистки и сбора статистики, в этом представлении отсутствуют.
pg_stat_xact_sys_tables	Аналогично pg_stat_xact_all_tables, за исключением того, что отображаются только системные таблицы.
pg_stat_xact_user_tables	Аналогично pg_stat_xact_all_tables, за исключением того, что отображаются только пользовательские таблицы.
pg_stat_all_indexes	По одной строке для каждого индекса в текущей базе данных со статистикой по обращениям к этому индексу.
pg_stat_sys_indexes	Аналогично pg_stat_all_indexes, за исключением того, что показываются только индексы по системным таблицам.

pg_stat_user_indexes	Аналогично pg_stat_all_indexes, за исключением того, что показываются только индексы по пользовательским таблицам.
pg_statio_all_tables	По одной строке для каждой таблицы в текущей базе данных со статистикой по операциям ввода/вывода с этой таблицей.
pg_statio_sys_tables	Аналогично pg_statio_all_tables, за исключением того, что показываются только системные таблицы.
pg_statio_user_tables	Аналогично pg_statio_all_tables, за исключением того, что показываются только пользовательские таблицы.
pg_statio_all_indexes	По одной строке для каждого индекса в текущей базе данных со статистикой по операциям ввода/вывода для этого индекса.
pg_statio_sys_indexes	Аналогично pg_statio_all_indexes, за исключением того, что показываются только индексы по системным таблицам.
pg_statio_user_indexes	Аналогично pg_statio_all_indexes, за исключением того, что показываются только индексы по пользовательским таблицам.
pg_statio_all_sequences	По одной строке для каждой последовательности в текущей базе данных со статистикой по операциям ввода/вывода с этой последовательностью.
pg_statio_sys_sequences	Аналогично pg_statio_all_sequences, за исключением того, что показываются только системные последовательности. (В настоящее

	время системных последовательностей нет, поэтому это представление всегда пусто.)
pg_statio_user_sequences	Аналогично pg_statio_all_sequences, за исключением того, что показываются только пользовательские последовательности.
pg_stat_user_functions	По одной строке для каждой отслеживаемой функции со статистикой по выполнениям этой функции.
pg_stat_xact_user_functions	Аналогично pg_stat_user_functions, однако подсчитываются только вызовы функций, выполненные в текущей транзакции (которые ещё не были включены в pg_stat_user_functions).
pg_stat_slru	Одна строка со статистикой работы для каждого SLRU-кеша.
pg_stat_replication_slots	По одной строке со статистикой по использованию слота репликации для каждого такого слота.

Статистика по отдельным индексам особенно полезна для определения того, какие индексы используются и насколько они.

Представления pg_statio_ полезны, прежде всего, для определения эффективности буферного кеша. Если количество фактических дисковых чтений существенно меньше количества чтений из буферного кеша, то это означает, что кеш справляется с большинством запросов на чтение без обращения к ядру. Однако эта статистика не даёт полной картины: PostgreDB обрабатывает дисковый ввод/вывод так, что данные, не находящиеся в буферном кеше PostgreDB, могут все ещё располагаться в кеше ввода/вывода ядра, и, следовательно, для их получения физическое чтение может не использоваться. Для получения более детальной информации о процессе ввода/вывода в PostgreDB рекомендуется использовать сборщик статистики PostgreDB в сочетании с

утилитами операционной системы, которые дают более полное представление о том, как ядро осуществляет ввод/вывод

3.5.6.4 Просмотр информации о блокировках

Ещё одним удобным средством для отслеживания работы базы данных является системная таблица `pg_locks`. Она позволяет администратору базы просматривать информацию об имеющихся блокировках в менеджере блокировок. Например, это может использоваться для:

- просмотра всех имеющихся на данный момент блокировок, всех блокировок на отношения в определённой базе данных, всех блокировок на определённое отношение или всех блокировок, которые удерживает определённая сессия PostgreDB;
- определения отношения в текущей базе данных с наибольшим количеством неразрешённых блокировок (оно может быть причиной конкуренции между клиентами базы данных);
- определения воздействия конкуренции за блокировку на производительность базы данных в целом, а также то, как меняется конкуренция в зависимости от загруженности базы.

В PostgreDB имеется возможность отслеживать выполнение определённых команд. В настоящее время такое отслеживание поддерживается только для команд `ANALYZE`, `CLUSTER`, `CREATE INDEX`, `VACUUM`, `COPY` и `BASE_BACKUP` (то есть для команды репликации, которую выполняет `pg_basebackup`).

3.5.6.5 Отслеживание выполнения ANALYZE

Во время выполнения `ANALYZE` представление `pg_stat_progress_analyze` будет содержать по одной строке для каждого обслуживающего процесса, выполняющего эту команду.

Таблица 8 – Фазы ANALYZE

Фаза	Описание

initializing	Команда готовится начать сканирование кучи. Эта фаза должна быть очень быстрой.
acquiring sample rows	Команда сканирует таблицу с указанным relid, считывая строки выборки.
acquiring inherited sample rows	Команда сканирует дочерние таблицы, считывая строки выборки. Выполнение процедуры в этой фазе отражается в столбцах child_tables_total, child_tables_done и current_child_table_relid.
computing statistics	Команда вычисляет статистику по строкам выборки, полученным при сканировании таблицы.
computing extended statistics	Команда вычисляет расширенную статистику по строкам выборки, полученным при сканировании таблицы.
finalizing analyze	Команда вносит изменения в pg_class. После этой фазы ANALYZE завершит работу.

Когда ANALYZE обрабатывает секционированную таблицу, все её секции также рекурсивно анализируются. В этом случае сначала сообщается о ходе выполнения ANALYZE для родительской таблицы, которое сопровождается сбором наследуемой статистики, а затем о ходе обработки каждой её секции.

3.5.6.6 Отслеживание выполнения CREATE INDEX

Во время выполнения CREATE INDEX или REINDEX представление pg_stat_progress_create_index будет содержать по одной строке для каждого обслуживающего процесса, создающего индексы в этот момент.

Таблица 9 – Фазы CREATE INDEX

Фаза	Описание
	Инициализация — процедура CREATE

initializing	INDEX или REINDEX подготавливается к созданию индекса. Эта фаза должна быть очень быстрой.
waiting for writers before build	Ожидание окончания записи перед построением — процедура CREATE INDEX CONCURRENTLY или REINDEX CONCURRENTLY ожидает завершения транзакций, которые удерживают блокировки записи и могут читать таблицу. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе отражается в столбцах lockers_total, lockers_done и current_locker_pid.
building index	Построение индекса — код, реализующий метод доступа, строит индекс. В этой фазе методы доступа, поддерживающие отслеживание процесса, передают свои данные о текущем состоянии, и в этом столбце видна внутренняя фаза. Обычно ход построения индекса отражается в столбцах blocks_total и blocks_done, но также могут меняться и столбцы tuples_total и tuples_done.
waiting for writers before validation	Ожидание окончания записи перед проверкой — процедура CREATE INDEX CONCURRENTLY или REINDEX CONCURRENTLY ожидает завершения транзакций, которые удерживают блокировки записи и могут записывать в таблицу. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе отражается в столбцах lockers_total, lockers_done и current_locker_pid.
index validation: scanning index	Проверка индекса: сканирование — процедура CREATE INDEX CONCURRENTLY сканирует индекс, находя кортежи, требующие проверки. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе отражается в столбцах blocks_total (показывающем общий размер индекса) и blocks_done.
index	Проверка индекса: сортировка кортежей — процедура CREATE

validation: sorting tuples	INDEX CONCURRENTLY сортирует результат фазы сканирования индекса.
index validation: scanning table	Проверка индекса: сканирование таблицы — процедура CREATE INDEX CONCURRENTLY сканирует таблицу, чтобы проверить кортежи индекса, собранные в предыдущих двух фазах. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе отражается в столбцах blocks_total (показывающем общий размер таблицы) и blocks_done.
waiting for old snapshots	Ожидание старых снимков — процедура CREATE INDEX CONCURRENTLY или REINDEX CONCURRENTLY ожидает освобождения снимков теми транзакциями, которые могут видеть содержимое таблицы. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе отражается в столбцах lockers_total, lockers_done и current_locker_pid.
waiting for readers before marking dead	Ожидание завершения чтения перед отключением старого индекса — процедура REINDEX CONCURRENTLY ожидает завершения транзакций, которые удерживают блокировки чтения, прежде чем пометить старый индекс как нерабочий. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе отражается в столбцах lockers_total, lockers_done и current_locker_pid.
waiting for readers before dropping	Ожидание завершения чтения перед удалением старого индекса — процедура REINDEX CONCURRENTLY ожидает завершения транзакций, которые удерживают блокировки чтения, прежде чем удалить старый индекс. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой

	фазе отражается в столбцах lockers_total, lockers_done и current_locker_pid.

3.5.6.7 Отслеживание выполнения VACUUM

В процессе выполнения `VACUUM` представление `pg_stat_progress_vacuum` будет содержать по одной строке для каждого обслуживающего процесса (включая рабочие процессы автоочистки), производящего очистку в данный момент. Таблицы ниже показывают, какая информация будет отслеживаться, и поясняют, как её интерпретировать. Выполнение команд `VACUUM FULL` отслеживается через `pg_stat_progress_cluster`, так как и `VACUUM FULL`, и `CLUSTER` перезаписывают таблицу, тогда как обычная команда `VACUUM` модифицирует её саму.

Таблица 10 – Фазы VACUUM

Фаза	Описание
`initializing`	Инициализация — `VACUUM` готовится начать сканирование кучи. Эта фаза должна быть очень быстрой.
`scanning heap`	Сканирование кучи — `VACUUM` в настоящее время сканирует кучу. При этом будет очищена и, если требуется, дефрагментирована каждая страница, а возможно, также будет произведена заморозка. Отслеживать процесс сканирования можно, следя за содержимым столбца `heap_blks_scanned`.
`vacuuming indexes`	Очистка индексов — `VACUUM` в настоящее время очищает индексы. Если у таблицы есть какие-либо индексы, эта фаза будет наблюдаться минимум единожды в процессе очистки, после того, как куча будет просканирована полностью. Она может повторяться несколько раз в процессе очистки, если объёма maintenance_work_mem (или, в случае автоочистки, autovacuum_work_mem, если он задан) оказывается недостаточно для сохранения всех найденных «мёртвых» кортежей.

vacuuming heap	Очистка кучи — VACUUM в настоящее время очищает кучу. Очистка кучи отличается от сканирования, так как она происходит после каждой операции очистки индексов. Если heap_blks_scanned меньше чем heap_blks_total, система вернётся к сканированию кучи после завершения этой фазы; в противном случае она начнёт уборку индексов.
cleaning up indexes	Уборка индексов — VACUUM в настоящее время производит уборку в индексах. Это происходит после завершения полного сканирования кучи и очистки индексов и кучи.
truncating heap	Усечение кучи — VACUUM в настоящее время усекает кучу, чтобы возвратить операционной системе объём пустых страниц в конце отношения. Это происходит после уборки индексов.
performing final cleanup	Выполнение окончательной очистки — VACUUM выполняет окончательную очистку. На этой стадии VACUUM очищает карту свободного пространства, обновляет статистику в pg_class и передаёт статистику сборщику статистики, После этой фазы VACUUM завершит свою работу.

3.5.6.8 Отслеживание выполнения CLUSTER

Во время выполнения CLUSTER или VACUUM FULL представление pg_stat_progress_cluster будет содержать по одной строке для каждого обслуживающего процесса, выполняющего любую из этих команд. Таблицы ниже показывают, какая информация будет отслеживаться, и поясняют, как её интерпретировать.

Таблица 11 – Фазы CLUSTER и VACUUM FULL

Фаза	Описание
initializing	Команда готовится начать сканирование кучи. Эта фаза должна

	быть очень быстрой.
`seq scanning heap`	Команда в данный момент сканирует таблицу последовательным образом.
`index scanning heap`	CLUSTER в данный момент сканирует таблицу по индексу.
`sorting tuples`	CLUSTER в данный момент сортирует кортежи.
`writing new heap`	CLUSTER в данный момент записывает новую кучу.
`swapping relation files`	Команда в данный момент переставляет только что построенные файлы на место.
`rebuilding index`	Команда в данный момент перестраивает индекс.
`performing final cleanup`	Команда выполняет окончательную очистку. После этой фазы CLUSTER или VACUUM FULL завершит работу.

3.5.6.9 Отслеживание выполнения базового копирования

Когда приложение `pg_basebackup` или подобное выполняет базовое копирование, представление `pg_stat_progress_basebackup` будет содержать по одной строке для каждого процесса-передатчика WAL, выполняющего в данный момент команду репликации `BASE_BACKUP` и передающего копируемые данные. Таблицы ниже показывают, какая информация будет отслеживаться, и поясняют, как её интерпретировать.

Таблица 11 – Фазы базового копирования

Фаза	Описание
`initializing`	Процесс-передатчик WAL готовится начать копирование. Эта фаза должна быть очень быстрой.
`waiting for checkpoint to finish`	Процесс-передатчик WAL в настоящий момент выполняет pg_start_backup, чтобы подготовиться к получению базовой копии, и ждёт завершения контрольной точки для начала копирования.

`estimating backup size`	Процесс-передатчик WAL в настоящий момент оценивает общее количество файлов данных, которые будут передаваться при создании базовой копии.
`streaming database files`	Процесс-передатчик WAL в настоящий момент передаёт файлы данных в качестве содержимого базовой резервной копии.
`waiting for wal archiving to finish`	Процесс-передатчик WAL в настоящий момент выполняет pg_stop_backup, чтобы закончить копирование, и ждёт успешного завершения архивации всех файлов WAL, необходимых для базовой копии. Если при запуске pg_basebackup был указан параметр --wal-method=none или --wal-method=stream, резервное копирования заканчивается сразу после данной фазы.
`transferring wal files`	Процесс-передатчик WAL в настоящее время переносит все файлы WAL, заполненные во время копирования. Эта фаза следует за фазой waiting for wal archiving to finish, только если при запуске pg_basebackup указывался параметр --wal-method=fetch. По окончании этой фазы резервное копирование завершается.

3.5.6.10 Отслеживание выполнения COPY

Во время выполнения `COPY` представление `pg_stat_progress_copy` будет содержать по одной строке для каждого обслуживающего процесса, выполняющего эту команду.