УТВЕРЖДЕН
RU_ ПГ. 1002-0110

СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ "PG360"

Руководство оператора

RU_ ПГ. 1002-0110

Листов 195

Подп. и дата	
Инв.№ дубл.	
Взам. инв.№	
Подп. и дата	
Инв.№ подл.	

2025

Литера

АННОТАЦИЯ

Настоящий документ предоставляет сведения о назначении, условиях выполнения и порядке выполнения системы управления базами данных "PG360" в защищенном исполнении (далее – СУБД PG360), а также о сообщениях оператору (администратору СУБД PG360).

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ ПРОГРАММЫ

1.1. СУБД PG360 предназначена для сохранения, обработки и извлечения из базы данных предприятия той информации, к которой субъекту (пользователю, администратору или прикладной программе) предоставлен доступ, для последующего ее предоставления субъекту через интерфейс клиентского или серверного приложения (сервиса).

СУБД PG360является посредником между пользователем и базой данных, представляя конечному субъекту единое интегрированное представление данных в базе данных.

Функционально СУБД PG360обеспечивает управление созданием, обслуживанием баз данных и использованием информации баз данных. С ее помощью можно создавать, модифицировать или удалять записи, отправлять транзакцию – набор из нескольких последовательных запросов на языке запросов SQL.

1.2. Основные задачи, выполняемые СУБД PG360:

– гибкий доступ к базам данных, их организация и хранение;

– создавать и администрировать (удалять, изменять и объединять) базы данных;

– содержать данные в структурированном виде и необходимом формате;

– принимать подключения клиентских приложений пользователя;

– принимать подключения серверных приложений СУБД;

– выполнять различные запросы субъектов к базам данных;

– управлять файлами баз данных;

– защищать данные от нежелательных изменений и попыток взлома;

– загружать и сортировать данные с помощью фильтров;

– делать резервные копии, восстанавливать базы данных после сбоёв и поддерживать общую целостность.

1.3. СУБД PG360включает следующие взаимодействующие процессы (программы):

– postgres (программа сервера): главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запро сы клиентов к базам данных;

– клиентские приложения пользователя: процессы, запрашивающие выполнение операций в базе данных;

– серверные приложения СУБД: процессы, обеспечивающие дополнительные возможности по обслуживанию СУБД.

2. УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

2.1. Минимальный состав технических и программных средств

2.1.1. Минимальный состав технических средств:

– объем жесткого диска – не менее 5 ГБ;

– объем оперативной памяти – не менее 2 ГБ;

– процессор архитектуры – x86_64;

– тактовая частота процессора – не менее 1 ГГц.

2.1.2. Минимальный состав программных средств:

Для функционирования системы управления базами данных (далее – СУБД) PG360на сервере информационной системы, на котором устанавливается СУБД PG360, должна быть установлена операционная система (ОС) семейства Linux:

- Astra Linux 1.7

- AltLinux - Альт Сервер 10

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1. Общие положения

Администрирование сервера СУБД PG360(далее – сервера) состоит из следующих этапов:

1. Подготовка к работе и сопровождения сервера, включающий:
   – запуск СУБД PG360(п. 3.2.1.);
   – выключение СУБД PG360 (п. 3.2.2.);
   – управление ресурсами ядра (п. 3.2.2.);
   – возможности шифрования (п. 3.2.4.);
   – защита соединений TCP/IP (п. 3.2.5.).
2. Настройка сервера (п. 3.3.), включающего настройку параметров конфигурационных файлов для:
   – расположения файлов (п. 3.3.2.);
   – подключения и аутентификации (п. 3.3.3.);
   – потребления ресурсов (п. 3.3.4.);
   – журнала упреждающей записи (п. 3.3.5.);
   – репликации (п. 3.3.6.);
   – планирования запросов (п. 3.3.7.);
   – регистрации ошибок и протоколирования работы сервера (п. 3.3.8.);
   – статистики времени выполнения (п. 3.3.9.);
   – автоматической очистки (п. 3.3.10.);

– клиентских сеансов по умолчанию (п. 3.3.11.);
– управления блокировками (п. 3.3.12.);
– обработки ошибок (п. 3.3.13.);

3. Настройка аутентификации клиентского приложения (п. 3.4.);
4. Настройка ролей базы данных (п. 3.5.);
5. Управление базами данных (п. 3.6.);
6. Настройка локализации (п. 3.7.);
7. Регламентные задачи обслуживания базы данных (п. 3.8.);
8. Резервное копирование и восстановление (п. 3.9.);
9. Настройка отказоустойчивости, балансировки нагрузки и репликации (п. 3.10.);
10. Мониторинг работы СУБД (п. 3.11.);
11. Мониторинг использования диска (п. 3.12.);
12. Журнал упреждающей записи (WAL) и его настройка (п. 3.13.);
13. Настройка логической репликации (п. 3.14.).

3.2. Подготовка к работе и сопровождение сервера

3.2.1. Запуск СУБД PG360

Для запуска СУБД PG360необходимо выполнить команду:

```
systemctl start pg360
```

3.2.2. Выключение СУБД PG360

Для выключения СУБД PG360 необходимо выполнить команду:

```
systemctl stop pg360
```

3.2.3. Управление ресурсами ядра

При запуске нескольких копий сервера в одной ОС или при работе с очень большими базами данных возможно достичь исчерпания лимитов выделеных ресурсов ОС.

Далее описываются ресурсы ядра ОС, которые использует PG360, и подходы к решению проблем, связанных с ограниченностью этих ресурсов.

3.2.3.1. Разделяемая память и семафоры

PG360 требует, чтобы ОС предоставляла средства межпроцессного взаимодействия (IPC), в частности, разделяемую память и семафоры. Системы семейства Unix обычно предоставляют функции IPC в стиле «System V» или функции IPC в стиле «POSIX» или и те, и другие.

По умолчанию PG360запрашивает очень небольшой объём разделяемой памяти System V и намного больший объём анонимной разделяемой памяти mmap. Возможен также вариант использования одной большой области памяти System V. Помимо этого при запуске сервера создаётся значительное количество семафоров (в стиле System V или POSIX). В настоящее время семафоры POSIX используются в системах Linux, а на других платформах используются семафоры System V.

Функции IPC в стиле System V обычно сталкиваются с лимитами на уровне системы. Когда PG360 превышает один из этих лимитов, сервер отказывается запускаться, но должен выдать сообщение об ошибке и рекомендуемые действия. В таблице 1 приведены параметры IPC и их значения для запуска одного экземпляра сервера PG360(соответствующие параметры ядра в разных ОС называются аналогично).

Таблица 1. Параметры IPC в стиле System V

Имя	**Описание**	**Значения, необходимые для запуска одного экземпляра PG360**
SHMMAX	Максимальный размер сегмента разделяемой памяти (в байтах)	как минимум 1 КБ, но значение по умолчанию обычно гораздо больше
SHMMIN	Минимальный размер сегмента разделяемой памяти (в байтах)	1
SHMALL	Общий объём доступной разделяемой памяти (в байтах или страницах)	если в байтах, то же, что и SHMMAX; если в страницах, то ceil(SHMMAX/ PAGE_SIZE) , плюс потребность других приложений

Имя	**Описание**	**Значения, необходимые для запуска одного экземпляра PG360**
SHMSEG	Максимальное число сегментов разделяемой памяти для процесса	требуется только 1 сегмент, но значение по умолчанию гораздо больше
SHMMNI	Максимальное число сегментов разделяемой памяти для всей ОС	как SHMSEG плюс потребность других приложений
SEMMNI	Максимальное число идентификаторов семафоров (т. е., их наборов)	как минимум ceil((max_connections + autovacuum_max_workers + max_ wal_senders + max_worker_ processes + 5) / 16) плюс потребность других приложений
SEMMNS	Максимальное число семафоров для всей ОС	ceil((max_connections + autovacuum_max_workers + max_wal_ senders + max_worker_processes + 5) / 16) * 17 плюс потребность других приложений
SEMMSL	Максимальное число семафоров в наборе	не меньше 17
SEMMAP	Число записей в карте семафоров	см. текст ниже
SEMVMX	Максимальное значение семафора	не меньше 1000 (по умолчанию оно обычно равно 32767; без необходимости менять его не следует)

PG360запрашивает небольшой блок разделяемой памяти System V (обычно 48 байт на 64-битной платформе) для каждой копии сервера. Если запускать много копий сервера или явно настроить сервер для использования больших объёмов разделяемой памяти System V, то может понадобиться увеличить значение SHMALL, задающее общий объём разделяемой памяти System V, доступный для всей ОС. SHMALL во многих системах задаётся в страницах, а не в байтах.

Менее вероятны проблемы с минимальным размером сегментов разделяемой памяти (SHMMIN), который для PG360не должен превышать примерно 32 байт (обычно это всего 1 байт). Максимальное число сегментов для всей системы (SHMMNI) или для одного процесса (SHMSEG) обычно не влияет на работоспособность сервера, если только это число не равно нулю.

Когда PG360использует семафоры System V, он занимает по одному семафору на одно разрешённое подключение (max_connections), на разрешённый рабочий процесс автоочистки (autovacuum_max_workers) и фоновый процесс (max_worker_processes), в наборах по 16. В каждом таком наборе есть также 17-ый семафор, содержащий «магическое число», позволяющий обнаруживать коллизии с наборами семафоров других приложений. Максимальное число семафоров в системе задаётся параметром SEMMNS, который, следовательно, должен быть равен как минимум сумме max_connections, autovacuum_max_workers, max_wal_senders и max_worker_processes, плюс один дополнительный на каждые 16 семафоров подключений и рабочих процессов. Параметр SEMMNI определяет максимальное число наборов семафоров, которые могут существовать в системе в один момент времени. Таким образом, его значение должно быть не меньше чем ceil((max_connections + autovacuum_max_workers + max_wal_senders + max_worker_processes + 5) / 16). В качестве временного решения проблем, которые вызываются этими ограничениями, но обычно сопровождаются некорректными сообщениями функции semget, например, «No space left on device» (На устройстве не осталось места) можно уменьшить число разрешённых соединений.

В некоторых случаях может потребоваться увеличить SEMMAP как минимум до уровня SEMMNS. Если в системе есть такой параметр (а во многих системах его нет), он определяет

размер карты ресурсов семафоров, в которой выделяется запись для каждого непрерывного блока семафоров. Когда набор семафоров освобождается, эта запись либо добавляется к существующей соседней записи, либо регистрируется как новая запись в карте. Если карта переполняется, освобождаемые семафоры теряются (до перезагрузки). Таким образом, фрагментация пространства семафоров может со временем привести к уменьшению числа доступных семафоров.

Другие параметры, связанные с «аннулированием операций» с семафорами, например, SEMMNU и SEMUME, на работу PG360не влияют.

При использовании семафоров POSIX требуемое их количество не отличается от количества для System V, то есть по одному семафору на разрешённое подключение (max_connections), на разрешённый рабочий процесс автоочистки (autovacuum_max_workers) и фоновый процесс (max_worker_processes). На платформах, где предпочитается этот вариант, отсутствует определённый лимит ядра на количество семафоров POSIX.

В ОС Linux параметры разделяемой памяти по умолчанию вполне приемлемы, если не выбран в параметре shared_memory_type вариант sysv. И даже в этом случае их потребуется увеличить только для старых ядер, в которых эти параметры по умолчанию имеют маленькие значения.

Параметры разделяемой памяти можно изменить, воспользовавшись командой sysctl.

Чтобы сохранить эти изменения после перезагрузки, необходимо воспользоваться файлом /etc/ sysctl.conf.

3.2.3.2. RemoveIPC в systemd

При использовании systemd, необходимо позаботиться о том, чтобы ресурсы IPC (включая разделяемую память) не освобождались преждевременно ОС. Пользователей дистрибутивных пакетов PG360 это касается в меньшей степени, так как пользователь postgres обычно создаётся как системный пользователь.

Параметр RemoveIPC в logind.conf определяет, должны ли объекты IPC удаляться при полном выходе пользователя из системы. На системных пользователей это не распространяется. При включении параметра RemoveIPC объекты разделяемой памяти, используемые для параллельного выполнения запросов, удаляются без видимых причин, что приводит к появлению ошибок и предупреждений при попытке открыть и удалить их. Например:

```
WARNING: could not remove shared memory segment "/PostgreSQL.1450751626": No such file or directory
ПРЕДУПРЕЖДЕНИЕ: ошибка при удалении сегмента разделяемой памяти "/PostgreSQL.1450751626": Нет такого файла или каталога)
```

Событие «выхода пользователя из системы» может произойти при выполнении задачи обслуживания или если администратор войдёт под именем postgres, а затем выйдет, либо случится что-то подобное, так что предотвратить это довольно сложно.

Если же учётная запись пользователя была создана некорректно и изменить её невозможно, рекомендуется задать:

RemoveIPC=no

в /etc/systemd/logind.conf или другом подходящем файле конфигурации.

3.2.3.3. Ограничения ресурсов

3.2.3.3.1. В Unix-подобных ОС существуют различные типы ограничений ресурсов, которые могут влиять на работу сервера PG360.

Для PG360интерес представляют параметры:

– maxproc – ограничения на число процессов для пользователя;

– openfiles – ограничения на число открытых файлов;

– datasize – ограничения на объём памяти для каждого процесса.

Каждое из этих ограничений имеет «жёсткий» и «мягкий» предел. Мягкий предел действительно ограничивает использование ресурса, но пользователь может увеличить его значение до жёсткого предела. Изменить жёсткий предел может только пользователь root. За изменение этих параметров отвечает системный вызов setrlimit. Управлять этими ресурсами в командной строке позволяет встроенная команда ulimit (в оболочках Bourne) и limit (csh).

Пример задания параметров ограничений:

```
default:\
...      :datasize-cur=256M:\
         :maxproc-cur=256:\
         :openfiles-cur=256:\
```

(где, cur обозначает мягкий предел. Чтобы задать жёсткий предел, нужно заменить это окончание на -max.)

3.2.3.3.2 Ядро также может устанавливать общесистемные ограничения на использование некоторых ресурсов:

– максимальное число открытых файлов в Linux, определяется параметром ядра fs.file-max. Изменить этот предел можно, воспользовавшись командой sysctl -w fs.file-max=*N*. Чтобы эти изменения сохранялись после перезагрузки, следует добавить присваивание в файл /etc/sysctl.conf;

– максимальное число открытых файлов сервером PG360, установив параметр конфигурации max_files_per_process;

– максимальная длина очереди подключений к сокету (когда устанавливается большое количество клиентских подключений к серверу). Если количество запросов на подключение за короткий промежуток времени превышает этот максимум, некоторые из них будут отклонены до того, как главный процесс сможет их обработать, при этом клиенты получат неинформативное сообщение об ошибке подключения типа «Resource temporarily unavailable» (Ресурс временно недоступен) или «Connection refused» (Не удалось подключиться). Предел длины очереди на многих платформах по умолчанию составляет 128. Чтобы увеличить его, необходимо настройть соответствующий параметр ядра через sysctl и перезапустить главный процесс. Этот параметр называется net.core.somaxconn в Linux.

Сервер PG360 использует для обслуживания каждого подключения отдельный процесс, поэтому возможное число процессов должно быть не меньше числа разрешённых соединений плюс число процессов, требуемых для остальной системы.

3.2.3.4. Чрезмерное выделение памяти в Linux

В Linux механизм виртуальной памяти по умолчанию работает не оптимально для PG360. Вследствие того, что ядро выделяет память в чрезмерном объёме, оно может

уничтожить главный управляющий процесс PG360 (postmaster), если при выделении памяти процессу PG360 или другому процессу виртуальная память будет исчерпана.

Когда это происходит, можно получить примерно такое сообщение ядра:

```
Out of Memory: Killed process 12345 (postgres).
```

Это сообщение говорит о том, что процесс postgres был уничтожен из-за нехватки памяти. Хотя существующие подключения к базе данных будут работать по-прежнему, новые подключения приниматься не будут. Чтобы восстановить работу сервера PG360, его необходимо перезапустить.

Один из способов обойти эту проблему — запускать PG360 на выделенном сервере, где никакие другие процессы не займут всю память. Если физической памяти недостаточно, решить проблему также можно, увеличив объём пространства подкачки, так как уничтожение процессов при нехватке памяти происходит только когда заканчивается и физическая память, и место в пространстве подкачки.

Если памяти не хватает по вине самого PG360, эту проблему можно решить, изменив конфигурацию сервера. В некоторых случаях может помочь уменьшение конфигурационных параметров, связанных с памятью, а именно shared_buffers, work_mem и hash_mem_multiplier. В других случаях проблема может возникать, потому что разрешено слишком много подключений к самому серверу баз данных. Чаще всего в такой ситуации необходимо число подключений max_connections и организовать внешний пул соединений.

«Чрезмерное выделение» памяти можно предотвратить, изменив поведение ядра одним из следующих способов:

– включить режим строгого выделения памяти, воспользовавшись sysctl:

```
sysctl -w vm.overcommit_memory=2
```

либо поместив соответствующую запись в /etc/sysctl.conf. Хотя при этом механизм *OOM killer* (принудительное уничтожение процессов при нехватке памяти) всё равно может вызываться, вероятность такого уничтожения значительно уменьшится;

– исключить процесс postmaster из числа возможных жертв при нехватке памяти (возможно, вместе с изменением параметра vm.overcommit_memory). Для этого нужно изменить значение приоритета для этого процесса на -1000:

```
echo -1000 > /proc/self/oom_score_adj
```

в скрипте запуска управляющего процесса непосредственно перед тем, как запускать postmaster. Необхдимо добавить эту команду в стартовый скрипт, принадлежащий пользователю root. При этом также необходимо установить в данном скрипте переменные окружения перед запуском главного процесса:

```
export PG_OOM_ADJUST_FILE=/proc/self/oom_score_adj
export PG_OOM_ADJUST_VALUE=0
```

С такими параметрами дочерние процессы главного процесса сервера будут запускаться с обычной, нулевой поправкой очков OOM, так что при необходимости механизм OOM сможет уничтожать их. Можно задать и другое значение для PG_OOM_ADJUST_VALUE, чтобы дочерние процессы исполнялись с другой поправкой OOM. (PG_OOM_ADJUST_VALUE также можно опустить, в этом случае подразумевается нулевое значение.) Если не установить PG_OOM_ADJUST_FILE, дочерние процессы будут работать с той же поправкой очков OOM что и

главный процесс. Необходимо внести изменения, чтобы главный процесс оказался на особом положении.

3.2.3.5. Огромные страницы в Linux

Использование огромных страниц (huge pages) снижает накладные расходы при работе с большими непрерывными блоками памяти, что характерно для PG360, особенно при большом объёме shared_buffers. Также понадобится настроить параметр ядра vm.nr_hugepages. Чтобы оценить требуемое количество огромных страниц, необходимо запустить PG360 без поддержки огромных страниц и определить размер сегмента анонимной разделяемой памяти процесса postmaster, а также узнать размер огромной страницы, воспользовавшись файловой системой /proc. Например:

```
$ head -1 $PGDATA/postmaster.pid
4170
$ pmap 4170 | awk '/rw-s/ && /zero/ {print $2}'
6490428K
$ grep ^Hugepagesize /proc/meminfo
Hugepagesize:  2048 kB
```

В данном случае 6490428 / 2048 даёт примерно 3169.154, так что потребуется минимум 3170 огромных страниц, можно задать это значение следующим образом:

```
$ sysctl -w vm.nr_hugepages=3170
```

Большее значение стоит указать, если огромные страницы будут использоваться и другими программами в этой системе. Необходимо добавить этот параметр в /etc/sysctl.conf, чтобы он действовал и после перезагрузки.

Иногда ядро не может выделить запрошенное количество огромных страниц сразу, поэтому может потребоваться повторить эту команду или перезагрузить систему. (Немедленно после перезагрузки должен быть свободен больший объём памяти для преобразования в огромные страницы.) Чтобы проверить текущую ситуацию с размещением огромных страниц, необходимо выполнить команду:

```
$ grep Huge /proc/meminfo
```

Также может потребоваться дать пользователю ОС, запускающему сервер БД, право использовать огромные страницы, установив его группу в vm.hugetlb_shm_group с помощью sysctl, и/или разрешить блокировать память, выполнив ulimit -l.

По умолчанию PG360 использует огромные страницы, когда считает это возможным, а в противном случае переходит к обычным страницам. Чтобы задействовать огромные страницы принудительно, необходимо установить в файле postgresql.conf для параметра huge_pages значение on. Тогда PG360 не сможет запуститься, если не получит достаточного количества огромных страниц.

3.2.4. Возможности шифрования

PG360 обеспечивает шифрование на разных уровнях и дает гибкость в выборе средств защиты данных:

– 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, он более безопасен, чем собственный протокол проверки MD5 в PG360.

– Шифрование избранных столбцов. Модуль pgcrypto позволяет хранить в зашифрованном виде избранные поля. Это полезно, если ценность представляют только некоторые данные. Чтобы прочитать эти поля, клиент передаёт дешифрующий ключ, сервер расшифровывает данные и выдаёт их клиенту. Расшифрованные данные и ключ дешифрования находятся на сервере в процессе расшифровывания и передачи данных.

– Шифрование раздела данных. Шифрование хранилища данных можно реализовать на уровне файловой системы или на уровне блоков. Этот механизм не позволяет читать незашифрованные данные с дисков в случае кражи дисков или всего компьютера. При этом он не защищает данные от чтения, когда эта файловая система смонтирована, так как на смонтированном устройстве ОС видит все данные в незашифрованном виде. Однако, чтобы смонтировать файловую систему, нужно передать ОС ключ.

– Шифрование на стороне клиента. Если системный администратор сервера, где работает база данных, не является доверенным, клиент должен сам шифровать данные; тогда незашифрованные данные никогда не появятся на этом сервере. В этом случае клиент шифрует данные, прежде чем передавать их серверу, а получив из базы данных результаты, он расшифровывает их для использования.

3.2.5. Защита соединений TCP/IP

Для обеспечения доверенной связи между сервером (программа postgres) и субъектами (пользователями, приложениями) в PG360 допускает применение ПО других продуктов среды эксплуатации, при условии использования продуктов прошедших сертификацию или экспертизу в Национальной системе оценки соответствия, или, если разрешено законодательно, встроенных в ОС продуктов (например, SSL, SSH).

Для запуска сервера PG360 с включённым механизмом SSL, необходимо задать в файле postgresql.conf для параметра значение «on» и настроить другие конфигурационные параметры SSL (см. п. 3.3.3).

3.3. Настройка СУБД PG360

На работу СУБД PG360 оказывают влияние множество параметров конфигурации.

3.3.1. Изменение параметров

3.3.1.1. 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 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. 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.

– *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.

3.3.1.2. Определение параметров в файле конфигурации

Значения параметров конфигурации указываются в файле postgresql.conf, который обычно находится в каталоге данных. При инициализации каталога кластера БД в этот каталог помещается копия стандартного файла.

Каждый параметр определяется в отдельной строке. Знак равенства в ней между именем и значением является необязательным. Пробельные символы в строке не играют роли (кроме значений, заключённых в апострофы), а пустые строки игнорируются. Знаки решётки (#) обозначают продолжение строки как комментарий. Значения параметров, не являющиеся простыми идентификаторами или числами, должны заключаться в апострофы. Чтобы включить в такое значение собственно апостроф, его следует продублировать (предпочтительнее) или предварить обратной косой чертой. Если один и тот же параметр определяется в файле конфигурации неоднократно, действовать будет только последнее определение, остальные игнорируются.

Параметры, установленные таким образом, задают значения по умолчанию для данного кластера. Эти значения будут действовать в активных сеансах, если не будут переопределены.

Основной процесс сервера перечитывает файл конфигурации заново, получая сигнал SIGHUP. Послать сигнал 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. Учтите, что 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 (об этом сообщается в журнале).

В дополнение к файлу postgresql.conf в каталоге данных PG360 содержится файл postgresql.auto.conf, который имеет тот же формат, что и файле postgresql.conf, но предназначен для автоматического изменения, а не для редактирования вручную. Этот файл содержит параметры, задаваемые командой ALTER SYSTEM. Он считывается одновременно с файлом postgresql.conf и заданные в нём параметры действуют таким же образом. Параметры в файле postgresql.auto.conf переопределяют те, что указаны в postgresql.conf.

Вносить изменения в файле 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.

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.

3.3.1.3. Управление параметрами через SQL

В PG360 есть три SQL-команды, задающие для параметров значения по умолчанию:

– команда ALTER SYSTEM даёт возможность изменять глобальные значения средствами SQL, она функционально равнозначна редактированию файла postgresql.conf;

– команда ALTER DATABASE позволяет переопределить глобальные параметры на уровне БД;

– команда ALTER ROLE позволяет переопределить для конкретного пользователя как глобальные, так и локальные для БД параметры.

Значения, установленные командами ALTER DATABASE и ALTER ROLE, применяются только при новом подключении к БД. Они переопределяют значения, полученные из файлов конфигурации или командной строки сервера, и применяются по умолчанию в рамках сеанса. Некоторые параметры невозможно изменить после запуска сервера, поэтому их нельзя установить этими командами.

Когда клиент подключён к БД, он может воспользоваться двумя дополнительными командами SQL (и равнозначными функциями), которые предоставляет PG360 для управления параметрами конфигурации:

– команда SHOW позволяет узнать текущее значение любого параметра;

– команда SET позволяет изменить текущее значение тех параметров, которые устанавливаются локально в рамках сеанса; на другие сеансы она не влияет.

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.

3.3.1.4. Управление параметрами в командной строке

Помимо изменения глобальных значений по умолчанию и переопределения их на уровне БД или роли, параметры PG360 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.

Например:

```
postgres -c log_connections=yes -c log_destination='syslog'
```

Параметры, заданные таким образом, переопределяют те, что были установлены в файле postgresql.conf или командой ALTER SYSTEM, так что их нельзя изменить глобально без перезапуска сервера.

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. По историческим причинам формат PGOPTIONS похож на тот, что применяется при запуске команды postgres; в частности, в нём должен присутствовать флаг -c.

3.3.1.5. Упорядочение содержимого файлов конфигурации

PG360 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, файл 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 следующим образом:

```
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 следующим образом:

```
include_dir 'каталог'
```

Имена, заданные не абсолютным путём, рассматриваются относительно каталога, содержащего текущий файл конфигурации. В заданном каталоге включению подлежат только файлы с именами, оканчивающимися на .conf.

При этом файлы с именами, начинающимися с «.», тоже игнорируются, для предотвращения ошибок, так как они считаются скрытыми в ряде систем.

Набор файлов во включаемом каталоге обрабатывается по порядку имён (определяемому правилами, принятыми в C, то есть цифры идут перед буквами, а буквы в верхнем регистре – перед буквами в нижнем).

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

3.3.2. Расположения файлов

В дополнение к файлу конфигурации postgresql.conf, PG360 обрабатывает два редактируемых вручную файла конфигурации, в которых настраивается аутентификация клиентов. По умолчанию все три файла конфигурации размещаются в каталоге данных кластера БД.

При необходимости разместить файлы конфигурации в других каталогах, необходимо установить следующие параметры:

– 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.

– 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.

Если необходимо 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.

При необходимости можно задать имена и пути файлов конфигурации по отдельности, воспользовавшись параметрами config_file, hba_file. Параметр config_file можно задать только в командной строке postgres, тогда как остальные можно задать и в основном файле конфигурации. Если явно заданы все три эти параметра плюс data_directory, то задавать -D или PGDATA не нужно.

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

3.3.3. Подключения и аутентификация

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». Хотя механизм аутентификации клиентов позволяет гибко управлять доступом пользователей к серверу, параметр listen_addresses может ограничить интерфейсы, через которые будут приниматься соединения, что бывает полезно для предотвращения злонамеренных попыток подключения через незащищённые сетевые интерфейсы. Этот параметр можно задать только при запуске сервера.

– port (integer) – TCP-порт, открываемый сервером; по умолчанию, 5432. Этот порт 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.

– superuser_reserved_connections (integer) – Определяет количество «слотов» подключений, которые PG360 будет резервировать для суперпользователей. При этом всего одновременно активными могут быть максимум max_connections подключений. Когда число активных одновременных подключений больше или равно max_connections минус superuser_reserved_connections, принимаются только подключения суперпользователей, а все другие подключения, в том числе подключения для репликации, запрещаются. По умолчанию резервируются три соединения. Это значение должно быть меньше значения max_connections. Задать этот параметр можно только при запуске сервера.

– unix_socket_directories (string) – Задаёт каталог Unix-сокета, через который сервер будет принимать подключения клиентских приложений. Создать несколько сокетов можно, перечислив в этом значении несколько каталогов через запятую. Пробелы между элементами этого списка игнорируются; если в пути каталога содержатся пробелы, его нужно заключать в двойные кавычки. При пустом значении сервер не будет работать с Unix-сокетами, в этом случае к нему можно будет подключиться только по TCP/IP. Значение по умолчанию обычно /tmp, но его можно изменить во время сборки. Задать этот параметр можно только при запуске сервера.

Помимо самого файла сокета, который называется .s.PGSQL.*nnnn* (где *nnnn* – номер порта сервера), в каждом каталоге unix_socket_directories создаётся обычный файл .s.PGSQL.*nnnn*.lock. Ни в коем случае нельзя удалять эти файлы вручную.

– unix_socket_group (string) – Задаёт группу-владельца Unix-сокетов. (Пользователем-владельцем сокетов всегда будет пользователь, запускающий сервер.) В сочетании с unix_socket_permissions данный параметр можно использовать как дополнительный механизм управления доступом к Unix-сокетам. По умолчанию он содержит пустую строку, то есть группой-владельцем становится основная группа пользователя, запускающего сервер. Задать этот параметр можно только при запуске сервера.

– unix_socket_permissions (integer) – Задаёт права доступа к Unix-сокетам. Для Unix-сокетов применяется обычный набор разрешений Unix. Значение параметра ожидается в числовом виде, который принимают функции chmod и umask. (Для применения обычного восьмеричного формата

число должно начинаться с 0 (нуля).) По умолчанию действуют разрешения 0777, при которых подключаться к сокету могут все. Другие разумные варианты – 0770 (доступ имеет только пользователь и группа) и 0700 (только пользователь). (Для Unix-сокетов требуется только право на запись.) Этот параметр можно задать только при запуске сервера.

– bonjour (boolean) – Включает объявления о существовании сервера посредством Bonjour. По умолчанию выключен. Задать этот параметр можно только при запуске сервера.

– bonjour_name (string) – Задаёт имя службы в среде Bonjour. Если значение этого параметра – пустая строка ('') (это значение по умолчанию), в качестве этого имени используется имя компьютера. Этот параметр игнорируется, если сервер был скомпилирован без поддержки Bonjour. Задать этот параметр можно только при запуске сервера.

– tcp_keepalives_idle (integer) – Задаёт период отсутствия сетевой активности, по истечении которого ОС должна отправить клиенту TCP-сигнал сохранения соединения. Если это значение задаётся без единиц измерения, оно считается заданным в секундах. При значении 0 (по умолчанию) действует величина, принятая по умолчанию в ОС. Этот параметр поддерживается только в системах, воспринимающих параметр сокета TCP_KEEPIDLE или равнозначный; в других системах он должен быть равен нулю. В сеансах, подключённых через Unix-сокеты, он игнорируется и всегда считается равным 0.

– tcp_keepalives_interval (integer) – Задаёт интервал, по истечении которого следует повторять TCP-сигнал сохранения соединения, если от клиента не получено подтверждение предыдущего сигнала. Если это значение задаётся без единиц измерения, оно считается заданным в секундах. При значении 0 (по умолчанию) действует величина, принятая по умолчанию в ОС. Этот параметр поддерживается только в системах, воспринимающих параметр сокета TCP_KEEPINTVL или равнозначный; в других системах он должен быть равен нулю. В сеансах, подключённых через Unix-сокеты, он игнорируется и всегда считается равным 0.

– tcp_keepalives_count (integer) – Задаёт число TCP-сигналов сохранения соединения, которые могут быть потеряны до того, как соединение сервера с клиентом будет признано прерванным. При значении 0 (по умолчанию) действует величина, принятая по умолчанию в ОС. Этот параметр поддерживается только в системах, воспринимающих параметр сокета TCP_KEEPCNT или равнозначный; в других системах он должен быть равен нулю. В сеансах, подключённых через Unix-сокеты, он игнорируется и всегда считается равным 0.

– tcp_user_timeout (integer) – Задаёт интервал, в течение которого переданные данные могут оставаться неподтверждёнными, прежде чем будет принято решение о принудительном закрытии TCP-соединения. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. При значении 0 (по умолчанию) действует величина, принятая по умолчанию в ОС. Этот параметр поддерживается только в системах, воспринимающих параметр сокета TCP_USER_TIMEOUT; в других системах он должен быть равен нулю. В сеансах, подключённых через доменные сокеты Unix, он игнорируется и всегда считается равным 0.

Параметры аутентификации:

– authentication_timeout (integer) – Максимальное время, за которое должна произойти аутентификация. Если потенциальный клиент не сможет пройти проверку подлинности за это время, сервер закроет соединение. Благодаря этому зависшие при подключении клиенты не будут занимать соединения неограниченно долго. Если это значение задаётся без единиц измерения, оно

считается заданным в миллисекундах. Значение этого параметра по умолчанию – одна минута (1m). Задать этот параметр можно только в файле postgresql.conf или в командной строке при запуске сервера.

– password_encryption (enum) – Когда в CREATE ROLE или ALTER ROLE задаётся пароль, этот параметр определяет, каким алгоритмом его шифровать. Значение по умолчанию – md5 (пароль сохраняется в виде хеша MD5), также в качестве псевдонима md5 принимается значение on.

– db_user_namespace (boolean) – Этот параметр позволяет относить имена пользователей к БД. По умолчанию он имеет значение off (выключен). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

Если он включён, имена создаваемых пользователей должны иметь вид *имя_пользователя@база_данных*. Когда подключающийся клиент передаёт *имя_пользователя*, к этому имени добавляется @ с именем БД, и сервер идентифицирует пользователя по этому полному имени. Для создания пользователя с именем, содержащим @, в среде SQL потребуется заключить это имя в кавычки.

Когда этот параметр включён, он не мешает создавать и использовать обычных глобальных пользователей. Чтобы подключиться с таким именем пользователя, достаточно добавить к имени @, например так: joe@. Получив такое имя, сервер отбросит @, и будет идентифицировать пользователя по начальному имени.

Параметр db_user_namespace порождает расхождение между именами пользователей на стороне сервера и клиента. Но проверки подлинности всегда выполняются с именем с точки зрения сервера, так что, настраивая аутентификацию, следует указывать серверное представление имени, а не клиентское. Так как метод аутентификации md5 подмешивает имя пользователя в качестве соли и на стороне сервера, и на стороне клиента, при включённом параметре db_user_namespace использовать md5 невозможно.

Параметры SSL:

– ssl (boolean) – Разрешает подключения SSL. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. Значение по умолчанию – off.

– ssl_ca_file (string) – Задаёт имя файла, содержащего сертификаты удовстоверяющих центров (УЦ) для SSL-сервера. При указании относительного пути он рассматривается от каталога данных. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. По умолчанию этот параметр пуст, что означает, что файл сертификатов УЦ не загружается и проверка клиентских сертификатов не производится.

– ssl_cert_file (string) – Задаёт имя файла, содержащего сертификат этого SSL-сервера. При указании относительного пути он рассматривается от каталога данных. Этот параметр можно

задать только в postgresql.conf или в командной строке при запуске сервера. Значение по умолчанию – server.crt.

– ssl_crl_file (string) – Задаёт имя файла, содержащего список отзыва клиентских сертификатов (CRL, Certificate Revocation List) для SSL. При указании относительного пути он рассматривается от каталога данных. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. По умолчанию этот параметр пуст, что означает, что файл CRL не загружается.

– ssl_key_file (string) – Задаёт имя файла, содержащего закрытый ключ SSL-сервера. При указании относительного пути он рассматривается от каталога данных. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. Значение по умолчанию – server.key.

– ssl_ciphers (string) – Задаёт список наборов шифронаборов SSL, которые могут применяться для SSL-соединений. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Значение по умолчанию – HIGH:MEDIUM:+3DES:!aNULL, где

HIGH – Наборы шифров, в которых используются шифры из группы высокого уровня (HIGH);

MEDIUM – Наборы шифров, в которых используются шифры из группы среднего уровня (MEDIUM);

– ssl_prefer_server_ciphers (boolean) – Определяет, должны ли шифры SSL сервера предпочитаться клиентским. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. Значение по умолчанию – on.

– ssl_ecdh_curve (string) – Задаёт имя кривой для использования при обмене ключами ECDH. Эту кривую должны поддерживать все подключающиеся клиенты. Это не обязательно должна быть кривая, с которой был получен ключ сервера. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. Значение по умолчанию – prime256v1.

Полный список доступных кривых можно получить командой openssl ecparam -list_curves. Однако не все из них пригодны для TLS.

– ssl_min_protocol_version (enum) – Задаёт минимальную версию протокола SSL/TLS, которая может использоваться.

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

– ssl_max_protocol_version (enum) – Задаёт максимальную версию протокола SSL/TLS, которая может использоваться. Допускаются те же версии, что и для параметра ssl_min_protocol_version, а также пустая строка, обозначающая отсутствие ограничения версии. По умолчанию версии не ограничиваются. Устанавливать максимальную возможную версию протокола полезно прежде всего для тестирования или в случае проблем при использовании нового протокола какими-то компонентами.

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

– ssl_dh_params_file (string) – Задаёт имя файла с параметрами алгоритма Диффи-Хеллмана. По умолчанию значение пустое, то есть используются стандартные параметры DH, заданные при

компиляции. Использование нестандартных параметров DH защищает от атаки, рассчитанной на взлом хорошо известных встроенных параметров DH. Создать собственный файл с параметрами DH можно, выполнив команду openssl dhparam -out dhparams.pem 2048.

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

– ssl_passphrase_command (string) – Задаёт внешнюю команду, которая будет вызываться, когда потребуется пароль для расшифровывания SSL-файла, например закрытого ключа.

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

– ssl_passphrase_command_supports_reload (boolean) – Этот параметр определяет, будет ли заданная параметром ssl_passphrase_command команда, запрашивающая пароль, также вызываться при перезагрузке конфигурации, если для файла ключа требуется пароль. Когда этот параметр отключён (по умолчанию), команда ssl_passphrase_command будет игнорироваться при перезагрузке, и конфигурация SSL не будет обновляться, если требуется пароль. Это значение подходит для команды, требующей для ввода пароля наличия терминала TTY, который может быть недоступен в процессе работы сервера. Включение данного параметра может быть уместно, если, например, пароль считывается из файла.

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

3.3.4. Потребление ресурсов

Параметры для настройки памяти:

– shared_buffers (integer) – Задаёт объём памяти, который будет использовать сервер БД 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 1 ГБ и более, разумным начальным значением shared_buffers будет 25% от объёма памяти. Существуют варианты нагрузки, при которых эффективны будут и ещё большие значения shared_buffers, но так как PG360 использует и кеш ОС, выделять для shared_buffers более 40% ОЗУ вряд ли будет полезно. При увеличении shared_buffers обычно требуется соответственно увеличить max_wal_size, чтобы растянуть процесс записи большого объёма новых или изменённых данных на более продолжительное время.

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

– huge_pages (enum) – Определяет, будут ли огромные страницы запрашиваться из основной области общей памяти. Допустимые значения: try (по умолчанию), on и off. Когда параметр huge_pages равен try, сервер будет пытаться запрашивать огромные страницы, но если это ему не удастся, вернётся к стандартному поведению. Со значением on, если получить

огромные страницы не удастся, сервер не будет запущен. Со значением off большие страницы не будут запрашиваться.

Эта поддержка обеспечивается, только когда параметр shared_memory_type имеет значение mmap (по умолчанию).

В результате использования огромных страниц уменьшаются таблицы страниц, и процессор тратит меньше времени на управление памятью, что приводит к увеличению быстродействия.

– temp_buffers (integer) – Задаёт максимальный объём памяти, выделяемой для временных буферов в каждом сеансе. Эти существующие только в рамках сеанса буферы используются исключительно для работы с временными таблицами. Если это значение задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Значение по умолчанию – 8 мегабайт (8MB). (Если BLCKSZ отличен от 8 КБ, значение по умолчанию корректируется пропорционально.) Этот параметр можно изменить в отдельном сеансе, но только до первого обращения к временным таблицам; после этого изменения его значения не будут влиять на текущий сеанс.

Сеанс выделяет временные буферы по мере необходимости до достижения предела, заданного параметром temp_buffers. Если сеанс не задействует временные буферы, то для него хранятся только дескрипторы буферов, которые занимают около 64 байт (в количестве temp_buffers). Однако если буфер действительно используется, он будет дополнительно занимать 8192 байта (или BLCKSZ байт, в общем случае).

– max_prepared_transactions (integer) – Задаёт максимальное число транзакций, которые могут одновременно находиться в «подготовленном» состоянии. При нулевом значении (по умолчанию) механизм подготовленных транзакций отключается. Задать этот параметр можно только при запуске сервера.

Если использовать транзакции не планируется, этот параметр следует обнулить, чтобы не допустить непреднамеренного создания подготовленных транзакций. Если же подготовленные транзакции применяются, то max_prepared_transactions, вероятно, должен быть не меньше, чем max_connections, чтобы подготовить транзакцию можно было в каждом сеансе.

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

– 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). В сложных запросах параллельно могут выполняться несколько операций сортировки или хеширования, и при этом примерно этот объём памяти может использоваться в каждой операции, прежде чем данные начнут вытесняться во временные файлы. Кроме того, такие операции могут выполняться одновременно в разных сеансах. Таким образом, общий объём памяти может многократно превосходить значение work_mem; это следует учитывать, выбирая подходящее значение. Операции сортировки используются для ORDER BY, DISTINCT и соединений слиянием. Хеш-таблицы используются при соединениях и агрегировании по хешу, а также обработке подзапросов IN с применением хеша.

Операции вычисления хеша обычно более требовательны к памяти, чем равнозначные им операции сортировки. Поэтому объём памяти, доступный для хеш-таблиц, определяется произведением work_mem и hash_mem_multiplier и может превышать обычный базовый объём work_mem.

– hash_mem_multiplier (floating point) – Используется для определения максимального объёма памяти, который может выделяться для операций с хешированием. Итоговый объём определяется произведением work_mem и hash_mem_multiplier. Значение по умолчанию равно 1.0, то есть для операций с хешированием устанавливается тот же максимум, равный work_mem, что и для операций с сортировкой.

Значение hash_mem_multiplier имеет смысл увеличить, когда постоянно наблюдается вытеснение данных на диск при выполнении запросов, а прямолинейное увеличение work_mem приводит к дефициту памяти (обычно проявляющемуся в ошибках «нехватка памяти»). В этих случаях значение 1.5 или 2.0 может быть подходящим при смешанной нагрузке, а значение 2.0–8.0 может помочь там, где work_mem уже увеличено до 40 Мбайт или более.

– 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.

Необходимо учесть, что когда выполняется автоочистка, этот объём может быть выделен autovacuum_max_workers раз, поэтому не стоит устанавливать значение по умолчанию слишком большим. Возможно, будет лучше управлять объёмом памяти для автоочистки отдельно, изменяя autovacuum_work_mem.

– autovacuum_work_mem (integer) – Задаёт максимальный объём памяти, который будет использовать каждый рабочий процесс автоочистки. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. При действующем по умолчанию значении -1 этот объём определяется значением maintenance_work_mem. Этот параметр не влияет на поведение команды VACUUM, выполняемой в других контекстах. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– logical_decoding_work_mem (integer) – Задаёт максимальный объём памяти, используемой для логического декодирования, после превышения которого некоторые декодированные изменения будут вымещаться на локальный диск. Тем самым ограничивается объём памяти, используемой подключениями потоковой логической репликации. По умолчанию его значение – 64 мегабайта (64MB). Так как каждое подключение репликации использует один буфер заданного размера, а количество таких подключений к одному серверу обычно невелико (оно ограничивается значением max_wal_senders), значение данного параметра вполне можно сделать достаточно большим, намного превышающим work_mem, чтобы минимизировать объём вымещаемых на диск декодируемых изменений.

– max_stack_depth (integer) – Задаёт максимальную безопасную глубину стека для исполнителя. В идеале это значение должно равняться предельному размеру стека, ограниченному

ядром (который устанавливается командой ulimit -s или аналогичной), за вычетом запаса примерно в мегабайт. Этот запас необходим, потому что сервер проверяет глубину стека не в каждой процедуре, а только в потенциально рекурсивных процедурах, например, при вычислении выражений. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение по умолчанию – два мегабайта (2MB) – выбрано с большим запасом, так что риск переполнения стека минимален. Но с другой стороны, его может быть недостаточно для выполнения сложных функций. Изменить этот параметр могут только суперпользователи.

– shared_memory_type (enum) – Выбирает механизм разделяемой памяти, используя который сервер будет работать с основной областью общей памяти, содержащей общие буферы PG360 и другие общие данные. Допустимые варианты: mmap (для выделения анонимных блоков разделяемой памяти с помощью mmap), sysv (для выделения разделяемой памяти System V функцией shmget). Не все варианты поддерживаются на разных платформах; первый из поддерживаемых данной платформой вариантов становится для неё вариантом по умолчанию. Применять sysv, который нигде не выбирается по умолчанию, вообще не рекомендуется, так как для выделения большого объёма памяти обычно требуется нестандартная настройка ядра.

– dynamic_shared_memory_type (enum) – Выбирает механизм динамической разделяемой памяти, который будет использоваться сервером. Допустимые варианты: posix (для выделения разделяемой памяти POSIX функцией shm_open), sysv (для выделения разделяемой памяти System V функцией shmget) и mmap (для эмуляции разделяемой памяти через отображение в память файлов, хранящихся в каталоге данных).

Параметр для настройки дискового пространства:

– temp_file_limit (integer) – Задаёт максимальный объём дискового пространства, который сможет использовать один процесс для временных файлов, например, при сортировке и хешировании, или для сохранения удерживаемого курсора. Транзакция, которая попытается превысить этот предел, будет отменена. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение -1 (по умолчанию) означает, что предел отсутствует. Изменить этот параметр могут только суперпользователи.

Этот параметр ограничивает общий объём, который могут занимать в момент времени все временные файлы, задействованные в данном процессе PG360. Следует отметить, что это *не* касается файлов явно создаваемых временных таблиц; ограничивается только объём временных файлов, которые создаются неявно при выполнении запросов.

Параметр для настройки использования ресурсов ядра:

– max_files_per_process (integer) – Задаёт максимальное число файлов, которые могут быть одновременно открыты каждым серверным подпроцессом. Значение по умолчанию – 1000 файлов.

Параметры для настройки задержки очистки по стоимости.

Во время выполнения команд VACUUM и ANALYZE система ведёт внутренний счётчик, в котором суммирует оцениваемую стоимость различных выполняемых операций ввода/вывода. Когда накопленная стоимость превышает предел (vacuum_cost_limit), процесс, выполняющий эту

операцию, засыпает на некоторое время (vacuum_cost_delay). Затем счётчик сбрасывается и процесс продолжается.

Данный подход реализован для того, чтобы администраторы могли снизить влияние этих команд на параллельную работу с базой, за счёт уменьшения нагрузки на подсистему ввода- вывода. Очень часто не имеет значения, насколько быстро выполнятся команды обслуживания (например, VACUUM и ANALYZE), но очень важно, чтобы они как можно меньше влияли на выполнение других операций с БД. Администраторы имеют возможность управлять этим, настраивая задержку очистки по стоимости.

По умолчанию этот режим отключён для выполняемых вручную команд VACUUM. Чтобы включить его, нужно установить в vacuum_cost_delay ненулевое значение.

– vacuum_cost_delay (floating point) – Продолжительность времени, в течение которого будет простаивать процесс, превысивший предел стоимости. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию равно нулю, то есть задержка очистки отсутствует. При положительных значениях интенсивность очистки будет зависеть от стоимости.

При настройке интенсивности очистки для vacuum_cost_delay обычно выбираются довольно небольшие значения, вплоть до 1 миллисекунды и меньше. Хотя в vacuum_cost_delay можно задавать дробные значения в миллисекундах, такие задержки могут быть неточными на старых платформах. На таких платформах для увеличения интенсивности VACUUM по сравнению с уровнем, обеспечиваемым при задержке 1 мс, потребуется настраивать другие параметры стоимости очистки. Тем не менее имеет смысл выбирать настолько малую задержку vacuum_cost_delay, насколько может обеспечить платформа; большие задержки не будут полезны.

– vacuum_cost_page_hit (integer) – Примерная стоимость очистки буфера, оказавшегося в общем кеше. Это подразумевает блокировку пула буферов, поиск в хеш-таблице и сканирование содержимого страницы. По умолчанию этот параметр равен одному.

– vacuum_cost_page_miss (integer) – Примерная стоимость очистки буфера, который нужно прочитать с диска. Это подразумевает блокировку пула буферов, поиск в хеш-таблице, чтение требуемого блока с диска и сканирование его содержимого. По умолчанию этот параметр равен 10.

– vacuum_cost_page_dirty (integer) – Примерная стоимость очистки, при которой изменяется блок, не модифицированный ранее. В неё включается дополнительная стоимость ввода/вывода, связанная с записью изменённого блока на диск. По умолчанию этот параметр равен 20.

– vacuum_cost_limit (integer) – Общая стоимость, при накоплении которой процесс очистки будет засыпать. По умолчанию этот параметр равен 200.

Параметры для настройки фоновой записи.

В числе специальных процессов сервера есть процесс *фоновой записи*, задача которого – осуществлять запись «грязных» (новых или изменённых) общих буферов на диск. Когда количество чистых общих буферов считается недостаточным, данный процесс записывает грязные буферы в файловую систему и помечает их как чистые. Это снижает вероятность того, что серверные процессы, выполняющие запросы пользователей, не смогут найти чистые буферы и им придётся сбрасывать грязные буферы самостоятельно. Однако процесс фоновой записи

увеличивает общую нагрузку на подсистему ввода/вывода, так как он может записывать неоднократно изменяемую страницу несколько раз, тогда как её можно было бы записать всего один раз в контрольной точке.

Следующие параметры позволяют настроить поведение фоновой записи для конкретных нужд:

– bgwriter_delay (integer) – Задаёт задержку между раундами активности процесса фоновой записи. Во время раунда этот процесс осуществляет запись некоторого количества загрязнённых буферов (это настраивается следующими параметрами). Затем он засыпает на время bgwriter_delay, и всё повторяется снова. Однако если в пуле не остаётся загрязнённых буферов, он может быть неактивен более длительное время. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. По умолчанию этот параметр равен 200 миллисекундам (200ms). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– bgwriter_lru_maxpages (integer) – Задаёт максимальное число буферов, которое сможет записать процесс фоновой записи за раунд активности. При нулевом значении фоновая запись отключается. По умолчанию значение этого параметра – 100 буферов. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– bgwriter_lru_multiplier (floating point) – Число загрязнённых буферов, записываемых в очередном раунде, зависит от того, сколько новых буферов требовалось серверным процессам в предыдущих раундах. Средняя недавняя потребность умножается на bgwriter_lru_multiplier и предполагается, что именно столько буферов потребуется на следующем раунде. Процесс фоновой записи будет записывать на диск и освобождать буферы, пока число свободных буферов не достигнет целевого значения. (При этом число буферов, записываемых за раунд, ограничивается сверху параметром bgwriter_lru_maxpages.) Таким образом, со множителем, равным 1.0, записывается ровно столько буферов, сколько требуется по предположению («точно по плану»). Увеличение этого множителя даёт некоторую страховку от резких скачков потребностей, тогда как уменьшение отражает намерение оставить некоторый объём записи для серверных процессов. По умолчанию он равен 2.0. Этот параметр можно установить только в файле postgresql.conf или в командной строке при запуске сервера.

– bgwriter_flush_after (integer) – Когда процессом фоновой записи 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. Этот параметр действует не на всех платформах. Если значение параметра задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Он может принимать значение от 0 (при этом управление отложенной записью отключается) до 2 мегабайт (2MB). Значение по умолчанию – 512kB. (Если BLCKSZ отличен от 8 КБ, значение по умолчанию и максимум корректируются пропорционально.) Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

С маленькими значениями bgwriter_lru_maxpages и bgwriter_lru_multiplier уменьшается активность ввода/вывода со стороны процесса фоновой записи, но увеличивается вероятность того, что запись придётся производить непосредственно серверным процессам, что замедлит выполнение запросов.

Параметры для настройки асинхронного поведения:

– effective_io_concurrency (integer) – Задаёт допустимое число параллельных операций ввода/вывода, которое говорит PG360 о том, сколько операций ввода/вывода могут быть выполнены одновременно. Чем больше это число, тем больше операций ввода/вывода будет пытаться выполнить параллельно PG360 в отдельном сеансе. Допустимые значения лежат в интервале от 1 до 1000, а нулевое значение отключает асинхронные запросы ввода/вывода. В настоящее время этот параметр влияет только на сканирование по битовой карте.

Значение по умолчанию равно 1 в системах, где это поддерживается, и 0 в остальных.

– maintenance_io_concurrency (integer) – Этот параметр подобен effective_io_concurrency, но используется для операций обслуживания, которые выполняются в различных клиентских сеансах.

Значение по умолчанию равно 10 в системах, где это поддерживается, и 0 в остальных.

– 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 параметры max_parallel_workers, max_parallel_maintenance_workers и max_parallel_workers_per_gather.

– max_parallel_workers_per_gather (integer) – Задаёт максимальное число рабочих процессов, которые могут запускаться одним узлом Gather или Gather Merge. Параллельные рабочие процессы берутся из пула процессов, контролируемого параметром max_worker_processes, в количестве, ограничиваемом значением max_parallel_workers. Значение по умолчанию – 2. Значение 0 отключает параллельное выполнение запросов.

– max_parallel_maintenance_workers (integer) – Задаёт максимальное число рабочих процессов, которые могут запускаться одной служебной командой. В настоящее время параллельные процессы может использовать только CREATE INDEX при построении индекса-B-дерева и VACUUM без указания FULL. Параллельные рабочие процессы берутся из пула процессов, контролируемого параметром max_worker_processes, в количестве, ограничиваемом значением max_parallel_workers. Значение по умолчанию – 2. Значение 0 отключает использование параллельных исполнителей служебными командами.

– max_parallel_workers (integer) – Задаёт максимальное число рабочих процессов, которое система сможет поддерживать для параллельных операций. Значение по умолчанию – 8. При увеличении или уменьшения этого значения также может иметь смысл скорректировать max_parallel_maintenance_workers и max_parallel_workers_per_gather. Значение данного параметра,

превышающее max_worker_processes, не будет действовать, так как параллельные рабочие процессы берутся из пула рабочих процессов, ограничиваемого этим параметром.

– backend_flush_after (integer) – Когда одним обслуживающим процессом 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 сокращает задержки транзакций, но бывают ситуации (особенно когда объём рабочей нагрузки больше shared_buffers, но меньше страничного кеша ОС), когда производительность может упасть. Этот параметр действует не на всех платформах. Если значение параметра задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Он может принимать значение от 0 (при этом управление отложенной записью отключается) до 2 мегабайт (2MB). По умолчанию он имеет значение 0, то есть это поведение отключено. (Если BLCKSZ отличен от 8 КБ, максимальное значение корректируется пропорционально.)

– old_snapshot_threshold (integer) – Задаёт минимальное время, в течение которого можно пользоваться снимком состояния для запроса без риска получить ошибку снимок слишком стар. Данные, потерявшие актуальность и пребывающие в этом состоянии дольше заданного времени, могут быть вычищены. Это предотвращает замусоривание данными снимков, которые остаются задействованными долгое время. Во избежание получения некорректных результатов из-за очистки данных, которые должны были бы наблюдаться в таком снимке, клиенту будет выдана ошибка, если возраст снимка превысит заданный предел и из этого снимка будет запрошена страница, изменённая со времени его создания.

Если это значение задаётся без единиц измерения, оно считается заданным в минутах. Значение -1 (по умолчанию) отключает это поведение, фактически делая предельный срок снимков бесконечным. Этот параметр можно задать только при запуске сервера.

3.3.5. Журнал упреждающей записи

3.3.5.1. Параметры

Параметры для настройки журнала упреждающей записи:

– wal_level (enum) – определяет, как много информации записывается журнал упреждающей записи (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. Значение 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.

На уровне minimal генерируется минимальный объём WAL. В журнал не записывается информация о производимых до конца транзакции операциях с постоянными отношениями, созданными или перезаписанными в данной транзакции. Это позволяет значительно ускорить такие операции. Такая оптимизация включается после следующих команд:

```
ALTER ... SET TABLESPACE CLUSTER
CREATE TABLE
REFRESH MATERIALIZED VIEW (без CONCURRENTLY) REINDEX
TRUNCATE
```

Однако минимальный журнал не будет содержать достаточно информации для восстановления данных из базовой копии и журналов, поэтому для реализации стратегии архивации WAL и потоковой репликации необходим уровень replica или более высокий.

На уровне logical в журнал записывается та же информация, что и на уровне replica, плюс информация, необходимая для извлечения из журнала наборов логических изменений. Повышение уровня до logical приводит к значительному увеличению объёма WAL, особенно если многие таблицы имеют характеристику REPLICA IDENTITY FULL и выполняется множество команд UPDATE и DELETE.

– fsync (boolean) – если этот параметр установлен, сервер PG360 старается добиться, чтобы изменения были записаны на диск физически, выполняя системные вызовы fsync() или другими подобными методами. Это даёт гарантию, что кластер БД сможет вернуться в согласованное состояние после сбоя оборудования или ОС.

Хотя отключение fsync часто даёт выигрыш в скорости, это может привести к неисправимой порче данных в случае отключения питания или сбоя системы. Поэтому отключать fsync рекомендуется, только если можно восстановить всю базу из внешнего источника.

В качестве примеров, когда отключение fsync неопасно, можно привести начальное наполнение нового кластера данными из копии, обработку массива данных, после которой БД можно удалить и создать заново, либо эксплуатацию копии БД только для чтения, которая регулярно пересоздаётся и не используется для отработки отказа. Качественное оборудование само по себе не является достаточной причиной для отключения fsync.

При смене значения fsync с off на on для надёжного восстановления также необходимо сбросить все изменённые буферы из ядра в надёжное хранилище. Это можно сделать, когда сервер остановлен или когда режим fsync включён, с помощью команды initdb --sync-only, либо выполнить команду sync, размонтировать файловую систему или перезагрузить сервер.

Во многих случаях отключение synchronous_commit для некритичных транзакций может дать больший выигрыш в скорости, чем отключение fsync, при этом не добавляя риски повреждения данных.

Параметр fsync можно задать только в файле postgresql.conf или в командной строке при запуске сервера.

– 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.).

Если значение synchronous_standby_names не задано, для данного параметра имеют смысл только значения on и off; с вариантами remote_apply, remote_write и local будет выбран тот же уровень синхронизации, что и с on. Локальное действие всех отличных от off режимов заключается в ожидании локального сброса WAL на диск. В режиме off ожидание отсутствует, поэтому может образоваться окно от момента, когда клиент узнаёт об успешном завершении, до момента, когда транзакция действительно гарантированно защищена от сбоя. (Максимальный

размер окна равен тройному значению wal_writer_delay.) В отличие от fsync, значение off этого параметра не угрожает целостности данных: сбой ОС или БД может привести к потере последних транзакций, считавшихся зафиксированными, но состояние БД будет точно таким же, как и в случае штатного прерывания этих транзакций. Поэтому выключение режима synchronous_commit может быть полезной альтернативой отключению fsync, когда производительность важнее, чем надёжная гарантия сохранности каждой транзакции.

Если значение synchronous_standby_names не пустое, параметр synchronous_commit также определяет, должен ли сервер при фиксировании транзакции ждать, пока соответствующие записи WAL будут обработаны на ведомом сервере (серверах).

Со значением remote_apply фиксирование завершается только после получения ответов от текущих синхронных ведомых серверов, говорящих, что они получили запись о фиксировании транзакции, сохранили её в надёжном хранилище, а также применили транзакцию, так что она стала видна для запросов на этих серверах. С таким вариантом задержка при фиксировании оказывается больше, так как необходимо дожидаться воспроизведения WAL. Со значением on фиксирование завершается только после получения ответов от текущих синхронных ведомых серверов, подтверждающих, что они получили запись о фиксировании транзакции и передали её в надёжном хранилище. Это гарантирует, что транзакция не будет потеряна, если только БД не будет повреждена и на ведущем, и на всех синхронных ведомых серверах. Со значением remote_write фиксирование завершается после получения ответов от текущих синхронных серверов, говорящих, что они получили запись о фиксировании транзакции и сохранили её в своих ФС. Этот вариант позволяет гарантировать сохранность данных в случае отказа ведомого сервера PG360, но не в случае сбоя на уровне ОС, так как данные могут ещё не достичь надёжного хранилища на этом сервере. Со значением local фиксирование завершается после локального сброса данных, не дожидаясь репликации. Обычно это нежелательный вариант при синхронной репликации, но он представлен для полноты.

Этот параметр можно изменить в любое время; поведение каждой конкретной транзакции определяется значением, действующим в момент её фиксирования. Таким образом, есть возможность и смысл фиксировать некоторые транзакции синхронно, а другие – асинхронно.

– 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 PG360 записывает в WAL всё содержимое каждой страницы при первом изменении этой страницы после контрольной

точки. Это необходимо, потому что запись страницы, прерванная при сбое ОС, 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. (Так как воспроизведение WAL всегда начинается от контрольной точки, достаточно сделать это при первом изменении каждой страницы после контрольной точки. Таким образом, уменьшить затраты на запись полных страниц можно, увеличив интервалы контрольных точек.)

Отключение этого параметра ускоряет обычные операции, но может привести к неисправимому повреждению или незаметной порче данных после сбоя системы. Так как при этом возникают практически те же риски, что и при отключении fsync, хотя и в меньшей степени, отключать его следует только при тех же обстоятельствах, которые перечислялись в рекомендациях для вышеописанного параметра.

Отключение этого параметра не влияет на возможность применения архивов WAL для восстановления состояния на момент времени.

Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. По умолчанию этот параметр имеет значение on.

– wal_log_hints (boolean) – Когда этот параметр имеет значение on, сервер PG360 записывает в WAL всё содержимое каждой страницы при первом изменении этой страницы после контрольной точки, даже при второстепенных изменениях так называемых вспомогательных битов.

Если включён расчёт контрольных сумм данных, изменения вспомогательных битов всегда проходят через WAL и этот параметр игнорируется. С помощью этого параметра можно проверить, насколько больше дополнительной информации записывалось бы в журнал, если бы для БД был включён подсчёт контрольных сумм.

Этот параметр можно задать только при запуске сервера. По умолчанию он имеет значение off.

– wal_compression (boolean) – Когда этот параметр имеет значение on, сервер PG360 сжимает образ полной страницы, записываемый в WAL, когда включён режим full_page_writes или при создании базовой копии. Сжатый образ страницы будет развёрнут при воспроизведении WAL. Значение по умолчанию – off. Изменить этот параметр могут только суперпользователи.

– wal_init_zero (boolean) – Если этот параметр включён (on), создаваемые файлы WAL заполняются нулями. В ряде файловых систем благодаря этому заранее выделяется пространство, которое потребуется для записи WAL. Однако с файловыми системами, работающими по принципу COW (*Copy-On-Write*, Копирование при записи), это может быть бессмысленно, поэтому данный параметр позволяет отключить в данном случае неэффективное поведение. 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. В файловых системах COW может быть быстрее создать новые файлы, поэтому данный параметр позволяет отключить это поведение.

– 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). Это значение можно задать вручную, если выбираемое автоматически слишком мало или велико, но при этом любое положительное число меньше 32 КБ будет восприниматься как 32 КБ. Если это значение задаётся без единиц измерения, оно считается заданным в блоках WAL (размер которых равен XLOG_BLCKSZ байт, обычно это 8 КБ). Этот параметр можно задать только при запуске сервера.

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. Автонастройка, действующая при значении по умолчанию (-1), в большинстве случаев выбирает разумные значения.

– 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. Если предыдущая операция сброса имела место в течение заданного параметром wal_writer_delay времени и полученный за это время объём WAL не достиг значения wal_writer_flush_after, данные WAL только передаются ОС, но не сбрасываются на диск. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию – 200 миллисекунд (200ms). Во многих системах разрешение таймера паузы составляет 10 мс; если задать в wal_writer_delay значение, не кратное 10, может быть получен тот же результат, что и со следующим за ним кратным 10. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– wal_writer_flush_after (integer) – Определяет, при каком объёме процесс записи WAL будет сбрасывать WAL на диск. Если предыдущая операция сброса имела место в течение заданного параметром wal_writer_delay времени и полученный после неё объём WAL не достиг значения wal_writer_flush_after, данные WAL только передаются ОС, но не сбрасываются на диск. Если wal_writer_flush_after равен 0, WAL сбрасывается на диск немедленно. Если это значение задаётся без единиц измерения, оно считается заданным в блоках WAL (размер которых равен XLOG_BLCKSZ байт, обычно это 8 КБ). Значение по умолчанию – 1 мегабайт (1MB). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– 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. Изменение этого параметра в зависимости от характеристик вашего хранилища может быть полезным, если при фиксировании такой транзакции наблюдается замедление других транзакций. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Значение по умолчанию – два мегабайта (2MB).

– 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. Однако этот параметр также увеличивает задержку максимум до commit_delay при каждом сохранении WAL. Эта задержка окажется бесполезной, если никакие другие транзакции не будут зафиксированы за это время, поэтому она добавляется, только если в момент запроса сохранения WAL активны как минимум commit_siblings других транзакций. Кроме того, эти задержки не добавляются при выключенном fsync. Если это значение задаётся без единиц измерения, оно считается заданным в микросекундах. По умолчанию значение commit_delay равно нулю (задержка отсутствует). Изменить этот параметр могут только суперпользователи.

– 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.

3.3.5.2. Контрольные точки

Параметры для настройки контрольных точек:

– checkpoint_timeout (integer) – Максимальное время между автоматическими контрольными точками в WAL. Если это значение задаётся без единиц измерения, оно считается заданным в секундах. Допускаются значения от 30 секунд до одного дня. Значение по умолчанию – пять минут (5min). Увеличение этого параметра может привести к увеличению времени, которое потребуется для восстановления после сбоя. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– checkpoint_completion_target (floating point) – Задаёт целевое время для завершения процедуры контрольной точки, как коэффициент для общего времени между контрольными точками. По умолчанию это значение равно 0.5. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– 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. Этот параметр действует не на всех платформах. Если значение параметра задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Он может принимать значение от 0 (при этом управление отложенной записью отключается) до 2 мегабайт (2MB). Значение по умолчанию – 256kB. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– 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). Если это значение задаётся без единиц измерения, оно считается заданным в секундах. Значение по умолчанию равно 30 секундам (30s). При нуле это предупреждение отключается. Если checkpoint_timeout меньше чем checkpoint_warning, предупреждения так же не будут выводиться. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– max_wal_size (integer) – Максимальный размер, до которого может вырастать WAL во время автоматических контрольных точек. Это мягкий предел; размер WAL может превышать max_wal_size при особых обстоятельствах, например при большой нагрузке, сбое в archive_command или при большом значении wal_keep_size. Если это значение задаётся без единиц измерения, оно считается заданным в мегабайтах. Значение по умолчанию – 1 ГБ. Увеличение этого параметра может привести к увеличению времени, которое потребуется для восстановления после сбоя. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– 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. Если это значение задаётся без единиц измерения, оно считается заданным в мегабайтах. Значение по умолчанию – 80 МБ. Этот параметр можно установить только в postgresql.conf или в командной строке сервера.

3.3.5.3. Архивация

Параметры для настройки архивации журнала упреждающей записи:

– 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).

Параметры archive_mode и archive_command разделены, чтобы команду архивации (archive_command) можно было изменять, не отключая режим архивации. Этот параметр можно задать только при запуске сервера. Режим архивации нельзя включить, когда установлен минимальный уровень WAL (wal_level имеет значение minimal).

– archive_command (string) – Команда локальной оболочки, 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. (Путь задаётся относительно рабочего каталога сервера, то есть каталога данных кластера.) Чтобы вставить в команду символ %, его нужно записать как %%. Важно, чтобы команда возвращала нулевой код, только если она завершается успешно.

Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. Если режим архивации (archive_mode) не был включён при запуске, этот параметр игнорируется. Если значение archive_command – пустая строка (по умолчанию), но archive_mode включён, архивация WAL временно отключается, но сервер продолжает накапливать файлы сегментов WAL в ожидании, что команда будет вскоре определена. Если в качестве archive_command задать команду, которая ничего не делает, но сообщает об успешном завершении, архивация по сути отключается, но при этом нарушается цепочка файлов WAL, необходимых для восстановления архива, поэтому такой вариант следует использовать только в особых случаях.

– archive_timeout (integer) – Команда archive_command вызывается только для завершённых сегментов WAL. Поэтому, если ваш сервер записывает мало данных WAL (или это наблюдается в некоторые периоды времени), от завершения транзакции до надёжного сохранения её в архивном хранилище может пройти довольно много времени. Для ограничения времени существования неархивированных данных можно установить значение archive_timeout, чтобы сервер периодически переключался на новый файл сегмента WAL. Когда этот параметр больше нуля, сервер будет переключаться на новый файл сегмента, если с момента последнего переключения на новый файл прошло заданное время и наблюдалась какая-то активность БД, даже если это была просто контрольная точка. (Контрольные точки пропускаются, если в базе отсутствует активность). Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера.

3.3.5.4. Восстановление из архива

В этом разделе описываются параметры, действующие только в процессе восстановления. Они должны сбрасываться для любой последующей операции восстановления.

Под «восстановлением» здесь понимается и использование сервера в качестве ведомого, и выполнение целевого восстановления данных. Обычно ведомые серверы используется для обеспечения высокой степени доступности и/или масштабируемости чтения, тогда как целевое восстановление производится в случае потери данных.

Чтобы запустить сервер в режиме ведомого необходимо создать в каталоге данных файл standby.signal. Сервер перейдёт к восстановлению и останется в этом состоянии и по достижении конца заархивированного WAL, чтобы осуществлять восстановление дальше. Для этого он подключится к передающему серверу, используя параметры в primary_conninfo, или будет получать новые сегменты WAL с помощью команды restore_command.

Чтобы запустить сервер в режиме целевого восстановления необходимо создать в каталоге данных файл recovery.signal. В случае одновременного существования файлов standby.signal и recovery.signal предпочтение отдаётся режиму ведомого. Режим целевого восстановления завершается после полного воспроизведения WAL из архива или при достижении целевой точки (recovery_target).

Параметры восстановления из архива:

– restore_command (string) – Команда оболочки ОС, 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. (Путь указывается относительно текущего рабочего каталога, т. е. относительно каталога хранения данных кластера.) Любое вхождение %r заменяется на имя файла, в котором содержится последняя действительная точка восстановления. Это самый ранний файл, который требуется хранить для возможности восстановления; зная его имя, размер архива можно уменьшить до минимально необходимого.

Команда возвращает ноль на выходе лишь в случае успешного выполнения. Когда команде будут поступать имена файлов, отсутствующих в архиве, в этом случае она возвращает ненулевой статус.

В случае прерывания команды сигналом (отличным от SIGTERM, который используется для остановки сервера БД) или при возникновении ошибки оболочки (например, если команда не найдена), процесс восстановления будет остановлен и сервер не запустится.

Этот параметр можно задать только при запуске сервера.

– archive_cleanup_command (string) – Этот необязательный параметр указывает команду оболочки ОС, которая будет вызываться при каждой точке перезапуска. Назначение команды archive_cleanup_command – предоставить механизм очистки от старых архивных файлов WAL, которые более не нужны на ведомом сервере.

В случаях, когда команда возвращает ненулевой статус завершения, в журнал записывается предупреждающее сообщение. Если же команда прерывается сигналом или оболочка ОС выдаёт ошибку (например, команда не найдена), вызывается фатальная ошибка.

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

– recovery_end_command (string) – Этот параметр задаёт команду оболочки, которая будет выполнена единожды в конце процесса восстановления. Назначение параметра recovery_end_command – предоставить механизм для очистки после репликации или восстановления.

В случаях, когда команда возвращает ненулевой статус завершения, в журнал записывается предупреждающее сообщение, но сервер, несмотря на это, продолжает запускаться. Если же команда прерывается сигналом или оболочка ОС выдаёт ошибку (например, команда не найдена), кластер БД не запускается.

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

3.3.5.5. Цель восстановления

По умолчанию восстановление производится вплоть до окончания журнала WAL. Чтобы остановить процесс восстановления в более ранней точке, можно использовать один из следующих параметров:

recovery_target;

recovery_target_lsn;

recovery_target_name;

recovery_target_time;

recovery_target_xid.

Если в конфигурационном файле устанавливаются сразу несколько этих параметров, выдаётся ошибка. Задать эти параметры можно только при запуске сервера.

– recovery_target= 'immediate' – Данный параметр указывает, что процесс восстановления должен завершиться, как только будет достигнуто целостное состояние, т. е. как можно раньше. При восстановлении из оперативной резервной копии, это будет точкой, в которой завершился процесс резервного копирования.

– recovery_target_name (string) – Этот параметр указывает именованную точку восстановления (созданную с помощью pg_create_restore_point()), до которой будет производиться восстановление.

– recovery_target_time (timestamp) – Данный параметр указывает точку времени, вплоть до которой будет производиться восстановление. Окончательно точка останова определяется в зависимости от значения параметра recovery_target_inclusive.

Значение этого параметра задаётся в том же формате, что принимается типом данных timestamp with time zone, за исключением того, что в нём нельзя использовать сокращённое название часового пояса (если только переменная timezone_abbreviations не была установлена в файле конфигурации выше). Поэтому рекомендуется задавать числовое смещение от UTC или записывать название часового пояса полностью.

– recovery_target_xid (string) – Параметр указывает идентификатор транзакции, вплоть до которой необходимо произвести процедуру восстановления. 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 параметра recovery_target_inclusive.

– recovery_target_lsn (pg_lsn) – Данный параметр указывает LSN позиции в журнале упреждающей записи, до которой должно выполняться восстановление. Точная позиция остановки зависит также от параметра recovery_target_inclusive. Этот параметр принимает значение системного типа данных pg_lsn.

Следующие параметры уточняют целевую точку восстановления и определяют, что будет происходить при её достижении:

– recovery_target_inclusive (boolean) – Указывает на необходимость остановки сразу после (on) либо до (off) достижения целевой точки. Применяется одновременно с параметрами recovery_target_lsn, recovery_target_time или recovery_target_xid. Этот параметр определяет, нужно ли восстанавливать транзакции, у которых позиция в WAL (LSN), время фиксации либо идентификатор в точности совпадает с заданным соответствующим значением. 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.

Вариант pause позволяет выполнить запросы к БД и убедиться в том, что достигнутая цель оказалась желаемой точкой восстановления. Для снятия с паузы нужно вызвать pg_wal_replay_resume(), что в итоге приведёт к завершению восстановления. Если желаемая точка восстановления еще не достигнута, то нужно остановить сервер, установить более позднюю цель и перезапустить сервер для продолжения восстановления.

Вариант shutdown полезен для получения готового экземпляра сервера в желаемой точке. При этом данный экземпляр сможет воспроизводить дополнительные записи WAL (а при перезапуске ему придётся воспроизводить записи WAL после последней контрольной точки).

Этот параметр не действует, если цель восстановления не установлена. Если не включён режим hot_standby, значение pause действует так же, как и shutdown. Если цель восстановления достигается в процессе повышения, pause действует как promote.

Если задана цель восстановления, но восстановление архива завершается до её завершения, сервер завершит работу с критической ошибкой.

3.3.6. Репликация

Описанные далее параметры 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. Следующие параметры в основном относятся к передающим и ведомым серверам, хотя некоторые параметры имеют смысл только для ведущего. Все эти параметры могут быть разными в рамках одного кластера, если это требуется.

3.3.6.1. Передающие серверы

Эти 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.)

Этот параметр задаёт только минимальный объём сегментов, который будет сохраняться в каталоге pg_wal, для архивации WAL или для восстановления с момента контрольной точки может потребоваться сохранить больше сегментов. Если wal_keep_size равен нулю (это значение по умолчанию), система не сохраняет никакие дополнительные сегменты для ведомых серверов, поэтому число старых сегментов WAL, доступных для ведомых, зависит от положения предыдущей контрольной точки и состояния архивации WAL. Если это значение задаётся без единиц измерения, оно считается заданным в мегабайтах. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– 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. Со значением max_slot_wal_keep_size, равным -1 (по умолчанию), для слотов репликации может сохраняться неограниченный объём файлов WAL. При неотрицательном значении, если позиция restart_lsn для слота репликации отстаёт от текущего LSN более чем на заданное количество мегабайт, использующий этот слот ведомый сервер может лишиться возможности продолжить репликацию вследствие удаления нужных ему файлов WAL. Доступность WAL для слотов репликации показывается в представлении pg_replication_slots. Если это значение указано без единиц измерения, оно считается заданным в мегабайтах. Данный параметр можно задать только в файле postgresql.conf или в командной строке сервера.

– 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. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. По умолчанию этот параметр имеет значение off.

3.3.6.2. Главный сервер

Эти параметры можно задать на главном/ведущем сервере, который должен передавать данные репликации одному или нескольким ведомым. Помимо этих параметров на ведущем

сервере должен быть правильно установлен wal_level, а также может быть включена архивация WAL. 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 (что показывает признак streaming в представлении pg_stat_replication). Указание нескольких имён ведомых серверов позволяет обеспечить очень высокую степень доступности и защиту от потери данных.

Именем ведомого сервера в этом контексте считается значение application_name ведомого сервера, задаваемое в свойствах подключения. При организации физической репликации оно должно задаваться в строке primary_conninfo; по умолчанию это значение параметра cluster_name, если он задан, или walreceiver в противном случае. Для логической репликации его можно задать в строке подключения для подписки (по умолчанию это имя подписки).

Этот параметр принимает список ведомых серверов в одной из следующих форм:

```
[FIRST] число_синхронных ( имя_ведомого [, ...] ) ANY число_синхронных ( имя_ведомого [, ...] ) имя_ведомого [, ...]
```

здесь *число_синхронных* – число синхронных ведомых серверов, от которых необходимо дожидаться ответов для завершения транзакций, а *имя_ведомого* – имя ведомого сервера. Слова FIRST и ANY задают метод выбора синхронных ведомых из перечисленных серверов.

Ключевое слово FIRST, в сочетании с *числом_синхронных*, выбирает синхронную репликацию на основе приоритетов, когда транзакции фиксируются только после того, как их записи в WAL реплицируются на *число_синхронных* ведомых серверов, выбираемых согласно приоритетам. Ведомые серверы, имена которых идут в этом списке первыми, будут иметь больший приоритет и будут считаться синхронными. Серверы, следующие в списке за ними, будут считаться потенциальными синхронными. Если один из текущих синхронных серверов по какой- то причине отключается, он немедленно будет заменён следующим сервером с наибольшим приоритетом. Ключевое слово FIRST может быть опущено.

Ключевое слово ANY, в сочетании с *числом_синхронных*, выбирает синхронную репликацию на основе кворума, когда транзакции фиксируются только после того, как их записи в WAL реплицируются на *как минимум число_синхронных* перечисленных серверов.

Ключевые слова FIRST и ANY воспринимаются без учёта регистра. Если такое же имя оказывается у одного из ведомых серверов, его *имя_ведомого* нужно заключить в двойные кавычки.

Специальному элементу * соответствует имя любого ведомого.

Уникальность имён ведомых серверов не контролируется. В случае дублирования имён более приоритетным будет один из серверов с подходящим именем, хотя какой именно, не определено.

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.

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

– 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. Это значение можно сделать ненулевым на ведущем сервере, работающем с серверами горячего резерва. В результате увеличится время, в течение которого будут успешно выполняться запросы на ведомом сервере без конфликтов из-за ранней очистки строк. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

3.3.6.3. Ведомые серверы

Данные 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. Его можно указать как в строке primary_conninfo, так и отдельно, в файле ~/.pgpass на резервном сервере (для базы данных replication). В строке primary_conninfo имя БД задавать не нужно.

Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Если значение данного параметра меняется во время работы процесса-приёмника WAL, этому процессу посылается сигнал для отключения, и ожидается, что он перезапустится с новым значением (если только определена непустая строка primary_conninfo). Этот параметр оказывает влияние только при работе сервера в режиме ведомого.

– 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. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Если значение данного параметра меняется во время работы процесса-приёмника WAL, этому процессу посылается сигнал для отключения, и ожидается, что он перезапустится с новым значением. Этот параметр не действует, если строка primary_conninfo не определена или сервер работает не в режиме ведомого.

– promote_trigger_file (string) – Указывает триггерный файл, при появлении которого завершается работа в режиме ведомого. Даже если это значение не установлено, существует возможность назначить ведомый сервер ведущим с помощью команды pg_ctl promote или функции pg_promote(). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

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

Задать этот параметр можно только при запуске сервера. Данный параметр используется только в режиме ведомого сервера или при восстановлении архива.

– 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. Задержка max_standby_archive_delay применяется при обработке данных WAL, считываемых из архива (не текущих данных). Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию равно 30 секундам. При значении, равном -1, ведомый может ждать завершения конфликтующих запросов неограниченное время. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

Параметр max_standby_archive_delay определяет не максимальное время, которое отводится для выполнения каждого запроса, а максимальное общее время, за которое должны быть применены изменения из одного сегмента WAL. Таким образом, если один запрос привёл к значительной задержке при обработке сегмента WAL, остальным конфликтующим запросам будет отведено гораздо меньше времени.

– 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. Задержка max_standby_streaming_delay применяется при обработке данных WAL, поступающих при потоковой репликации. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию равно 30 секундам. При значении, равном -1, ведомый может ждать завершения конфликтующих запросов неограниченное время. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

Параметр max_standby_streaming_delay определяет не максимальное время, которое отводится для выполнения каждого запроса, а максимальное общее время, за которое должны быть применены изменения из WAL после получения от главного сервера. Таким образом, если один запрос привёл к значительной задержке, остальным конфликтующим запросам будет отводиться гораздо меньше времени, пока резервный сервер не догонит главный.

– 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 (не задан в primary_slot_name). По умолчанию этот параметр отключён. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Если значение данного параметра меняется во время работы процесса-приёмника WAL, этому процессу посылается сигнал для отключения, и ожидается, что он перезапустится с новым значением.

– 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, где это состояние можно наблюдать в представлении pg_stat_replication. В этом сообщении передаются следующие позиции в журнале упреждающей записи: позиция изменений записанных, изменений, сохранённых на диске, и изменений применённых. Значение параметра определяет максимальный интервал между сообщениями. Сообщения о состоянии передаются при каждом продвижении позиций записанных или

сохранённых на диске изменений, но с промежутком не больше, чем заданный этим параметром. Таким образом, последняя переданная позиция применённых изменений может немного отставать от фактической в текущий момент. Если это значение задаётся без единиц измерения, оно считается заданным в секундах. Значение по умолчанию равно 10 секундам. При нулевом значении этого параметра передача состояния полностью отключается. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– hot_standby_feedback (boolean) – Определяет, будет ли сервер горячего резерва сообщать ведущему или вышестоящему ведомому о запросах, которые он выполняет в данный момент. Это позволяет исключить необходимость отмены запросов, вызванную очисткой записей, но при некоторых типах нагрузки это может приводить к раздуванию БД на ведущем сервере. Эти сообщения о запросах будут отправляться не чаще, чем раз в интервал, задаваемый параметром wal_receiver_status_interval. Значение данного параметра по умолчанию – off. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

Если используется каскадная репликация, сообщения о запросах передаются выше, пока в итоге не достигнут ведущего сервера. На промежуточных серверах эта информация никак не задействуется.

Этот параметр не переопределяет поведение old_snapshot_threshold, установленное на ведущем сервере; снимок на ведомом сервере, имеющий возраст больше заданного указанным параметром на ведущем, может стать недействительным, что приведёт к отмене транзакций на ведомом. Это объясняется тем, что предназначение old_snapshot_threshold заключается в указании абсолютного ограничения времени, в течение которого могут накапливаться мёртвые строки, которое иначе могло бы нарушаться из-за конфигурации ведомого.

– 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. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию – 60 секунд. При значении, равном нулю, тайм-аут отключается. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

– 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. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию – 5 секунд. Задать этот параметр можно только в postgresql.conf или в командной строке сервера.

Этот параметр используется в конфигурациях, когда для узла в схеме восстановления нужно регулировать время ожидания новых данных WAL.

– 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. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию равно нулю, то есть задержка не добавляется.

Возможна ситуация, когда задержка репликации между серверами превышает значение этого параметра. В этом случае дополнительная задержка не добавляется. Задержка вычисляется как разница между меткой времени, записанной в WAL на ведущем сервере, и текущим временем на ведомом. Запаздывание передачи, связанное с задержками в сети или каскадной репликацией, может существенно сократить реальное время ожидания. Если время на главном и ведомом сервере не синхронизировано, это может приводить к применению записей ранее ожидаемого, однако это не очень важно, потому что полезные значения этого параметра намного больше, чем обычно бывает разница во времени между двумя серверами.

Задержка применяется лишь для записей WAL, представляющих фиксацию транзакций. Остальные записи проигрываются незамедлительно, так как их эффект не будет заметен до применения соответствующей записи о фиксации транзакции, благодаря правилам видимости MVCC.

Задержка добавляется, как только восстанавливаемая БД достигает согласованного состояния, и исключается, когда ведущий сервер переключается в режим основного. После переключения ведущий сервер завершает восстановление незамедлительно.

Данный параметр предназначен для применения в конфигурациях с потоковой репликацией; однако если он задан, он будет учитываться во всех случаях, кроме восстановления после сбоя. Задержка, устанавливаемая этим параметром, влияет и на работу механизма hot_standby_feedback, что может привести к увеличению базы на главном сервере; использовать данный параметр при включении этого механизма следует с осторожностью.

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

3.3.6.4. Подписчики

Параметры управления 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.

Рабочие процессы логической репликации берутся из пула, контролируемого параметром max_worker_processes.

Значение по умолчанию – 4. Этот параметр можно задать только при запуске сервера.

– 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.

Одну таблицу может обрабатывать только один рабочий процесс синхронизации.

Рабочие процессы синхронизации берутся из пула, контролируемого параметром max_logical_replication_workers.

Значение по умолчанию – 2. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

Параметры конфигурации wal_receiver_timeout, wal_receiver_status_interval и wal_retrieve_retry_interval также воздействуют на рабочие процессы логической репликации.

3.3.7. Планирование запросов

3.3.7.1. Конфигурация методов планировщика

Параметры конфигурации, дающие возможность влиять на планы, выбираемые оптимизатором запросов. Если автоматически выбранный оптимизатором план конкретного запроса оказался неоптимальным, в качестве *временного* решения можно воспользоваться одним из этих параметров и вынудить планировщик выбрать другой план:

– enable_bitmapscan (boolean) – Включает или отключает использование планов сканирования по битовой карте. По умолчанию имеет значение on (вкл.).

– enable_gathermerge (boolean) – Включает или отключает использование планов соединения посредством сбора. По умолчанию имеет значение on (вкл.).

– enable_hashagg (boolean) – Включает или отключает использование планов агрегирования по хешу. По умолчанию имеет значение on (вкл.).

– enable_hashjoin (boolean) – Включает или отключает использование планов соединения по хешу. По умолчанию имеет значение on (вкл.).

– enable_incremental_sort (boolean) Включает или отключает использование планировщиком инкрементальной сортировки. По умолчанию имеет значение on (вкл.).

– enable_indexscan (boolean) – Включает или отключает использование планов сканирования по индексу. По умолчанию имеет значение on (вкл.).

– enable_indexonlyscan (boolean) – Включает или отключает использование планов сканирования только индекса. По умолчанию имеет значение on (вкл.).

– enable_material (boolean) – Включает или отключает использование материализации при планировании запросов. Полностью исключить материализацию невозможно, но при выключении этого параметра планировщик не будет вставлять узлы материализации, за исключением случаев, где они требуются для правильности. По умолчанию этот параметр имеет значение on (вкл.).

– enable_mergejoin (boolean) – Включает или отключает использование планов соединения слиянием. По умолчанию имеет значение on (вкл.).

– enable_nestloop (boolean) – Включает или отключает использование планировщиком планов соединения с вложенными циклами. Полностью исключить вложенные циклы невозможно, но при выключении этого параметра планировщик не будет использовать данный метод, если можно применить другие. По умолчанию этот параметр имеет значение on.

– enable_parallel_append (boolean) – Включает или отключает использование планировщиком планов с распараллеливанием добавления данных. По умолчанию имеет значение on (вкл.).

– enable_parallel_hash (boolean) – Включает или отключает использование планировщиком планов соединения по хешу с распараллеливанием хеширования. Не действует, если планы соединения по хешу отключены. По умолчанию имеет значение on (вкл.).

– enable_partition_pruning (boolean) – Включает или отключает в планировщике возможность устранять секции секционированных таблиц из планов запроса. Также влияет на возможность планировщика генерировать планы запросов, позволяющие исполнителю пропускать (игнорировать) секции при выполнении запросов. По умолчанию имеет значение on (вкл.).

– enable_partitionwise_join (boolean) – Включает или отключает использование планировщиком соединения с учётом секционирования, что позволяет выполнять соединение секционированных таблиц путём соединения соответствующих секций. Соединение с учётом секционирования в настоящее время может применяться, только когда условия соединения включают все ключи секционирования; при этом ключи должны быть одного типа данных и дочерние секции должны соответствовать один-к-одному. Так как для планирования соединения с учётом секций может потребоваться гораздо больше процессорного времени и памяти, по умолчанию этот параметр выключен (off).

– enable_partitionwise_aggregate (boolean) – Включает или отключает использование планировщиком группировки или агрегирования с учётом секционирования, что позволяет выполнять группировку или агрегирование в секционированных таблицах по отдельности для каждой секции. Если предложение GROUP BY не включает ключи секционирования, на уровне секций может быть выполнено только частичное агрегирование, а затем требуется итоговая обработка. Так как для планирования группировки или агрегирования может потребоваться гораздо больше процессорного времени и памяти, по умолчанию этот параметр выключен (off).

– enable_seqscan (boolean) – Включает или отключает использование планировщиком планов последовательного сканирования. Полностью исключить последовательное сканирование невозможно, но при выключении этого параметра планировщик не будет использовать данный метод, если можно применить другие. По умолчанию этот параметр имеет значение on.

– enable_sort (boolean) – Включает или отключает использование планировщиком шагов с явной сортировкой. Полностью исключить явную сортировку невозможно, но при выключении этого параметра планировщик не будет использовать данный метод, если можно применить другие. По умолчанию этот параметр имеет значение on.

– enable_tidscan (boolean) – Включает или отключает использование планов сканирования TID. По умолчанию имеет значение on (вкл.).

3.3.7.2. Константы стоимости для планировщика

Переменные *стоимости* задаются по произвольной шкале. Значение имеют только их отношения, поэтому умножение или деление всех переменных на один коэффициент никак не повлияет на выбор планировщика. По умолчанию эти переменные определяются относительно стоимости чтения последовательной страницы: то есть, переменную seq_page_cost удобно задать равной 1.0, а все другие переменные стоимости определить относительно неё:

– seq_page_cost (floating point) – Задаёт приблизительную стоимость чтения одной страницы с диска, которое выполняется в серии последовательных чтений. Значение по умолчанию равно 1.0. Это значение можно переопределить для таблиц и индексов в определённом табличном пространстве, установив одноимённый параметр табличного пространства ALTER TABLESPACE.

– random_page_cost (floating point) – Задаёт приблизительную стоимость чтения одной произвольной страницы с диска. Значение по умолчанию равно 4.0. Это значение можно переопределить для таблиц и индексов в определённом табличном пространстве, установив одноимённый параметр табличного пространства ALTER TABLESPACE.

При уменьшении этого значения по отношению к seq_page_cost система начинает предпочитать сканирование по индексу; при увеличении такое сканирование становится более дорогостоящим. Оба эти значения также можно увеличить или уменьшить одновременно, чтобы изменить стоимость операций ввода/вывода по отношению к стоимости процессорных операций, которая определяется следующими параметрами.

– cpu_tuple_cost (floating point) – Задаёт приблизительную стоимость обработки каждой строки при выполнении запроса. Значение по умолчанию – 0.01.

– cpu_index_tuple_cost (floating point) – Задаёт приблизительную стоимость обработки каждой записи индекса при сканировании индекса. Значение по умолчанию – 0.005.

– cpu_operator_cost (floating point) – Задаёт приблизительную стоимость обработки оператора или функции при выполнении запроса. Значение по умолчанию – 0.0025.

– parallel_setup_cost (floating point) – Задаёт приблизительную стоимость запуска параллельных рабочих процессов. Значение по умолчанию – 1000.

– parallel_tuple_cost (floating point) – Задаёт приблизительную стоимость передачи одного кортежа от параллельного рабочего процесса другому процессу. Значение по умолчанию – 0.1.

– min_parallel_table_scan_size (integer) – Задаёт минимальный объём данных таблицы, подлежащий сканированию, при котором может применяться параллельное сканирование. Для параллельного последовательного сканирования объём сканируемых данных всегда равняется размеру таблицы, но когда используются индексы, этот объём обычно меньше. Если это значение задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Значение по умолчанию – 8 мегабайт (8MB).

– min_parallel_index_scan_size (integer) – Задаёт минимальный объём данных индекса, подлежащий сканированию, при котором может применяться параллельное сканирование. При параллельном сканировании по индексу обычно не затрагивается весь индекс; здесь учитывается число страниц, которое по мнению планировщика будет затронуто при сканировании. Этот параметр также учитывается, когда нужно определить, может ли некоторый индекс обрабатываться при параллельной очистке. Если это значение задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Значение по умолчанию – 512 килобайт (512kB).

– effective_cache_size (integer) – Определяет представление планировщика об эффективном размере дискового кеша, доступном для одного запроса. Это представление влияет на оценку стоимости использования индекса; чем выше это значение, тем больше вероятность, что будет применяться сканирование по индексу, чем ниже, тем более вероятно, что будет выбрано последовательное сканирование. Если это значение задаётся без единиц измерения, оно считается заданным в блоках (размер которых равен BLCKSZ байт, обычно это 8 КБ). Значение по умолчанию – 4 гигабайта (4GB). Если BLCKSZ отличен от 8 КБ, значение по умолчанию корректируется пропорционально.

– jit_above_cost (floating point) – Устанавливает предел стоимости запроса, при превышении которого включается JIT– компиляция, если она поддерживается. Применение JIT занимает время при планировании, но может ускорить выполнение запроса в целом. Значение -1 отключает JIT–компиляцию. Значение по умолчанию – 100000.

– jit_inline_above_cost (floating point) – Устанавливает предел стоимости, при превышении которого будет допускаться встраивание функций и операторов в процессе JIT-компиляции. Встраивание занимает время при планировании, но в целом может ускорить выполнение. Присваивать этому параметру значение, меньшее чем jit_above_cost, не имеет смысла. Значение -1 отключает встраивание. Значение по умолчанию – 500000.

– jit_optimize_above_cost (floating point) – Устанавливает предел стоимости, при превышении которого в JIT-компилированных программах может применяться дорогостоящая оптимизация. Такая оптимизация увеличивает время планирования, но в целом может ускорить выполнение. Присваивать этому параметру значение, меньшее чем jit_above_cost, не имеет смысла, а при значениях, превышающих jit_inline_above_cost, положительный эффект маловероятен. Значение -1 отключает оптимизации. Значение по умолчанию – 500000.

3.3.7.3. Генетический оптимизатор запросов

Генетический оптимизатор запросов (GEnetic Query Optimizer, GEQO) осуществляет планирование запросов, применяя эвристический поиск. Это позволяет сократить время планирования для сложных запросов (в которых соединяются множество отношений), ценой того, что иногда полученные планы уступают по качеству планам, выбираемым при полном переборе.

Параметры настройки генетического оптимизатора запросов:

– geqo (boolean) – Включает или отключает генетическую оптимизацию запросов. По умолчанию она включена. В производственной среде её лучше не отключать; более гибко управлять GEQO можно с помощью переменной geqo_threshold.

– geqo_threshold (integer) – Задаёт минимальное число элементов во FROM, при котором для планирования запроса будет привлечён генетический оптимизатор. Значение по умолчанию – 12. Для более простых запросов часто лучше использовать обычный планировщик, производящий полный перебор, но для запросов со множеством таблиц полный перебор займёт слишком много времени, чаще гораздо больше, чем будет потеряно из-за выбора не самого эффективного плана. Таким образом, ограничение по размеру запроса даёт удобную возможность управлять GEQO.

– geqo_effort (integer) – Управляет выбором между сокращением временем планирования и повышением качества плана запроса в GEQO. Это значение должна задаваться целым числом от 1 до 10. Значение по умолчанию равно пяти. Чем больше значение этого параметра, тем больше времени будет потрачено на планирование запроса, но и тем больше вероятность, что будет выбран эффективный план.

– geqo_pool_size (integer) – Задаёт размер пула для алгоритма GEQO, то есть число особей в генетической популяции. Это число должно быть не меньше двух, но полезные значения обычно лежат в интервале от 100 до 1000. Если оно равно нулю (это значение по умолчанию), то подходящее число выбирается, исходя из значения geqo_effort и числа таблиц в запросе.

– geqo_generations (integer) – Задаёт число поколений для GEQO, то есть число итераций этого алгоритма. Оно должно быть не меньше единицы, но полезные значения находятся в том же диапазоне, что и размер пула.

Если оно равно нулю (это значение по умолчанию), то подходящее число выбирается, исходя из geqo_pool_size.

– geqo_selection_bias (floating point) – Задаёт интенсивность селекции для GEQO, то есть селективное давление в популяции. Допустимые значения лежат в диапазоне от 1.50 до 2.00 (это значение по умолчанию).

– geqo_seed (floating point) – Задаёт начальное значение для генератора случайных чисел, который применяется в GEQO для выбора случайных путей в пространстве поиска порядка соединений. Может иметь значение от нуля (по умолчанию) до одного. При изменении этого значения меняется набор анализируемых путей, в результате чего может быть найден как более, так и менее оптимальный путь.

3.3.7.4. Другие параметры планировщика*:*

– default_statistics_target (integer) – Устанавливает значение ориентира статистики по умолчанию, распространяющееся на столбцы, для которых командой ALTER TABLE SET STATISTICS не заданы отдельные ограничения. Чем больше установленное значение, тем больше времени требуется для выполнения ANALYZE, но тем выше может быть качество оценок планировщика. Значение этого параметра по умолчанию – 100.

– constraint_exclusion (enum) – Управляет использованием планировщиком ограничений таблицы для оптимизации запросов. Допустимые значения constraint_exclusion: on (задействовать ограничения всех таблиц), off (никогда не задействовать ограничения) и partition (задействовать ограничения только для дочерних таблиц и подзапросов UNION ALL). Значение по умолчанию – partition. Оно часто помогает увеличить производительность, когда применяются традиционные деревья наследования.

– cursor_tuple_fraction (floating point) – Задаёт для планировщика оценку процента строк, которые будут получены через курсор. Значение по умолчанию – 0.1 (10%). При меньших значениях планировщик будет склонен использовать для курсоров планы с «быстрым стартом», позволяющие получать первые несколько строк очень быстро, хотя для выборки всех строк может уйти больше времени. При больших значениях планировщик стремится оптимизировать общее время запроса. При максимальном значении, равном 1.0, работа с курсорами планируется так же, как и обычные запросы – минимизируется только общее время, а не время получения первых строк.

– from_collapse_limit (integer) – Задаёт максимальное число элементов в списке FROM, до которого планировщик будет объединять вложенные запросы с внешним запросом. При меньших значениях сокращается время планирования, но план запроса может стать менее эффективным. По умолчанию это значение равно 8.

– jit (boolean) – Определяет, может ли PG360 использовать компиляцию JIT, если она поддерживается. По умолчанию параметр включён (on).

– join_collapse_limit (integer) – Задаёт максимальное количество элементов в списке FROM, до достижения которого планировщик будет сносить в него явные конструкции JOIN (за исключением FULL JOIN). При меньших значениях сокращается время планирования, но план запроса может стать менее эффективным.

По умолчанию эта переменная имеет то же значение, что и from_collapse_limit, и это приемлемо в большинстве случаев. При значении, равном 1, предложения JOIN переставляться не будут, так что явно заданный в запросе порядок соединений определит фактический порядок, в

котором будут соединяться отношения. Так как планировщик не всегда выбирает оптимальный порядок соединений, опытные пользователи могут временно задать для этой переменной значение 1, а затем явно определить желаемый порядок.

– parallel_leader_participation (boolean) – Позволяет ведущему процессу выполнять план запроса ниже узлов Gather и Gather Merge, не ожидая рабочие процессы. По умолчанию этот параметр включён (on). Значение off снижает вероятность блокировки рабочих процессов в случае, если ведущий процесс будет читать кортежи недостаточно быстро, но тогда ведущему приходится дожидаться запуска рабочих процессов, и только затем выдавать первые кортежи. Степень положительного или отрицательного влияния ведущего зависит от типа плана, числа рабочих процессов и длительности запроса.

– force_parallel_mode (enum) – Позволяет распараллеливать запрос в целях тестирования, даже когда от этого не ожидается никакого выигрыша в скорости. Допустимые значения параметра force_parallel_mode – off (использовать параллельный режим только когда ожидается увеличение производительности), on (принудительно распараллеливать все запросы, для которых это безопасно) и regress (как on, но с дополнительными изменениями поведения, описанными ниже).

Со значением on узел Gather добавляется в вершину любого плана запроса, для которого допускается распараллеливание, так что запрос выполняется внутри параллельного исполнителя. Даже когда параллельный исполнитель недоступен или не может быть использован, такие операции, как запуск подтранзакции, которые не должны выполняться в контексте параллельного запроса, не будут выполняться в этом режиме, если только планировщик не решит, что это приведёт к ошибке запроса. Если при включении этого параметра возникают ошибки или выдаются неожиданные результаты, вероятно, некоторые функции, задействованные в этом запросе, нужно пометить как PARALLEL UNSAFE (или, возможно, PARALLEL RESTRICTED).

Значение regress действует так же, как и значение on, с некоторыми дополнительными особенностями, предназначенными для облегчения автоматического регрессионного тестирования. Обычно сообщения от параллельных исполнителей включают строку контекста, отмечающую это, но значение regress подавляет эту строку, так что вывод не отличается от выполнения в не параллельном режиме. Кроме того, узлы Gather, добавляемые в планы с этим значением параметра, скрываются в выводе EXPLAIN, чтобы вывод соответствовал тому, что будет получен при отключении этого параметра (со значением off).

– plan_cache_mode (enum) – Подготовленные операторы могут выполняться с использованием специализированных или общих планов. Специализированные планы строятся заново для каждого выполнения с конкретным набором значений параметров, тогда как общий план не зависит от значений параметров и может использоваться многократно. Таким образом, общий план позволяет сэкономить время планирования, но он может быть неэффективным, если идеальные планы в большой степени определяются значениями параметров. Выбор между этими вариантами обычно производится автоматически, но его можно переопределить, воспользовавшись параметром plan_cache_mode. Он может принимать значение auto (по умолчанию), force_custom_plan (принудительно использовать специализированные планы) и force_generic_plan (принудительно использовать общие планы). Значение этого параметра учитывается при выполнении плана, а не при построении.

3.3.8. Регистрация ошибок и протоколирование работы сервера

3.3.8.1.Ведение протоколирования подлежащих аудиту контролируемых событий в СУБД PG360 обеспечивается базовыми (п. 3.3.8.2) и расширенными (п. 3.3.8.3) средствами ведения журнала регистрации событий.

В таблице 3.8.1 указано, какое средство ведения журнала регистрации событий: базовое (log_statement) или расширенное (pg_audit) – можно использовать для регистрации подлежащих аудиту контролируемых событий, используя параметры конфигурации, указанные в третьем столбце.

Таблица 3.3.8.1– Перечень контролируемых событий, подлежащих аудиту

События, подлежащие аудиту	**Средство регистрации**	**Параметры конфигурации**
Неудачное использование механизма аутентификации	log_statement	log_connections = on
Неудачное использование механизма идентификации пользователя, включая предоставленную личность пользователя	log_statement	log_connections = on
Восстановление согласованности	log_statement	Всегда регистрируется независимо от того, какие настройки активны.
Отказ от нового сеанса на основании ограничения количества одновременных сеансов	log_statement	log_connections = on
Отказ в установлении сеанса из-за механизма установления сеанса	log_statement	log_connections = on
Запуск и остановка СУБД	log_statement	Всегда регистрируется независимо от того, какие настройки активны.
Запуск и отключение функций аудита	pg_audit	pg_audit.log = ‘MISC’
Все изменения в конфигурации аудита, которые происходят во время работы функций сбора аудита	pg_audit	
Успешные запросы на выполнение операции над объектом, на который распространяется ПВФБ	pg_audit	Варьируется в зависимости от объекта, который должен быть зарегистрирован в журнале регистрации событий.
Неудачная привязка атрибутов безопасности пользователя к субъекту (например, создание субъекта)	pg_audit	pg_audit.log = ‘ROLE’
Неудачный отзыв атрибутов безопасности	pg_audit	pg_audit.log = ‘ROLE’
Использование функций управления безопасностью	pg_audit	pg_audit.log =‘ROLE,MISC’
Изменение в группе пользователей-исполнителей ролей, и изменения в группе пользователей, являющихся частью роли	pg_audit	pg_audit.log = ‘ROLE’
Использование специальных разрешений	pg_audit	pg_audit.log = ‘ROLE’

СУБД PG360 должна быть сконфигурирована для создания записи контролируемых событий аудита, содержащих следующие сведения:

– дата и время события;

– тип события;

– идентификатор субъекта (если применимо);

– результат (успех или неудача); и

– информацию о контролируемых событиях, определенных в таблице 3.3.8.1, и дополнительную информацию, указанную в третьем столбце таблицы 3.3.8.1.

Информация в записи контролируемых событий аудита выбирается владельцем кластера с помощью параметра конфигурации log_line_prefix в файле конфигурации postgresql.conf путем конфигурации префиксов записи журнала регистрации событий. По умолчанию используется пустая строка. Префикс log_line должен быть установлен с отметкой времени (%t) и именем пользователя (%u).

СУБД PG360 обеспечивает выбор событий, подлежащих регистрации, из набора всех контролируемых событий (см. таблицу 3.3.8.1), которые могут проверяться, на основе:

– идентификатора объекта с помощью pg_audit;

– идентификатора пользователя и группы с помощью параметра ‘log_statement’ в postgrespl.conf;

– проверяемые события, перечисленные в столбце 1 таблицы 3.3.8.1, с использованием параметров конфигурации, изложенных в столбце 3 этой таблицы.

Пользователь PG360 - это роль с привилегией входа в систему LOGIN. Поскольку пользователь это роль с назначенными привилегиями, то во многих случаях пользователь является синонимом группы. В тех случаях, когда роль изменена с роли пользователя на выполнение инструкции (оператора), эта «действующая роль» будет зарегистрирована.

Фильтрация по результату успех или неудача выполняется так же, как фильтрация по типам событий. Для регистрации в журнале только успешных событий, следует не включать события ошибок в параметры фильтрации. Для регистрации в журнале только неудачных событий, следует включить только события ошибок в параметры фильтрации.

Авторизованный администратор может выбрать набор регистрируемых событий из набора всех контролируемых событий, подлежащих аудиту, по уровню важности сообщения, используя log_min_error_statement (enum), как указано в таблице 3.3.8.2.

Таблица 3.3.8.2 – Уровни важности сообщений журнала регистрации событий

Значение	**Описание**
DEBUG [1-5]	Предоставляет информацию для использования разработчиками.
INFO	Предоставляет информацию, неявно запрошенную пользователем, например, во время VACUUM VERBOSE.
NOTICE	Предоставляет информацию, которая может быть полезна пользователям, например, усечение длинных идентификаторов и создание индексов как части первичных ключей.
WARNING	Выдает предупреждения пользователю, например, COMMIT (о фиксации) вне блока транзакции.
ERROR	Сообщает об ошибке, из-за которой текущая транзакция была прервана.
LOG	Сообщает информацию, представляющую интерес для администраторов, например, активность контрольной точки.
FATAL	Сообщает об ошибке, вызвавшей прерывание текущего сеанса.
PANIC	Сообщает об ошибке, из-за которой все сеансы были прерваны.

3.3.8.2. Базовые средства ведения журнала регистрации событий

3.3.8.2.1. PG360 имеет возможность генерировать запись подлежащих аудиту контролируемых событий о сбойных инструкциях (операторах), используя значение параметра 'log_min_messages' в файле конфигурации postgresql.conf. Установка значения параметра 'log_min_messages' в ERROR приведет к тому, что все неудачные инструкции будут записаны в журнал. Любое изменение postgresql.conf вступает в силу только после перезагрузки сервера.

3.3.8.2.2. Параметры для настройки протоколирования работы сервера (куда протоколировать):

– log_destination (string) – PG360 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. Для вывода в формате CSV должен быть включён logging_collector. 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. Файл current_logfiles переписывается когда при прокрутке создаётся новый файл журнала или когда изменяется значение log_destination. Он удаляется, когда в log_destination не задаётся ни stderr, ни csvlog, а также когда сборщик сообщений отключён;

– logging_collector (boolean) – Параметр включает сборщик сообщений (*logging collector*). Это фоновый процесс, который собирает отправленные в stderr сообщения и перенаправляет их в файлы журнала регистрации событий. Такой подход зачастую более полезен чем запись в syslog, поскольку некоторые сообщения в syslog могут не попасть. (Типичный пример с сообщениями об ошибках динамического связывания, другой пример – ошибки в скриптах типа archive_command.) Для установки параметра требуется перезапуск сервера;

– log_directory (string) – При включённом logging_collector, определяет каталог, в котором создаются файлы журнала регистрации событий. 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 файлов журнала регистрации событий. 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 программы для ротации файлов журнала регистрации событий. Если в log_destination включён вывод в формате CSV, то к имени файла журнала регистрации событий 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) – В системах Unix задаёт права доступа к файлам журнала регистрации событий, 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 файлы журнала регистрации событий. Также, может быть полезным значение 0640, разрешающее чтение файлов членам группы. Однако чтобы установить такое значение, нужно каталог для хранения файлов журнала регистрации событий (log_directory) вынести за

пределы каталога данных кластера. В любом случае нежелательно открывать для всех доступ на чтение файлов журнала регистрации событий, 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 файла журнала регистрации событий, 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 файла журнала регистрации событий. 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, PG360 будет перезаписывать существующие файлы журнала регистрации событий, 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 запуск сервера;

– syslog_sequence_numbers (boolean) – Когда сообщения выводятся в syslog и этот параметр включён (по умолчанию), все сообщения будут предваряться последовательно увеличивающимися номерами. 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 PG360 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 PG360. По умолчанию используется PG360. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера.

3.3.8.2.3. Параметры для настройки протоколирования работы сервера (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. 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, необходимо установить для этого параметра значение PANIC. Изменить этот параметр могут только суперпользователи;

– log_min_duration_statement (integer) – Записывает в журнал продолжительность выполнения всех команд, время работы которых не меньше указанного. Если значение этого параметра задаётся без единиц измерения, оно считается заданным в миллисекундах. При нулевом значении записывается продолжительность выполнения всех команд. Со значением -1 (по умолчанию) запись полностью отключается. Изменить этот параметр могут только суперпользователи. Этот параметр переопределяет log_min_duration_sample, то есть запросы с длительностью, превышающей заданное значение, всегда фиксируются в журнале, вне зависимости от параметров извлечения выборки. Для клиентов, использующих расширенный протокол запросов, будет записываться продолжительность фаз: разбор, связывание и выполнение;

– log_min_duration_sample (integer) – Позволяет сделать выборку по продолжительности команд, которые выполнялись не менее чем определённое время. При этом в журнал будут вноситься такие же записи, как и при включённом параметре log_min_duration_statement, но не для всех команд, а только для их подмножества, ограничиваемого параметром log_statement_sample_rate. Если значение этого параметра задаётся без единиц измерения, оно считается заданным в миллисекундах. При нулевом значении для выборки отбираются команды с любой продолжительностью. Со значением -1 (по умолчанию) формирование выборки по продолжительности полностью отключается. Изменить этот параметр могут только суперпользователи. Этот параметр имеет меньший приоритет, чем log_min_duration_statement, то есть команды с длительностью, превышающей log_min_duration_statement, будут регистрироваться в журнале всегда, вне зависимости от того, какой будет выборка;

– log_statement_sample_rate (floating point) – Определяет, какая доля команд с длительностью, достигшей log_min_duration_sample, будет регистрироваться в журнале. Выборка формируется вероятностным образом. Значение по умолчанию – 1.0, то есть выбираются и

регистрируются все команды. Со значением 0 запись команд выборки, в зависимости от их длительности, отключается, так же как и при log_min_duration_sample, равном -1. Изменить этот параметр могут только суперпользователи;

– log_transaction_sample_rate (floating point) – Задаёт долю транзакций, команды из которых будут записываться в журнал дополнительно (помимо команд, записываемых по другим причинам). Этот параметр действует на все транзакции, независимо от длительности команд. Выборка осуществляется вероятностным образом. Значение по умолчанию – 0, то есть команды из дополнительно выбираемых транзакций не записываются. При значении 1 записываются все команды из всех транзакций. Изменить этот параметр могут только суперпользователи.

3.3.8.2.4. Параметры для настройки протоколирования работы сервера (что протоколировать):

– application_name (string) – это любая строка длиной не более 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_checkpoints (boolean) – Включает протоколирование выполнения контрольных точек и точек перезапуска сервера. При этом записывается некоторая статистическая информация. Например, число записанных буферов и время, затраченное на их запись. 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 аутентификации 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. 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). Назначения спецсимволов:

– %a – выводит имя приложения;
– %u – выводит имя пользователя;
– %d – выводит имя БД;
– %r – выводит имя удалённого узла или IP– адрес, а также номер порта;
– %h – выводит имя удалённого узла или IP– адрес;
– %b – выводит тип обслуживающего процесса;
– %p – выводит идентификатор процесса;
– %t – выводит штамп времени, без миллисекунд;
– %m – выводит штамп времени, с миллисекундами;
– %n – выводит штамп времени, с миллисекундами (в виде времени Unix);
– %i – выводит тег команды: тип текущей команды в сессии;
– %e – выводит код ошибки SQLSTATE;
– %c – выводит идентификатор сессии;
– %l – выводит номер строки журнала для каждой сессии или процесса. Начинается с 1;
– %s – выводит штамп времени начала процесса;
– %v – выводит идентификатор виртуальной транзакции (backendID/ localXID);
– %x – выводит идентификатор транзакции (0 если не присвоен);

– %q – Ничего не выводит. Непользовательские процессы останавливаются в этой точке. Игнорируется пользовательскими процессами;

– %% – выводит символ %;

– 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_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. С ненулевыми значениями этого параметра сопряжены некоторые издержки, особенно если привязанные значения передаются в двоичной форме, так как их нужно будет переводить в текстовый вид;

– 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. С ненулевыми значениями этого параметра сопряжены некоторые издержки, так как PG360 должен будет сформировать в памяти текстовые представления всех значений параметров перед выполнением всех операторов, в том числе, выполненных в итоге без ошибок. Издержки дополнительно возрастают, когда привязанные параметры передаются не в текстовой, а в двоичной форме, так как их недостаточно просто скопировать в строку, их нужно ещё преобразовать;

– 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 записывает 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.

3.3.8.3. Расширенные средства ведения журнала регистрации событий pg_audit

3.3.8.3.1. Модуль pg_audit, позволяет гибко настраивать и регистрировать различные события информационной безопасности: вход/выход, попытки входа, попытки доступа к защищаемым объектам, изменение механизмов защиты информации и другие события и связывать их с идентификатором пользователя и группы, вызвавшим эти события.

Модуль pg_audit регистрирует следующие классы инструкций, используя параметр 'pg_audit.log':

– READ – SELECT и COPY, если источником является отношение или запрос;

– WRITE – INSERT, UPDATE, DELETE, TRUNCATE и COPY, когда назначение является отношением;

– FUNCTION – вызовы функций и блоки DO;

– ROLE – инструкции, относящиеся к ролям и привилегиям: GRANT, REVOKE, CREATE/ALTER/DROP ROLE;

– DDL – все DDL, не входящие в класс ROLE;

– MISC – различные команды, например, DISCARD, FETCH, CHECKPOINT, VACUUM, SET;

– MISC_SET – различные команды SET, например, SET ROLE;

– ALL – включает все вышеперечисленное.

Можно указать несколько классов, используя список, разделенный запятыми и классы можно вычесть, поставив перед классом знак «-». Значение по умолчанию «none».

Параметры pg_audit указываются глобально, и их невозможно указать на уровне базы данных или роли.

3.3.8.3.2 Настраиваемые параметры:

– pgaudit.log_catalog – Указывает, что ведение журнала регистрации событий сеанса должно быть включено в случае, когда все отношения в операторе находятся в pg_catalog. Отключение этого параметра уменьшит объем протоколируемых событий в журнале регистрации событий от таких инструментов, как psql и PgAdmin, которые интенсивно запрашивают каталог. По умолчанию включено;

– pgaudit.log_client – Указывает, будут ли сообщения журнала регистрации событий видны клиентскому процессу, такому как psql. Обычно этот параметр следует оставлять отключенным, но он может быть полезен для отладки или других целей. Обратите внимание, что pgaudit.log_level

включается только при включенном pgaudit.log_client. По умолчанию отключено;

– pgaudit.log_level – Указывает уровень журнала регистрации событий, который будет использоваться для записей журнала (допустимые уровни см. в разделе Уровни важности сообщений), но обратите внимание, что ERROR, FATAL и PANIC не разрешены). Этот параметр используется для регрессионного тестирования, а также может быть полезен конечным пользователям для тестирования или других целей. Обратите внимание, что pgaudit.log_level включается только при включенном pgaudit.log_client; в противном случае будет использоваться значение по умолчанию. Значение по умолчанию – log;

– pgaudit.log_parameter – Указывает, что журнал регистрации событий должен включать параметры, которые были переданы с оператором. При наличии параметров они будут включены в формате CSV после текста инструкции. По умолчанию отключено;

– pgaudit.log_parameter_max_size – Указывает, что значения параметра длиннее этого значения (в байтах) не должны регистрироваться в журнале регистрации событий, а заменяются на <long param suppressed>. Он задается в байтах, а не в символах, поэтому не учитывает многобайтовые символы в кодировке текстовых параметров. Этот параметр не действует, если параметр log_parameter выключен. Если этот параметр равен 0 (по умолчанию), все параметры регистрируются независимо от длины. Значение по умолчанию – 0;

– pgaudit.log_relation – Указывает, необходимо ли при протоколировании сеанса создавать отдельную запись в журнале регистрации событий для каждого отношения (TABLE, VIEW и т.д.), на которое ссылается оператор SELECT или DML. По умолчанию отключено;

– pgaudit.log_rows – Указывает, что журнал регистрации событий должен включать строки, извлеченные или затронутые оператором. Если включено, поле строк будет включено после поля параметров. По умолчанию отключено;

– pgaudit.log_statement – Указывает, будет ли протоколирование включать текст инструкции и параметры (если включено). По умолчанию включено;

– pgaudit.log_statement_once – Указывает, будет ли протоколирование включать текст оператора и параметры с первой записью журнала для комбинации оператора/подоператора или с каждой записью. Отключение этого параметра приведет к менее подробному- протоколированию, но может затруднить определение инструкции, сгенерировавшей запись в журнале регистрации событий, хотя пары оператор/подоператор вместе с идентификатором процесса должно быть достаточно для идентификации текста оператора, зарегистрированного с предыдущей записью. По умолчанию отключено;

– pgaudit.role – Указывает основную роль, используемую для ведения журнала регистрации событий объектов. Можно определить несколько ролей аудита, назначив главную роль. Это позволяет нескольким группам отвечать за различные аспекты ведения журналов регистрации событий. По умолчанию нет.

3.3.8.3.3 Записи аудита записываются стандартным средством ведения в журнала регистрации событий и содержат приведенные ниже столбцы в формате, разделенном запятыми. Выходные данные совместимы с форматом CSV, только если часть префикса строки журнала будет удалена из каждой записи журнала. Столбцы журнала регистрации событий:

– AUDIT_TYPE — СЕССИЯ или ОБЪЕКТ;

– STATEMENT_ID — уникальный идентификатор оператора для этого сеанса. Каждый идентификатор оператора представляет внутренний вызов. Идентификаторы операторов являются

последовательными, даже если некоторые операторы не регистрируются. Может быть несколько записей для идентификатора оператора, когда регистрируется более одного отношения;

– SUBSTATEMENT_ID — последовательный идентификатор для каждого подоператора в основном операторе. Например, вызов функции из запроса. Идентификаторы подоператоров непрерывны, даже если некоторые подоператоры не регистрируются. Может быть несколько записей для идентификатора подвыражения, когда регистрируется более одного отношения;

– CLASS – например, READ, ROLE (см. pgaudit.log);

– COMAND – напр. ALTER TABLE, SELECT;

– OBJECT_TYPE – TABLE, INDEX, VIEW и т. д. Доступно для операторов SELECT, DML и большинства операторов DDL;

– OBJECT_NAME – полное имя объекта (например, public.account). Доступно для операторов SELECT, DML и большинства операторов DDL.

– STATEMENT – оператор, выполняемый на серверной части;

– PARAMETER – если параметр pgaudit.log_parameter установлен, то это поле будет содержать параметры оператора в виде цитируемого CSV или <none>, если параметры отсутствуют. В противном случае поле <не регистрируется>.

Используйте log_line_prefix, чтобы добавить любые другие поля, необходимые для удовлетворения требований вашего журнала регистрации событий. Типичным префиксом строки журнала может быть '%m %u %d [%p]: ', который предоставляет дату/время, имя пользователя, имя базы данных и идентификатор процесса для каждого журнала регистрации событий.

Примечание 1:

1) Переименования объектов регистрируются под именем, в которое они были переименованы. Например, переименование таблицы:

```
ALTER TABLE test RENAME TO test2;
```

приведет к следующему результату:

```
AUDIT: SESSION,36,1,DDL,ALTER TABLE,TABLE,public.test2,ALTER TABLE test RENAME TO test2,<not logged>
```

2) Команда может быть зарегистрирована более одного раза. Например, когда таблица создается с первичным ключом, указанным во время создания, индекс для первичного ключа будет регистрироваться независимо, и для индекса в записи создания будет создан другой журнал регистрации событий. Однако несколько записей будут содержаться в одном идентификаторе оператора.

3) Autovacuum и Autoanalyze не регистрируются.

4) Операторы, которые выполняются после того, как транзакция перешла в прерванное состояние, не будут регистрироваться в журнале регистрации событий. Однако оператор, вызвавший ошибку, и любые последующие операторы, выполненные в прерванной транзакции, будут зарегистрированы как ОШИБКИ стандартным средством ведения журнала.

5) Невозможно надежно проверить суперпользователей с помощью pgAudit. Одним из решений является ограничение доступа к учетным записям суперпользователя и использование расширения set_user для увеличения разрешений при необходимости.

3.3.9. Статистика времени выполнения

3.3.9.1. Сбор статистики по запросам и индексам

Параметры, управляющие функциями сбора статистики на уровне сервера. Когда ведётся сбор статистики, собираемые данные можно просмотреть в семействе системных представлений pg_stat и pg_statio:

– track_activities (boolean) – Включает сбор сведений о текущих командах, выполняющихся во всех сеансах (в частности, отслеживается время запуска команды). По умолчанию этот параметр включён. Изменить этот параметр могут только суперпользователи.

– track_activity_query_size (integer) – Задаёт объём памяти, резервируемой для хранения текста выполняемой в данной момент команды в каждом активном сеансе, для поля pg_stat_activity.query. Если это значение задаётся без единиц измерения, оно считается заданным в

байтах. Значение по умолчанию – 1024 байта. Задать этот параметр можно только при запуске сервера.

– track_counts (boolean) – Включает сбор статистики активности в БД. Этот параметр по умолчанию включён, так как собранная информация требуется демону автоочистки. Изменить этот параметр могут только суперпользователи.

– track_io_timing (boolean) – Включает замер времени операций ввода/вывода. Этот параметр по умолчанию отключён, так как для этого требуется постоянно запрашивать текущее время у ОС, что может значительно замедлить работу на некоторых платформах. Изменить этот параметр могут только суперпользователи.

– track_functions (enum) – Включает подсчёт вызовов функций и времени их выполнения. Значение pl включает отслеживание только функций на процедурном языке, a all – также функций на языках SQL и C. Значение по умолчанию – none, то есть сбор статистики по функциям отключён. Изменить этот параметр могут только суперпользователи.

– stats_temp_directory (string) – Задаёт каталог, в котором будут храниться временные данные статистики. Этот путь может быть абсолютным или задаваться относительно каталога данных. Значение по умолчанию – pg_stat_tmp. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

3.3.9.2. Мониторинг статистики

Параметры для настройки мониторина статистики:

– log_statement_stats (boolean)

– log_parser_stats (boolean)

– log_planner_stats (boolean)

– log_executor_stats (boolean) – включают вывод статистики по производительности соответствующего модуля в протокол работы сервера. Параметр log_statement_stats включает вывод общей статистики по операторам, тогда как другие управляют статистикой по модулям (разбор, планирование, выполнение). Включить log_statement_stats одновременно с параметрами, управляющими модулями, нельзя. По умолчанию все эти параметры отключены. Изменить эти параметры могут только суперпользователи.

3.3.10. Автоматическая очистка

Параметры, управляюшие поведением механизма *автоочистки:*

– autovacuum (boolean) – Управляет состоянием демона, запускающего автоочистку. По умолчанию он включён, но чтобы автоочистка работала, нужно также включить track_counts. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Однако автоочистку можно отключить для отдельных таблиц, изменив их параметры хранения.

Если этот параметр отключён, система будет запускать процессы автоочистки, когда это необходимо для предотвращения зацикливания идентификаторов транзакций.

– 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 протоколирование 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.

– autovacuum_max_workers (integer) – Задаёт максимальное число процессов автоочистки (не считая процесс, запускающий автоочистку), которые могут выполняться одновременно. По умолчанию это число равно трём. Задать этот параметр можно только при запуске сервера.

– autovacuum_naptime (integer) – Задаёт минимальную задержку между двумя запусками автоочистки для отдельной БД. Демон автоочистки проверяет БД через заданный интервал времени и выдаёт команды VACUUM и ANALYZE, когда это требуется для таблиц этой базы. Если это значение задаётся без единиц измерения, оно считается заданным в секундах. По умолчанию задержка равна одной минуте (1min). Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера.

– autovacuum_vacuum_threshold (integer) – Задаёт минимальное число изменённых или удалённых кортежей, при котором будет выполняться VACUUM для отдельно взятой таблицы. Значение по умолчанию – 50 кортежей. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Однако данное значение можно переопределить для избранных таблиц, изменив их параметры хранения.

– autovacuum_vacuum_insert_threshold (integer) – Задаёт число добавленных кортежей, при достижении которого будет выполняться VACUUM для отдельно взятой таблицы. Значение по умолчанию – 1000 кортежей. При значении -1 процедура автоочистки не будет производить операции VACUUM с таблицами в зависимости от числа добавленных строк. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Однако данное значение можно переопределить для избранных таблиц, изменив их параметры хранения.

– autovacuum_analyze_threshold (integer) – Задаёт минимальное число добавленных, изменённых или удалённых кортежей, при котором будет выполняться ANALYZE для отдельно взятой таблицы. Значение по умолчанию – 50 кортежей. Этот параметр можно задать только в postgresql.conf или в командной строке при запуске сервера. Однако данное значение можно переопределить для избранных таблиц, изменив их параметры хранения.

– autovacuum_vacuum_scale_factor (floating point) – Задаёт процент от размера таблицы, который будет добавляться к autovacuum_vacuum_threshold при выборе порога срабатывания команды VACUUM. Значение по умолчанию – 0.2 (20% от размера таблицы). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Однако данное значение можно переопределить для избранных таблиц, изменив их параметры хранения.

– autovacuum_vacuum_insert_scale_factor (floating point) – Задаёт процент от размера таблицы, который будет добавляться к autovacuum_vacuum_insert_threshold при выборе порога срабатывания команды VACUUM. Значение по умолчанию – 0.2 (20% от размера таблицы). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Однако данное значение можно переопределить для избранных таблиц, изменив их параметры хранения.

– autovacuum_analyze_scale_factor (floating point) – Задаёт процент от размера таблицы, который будет добавляться к autovacuum_analyze_threshold при выборе порога срабатывания команды ANALYZE. Значение по умолчанию – 0.1 (10% от размера таблицы). Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера. Однако данное значение можно переопределить для избранных таблиц, изменив их параметры хранения.

– autovacuum_freeze_max_age (integer) – Задаёт максимальный возраст (в транзакциях) для поля pg_class.relfrozenxid некоторой таблицы, при достижении которого будет запущена операция VACUUM для предотвращения зацикливания идентификаторов транзакций в этой таблице. Система запустит процессы автоочистки для предотвращения зацикливания, даже если для всех других целей автоочистка отключена.

При очистке могут также удаляться старые файлы из подкаталога pg_xact, поэтому значение по умолчанию сравнительно мало – 200 миллионов транзакций. Задать этот параметр можно только при запуске сервера, но для отдельных таблиц его можно определить по-другому, изменив их параметры хранения.

– autovacuum_multixact_freeze_max_age (integer) – Задаёт максимальный возраст (в мультитранзакциях) для поля pg_class.relminmxid таблицы, при достижении которого будет запущена операция VACUUM для предотвращения зацикливания идентификаторов мультитранзакций в этой таблице. Система запустит процессы автоочистки для предотвращения зацикливания, даже если для всех других целей автоочистка отключена.

При очистке мультитранзакций могут также удаляться старые файлы из подкаталогов pg_multixact/members и pg_multixact/offsets, поэтому значение по умолчанию сравнительно мало – 400 миллионов мультитранзакций. Этот параметр можно задать только при запуске сервера, но для отдельных таблиц его можно определить по-другому, изменив их параметры хранения.

– autovacuum_vacuum_cost_delay (floating point) – Задаёт задержку при превышении предела стоимости, которая будет применяться при автоматических операциях VACUUM. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. При значении -1 применяется обычная задержка vacuum_cost_delay. Значение по умолчанию – 2 миллисекунды. 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.

– autovacuum_vacuum_cost_limit (integer) – Задаёт предел стоимости, который будет учитываться при автоматических операциях VACUUM. При значении -1 (по умолчанию) применяется обычное значение vacuum_cost_limit. Это значение распределяется пропорционально среди всех работающих процессов автоочистки, если их больше одного, так что сумма ограничений всех процессов никогда не превосходит данный предел. 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.

3.3.11. Параметры клиентских сеансов по умолчанию

Параметры для настройки поведения 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. 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 БД (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.

Текущее действующее значение пути поиска можно получить, воспользовавшись SQL-функцией current_schemas. Это значение может отличаться от значения search_path, так как current_schemas показывает, как были преобразованы элементы, фигурирующие в search_path.

– row_security (boolean) – Эта переменная определяет, должна ли выдаваться ошибка при применении политик защиты строк. Со значением on политики применяются в обычном режиме. Со значением off запросы, ограничиваемые минимум одной политикой, будут выдавать ошибку. Значение по умолчанию – on. Значение off рекомендуется, когда ограничение видимости строк чревато некорректными результатами; например, pg_dump устанавливает это значение. Эта

переменная не влияет на роли, которые обходят все политики защиты строк, а именно, на суперпользователей и роли с атрибутом BYPASSRLS.

– default_table_access_method (string) – Этот параметр задаёт табличный метод доступа по умолчанию, который будет использоваться при создании таблиц или материализованных представлений, если в команде CREATE не будет явно указан метод доступа, или при выполнении команды SELECT ... INTO, в которой явно задать метод доступа нельзя. Значение по умолчанию – heap.

– default_tablespace (string) – Эта переменная устанавливает табличное пространство по умолчанию, в котором будут создаваться объекты (таблицы и индексы), когда в команде CREATE табличное пространство не указывается явно.

Её значением может быть либо имя табличного пространства, либо пустая строка, подразумевающая использование табличного пространства по умолчанию в текущей БД. Если табличное пространство с заданным именем не существует, PG360 будет автоматически использовать табличное пространство по умолчанию. Если используется не пространство по умолчанию, пользователь должен иметь право CREATE для него, иначе он не сможет создавать объекты.

Эта переменная не используется для временных таблиц.

Эта переменная также не используется при создании БД. По умолчанию, новая БД наследует выбор табличного пространства от базы-шаблона, из которой она копируется.

Если на момент создания секционированной таблицы значение этого параметра отлично от пустого, оно задаёт табличное пространство для секционированной таблицы, в котором по умолчанию будут располагаться её секции, создаваемые в дальнейшем, даже если позже значение default_tablespace будет изменено.

– temp_tablespaces (string) – Эта переменная задаёт табличные пространства, в которых будут создаваться временные объекты (временные таблицы и индексы временных таблиц), когда в команде CREATE табличное пространство не указывается явно. В этих табличных пространствах также создаются временные файлы для внутреннего использования, например, для сортировки больших наборов данных. Её значение содержит список имён табличных пространств.

Когда temp_tablespaces задаётся интерактивно, указание несуществующего табличного пространства считается ошибкой, как и указание табличного пространства, для которого пользователь не имеет права CREATE. Однако при использовании значения, заданного ранее, несуществующие табличные пространства и пространства, для которых у пользователя нет права CREATE, просто игнорируются. В частности, это касается значения, заданного в postgresql.conf.

По умолчанию значение этой переменной – пустая строка. С таким значением все временные объекты создаются в табличном пространстве по умолчанию, установленном для текущей БД.

– check_function_bodies (boolean) – Этот параметр обычно включён. Выключение этого параметра (присваивание ему значения off) отключает проверку строки с телом функции, передаваемой команде CREATE FUNCTION. Отключение проверки позволяет избежать побочных эффектов процесса проверки и исключить ложные срабатывания из-за таких проблем, как ссылки вперёд. Этому параметру нужно присваивать значение off перед загрузкой функций от лица других пользователей; pg_dump делает это автоматически.

– 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».

– default_transaction_read_only (boolean) – SQL-транзакции в режиме «только чтение» не могут модифицировать не временные таблицы. Этот параметр определяет, будут ли новые транзакции по умолчанию иметь характеристику «только чтение». Значение этого параметра по умолчанию – off (допускается чтение и запись).

– default_transaction_deferrable (boolean) – Транзакция, работающая на уровне изоляции serializable, в режиме «только чтение» может быть задержана, прежде чем будет разрешено её выполнение. Однако когда она начинает выполняться, для обеспечения сериализуемости не требуется никаких дополнительных усилий, так что коду сериализации ни при каких условиях не придётся прерывать её из-за параллельных изменений, поэтому это вполне подходит для длительных транзакций в режиме «только чтение».

Этот параметр определяет, будет ли каждая новая транзакция по умолчанию откладываемой. В настоящее время его действие не распространяется на транзакции, для которых устанавливается режим «чтение/запись» или уровень изоляции ниже serializable. Значение по умолчанию – off (выкл.).

– 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.

– transaction_read_only (boolean) – Данный параметр отражает характеристику «только чтение» для текущей транзакции. В начале каждой транзакции ему присваивается текущее значение default_transaction_read_only. Последующая попытка изменить значение этого параметра равнозначна команде SET TRANSACTION.

– transaction_deferrable (boolean) – Данный параметр отражает характеристику «откладываемости» для текущей транзакции. В начале каждой транзакции ему присваивается текущее значение default_transaction_deferrable. Последующая попытка изменить значение этого параметра равнозначна команде SET TRANSACTION.

– session_replication_role (enum) – Управляет срабатыванием правил и триггеров, связанных с репликацией, в текущем сеансе. Изменение этой переменной требует наличия прав суперпользователя и приводит к сбросу всех ранее кешированных планов запросов. Она может принимать следующие значения: origin (значение по умолчанию), replica и local.

– statement_timeout (integer) – Задаёт максимальную длительность выполнения оператора, при превышении которой оператор прерывается. Если log_min_error_statement имеет значение ERROR или ниже, оператор, прерванный по тайм-ауту, будет также записан в журнал. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. При значении, равном нулю (по умолчанию), этот контроль длительности отключается.

Устанавливать значение statement_timeout в postgresql.conf не рекомендуется, так как это повлияет на все сеансы.

– lock_timeout (integer) – Задаёт максимальную длительность ожидания любым оператором получения блокировки таблицы, индекса, строки или другого объекта БД. Если ожидание не закончилось за указанное время, оператор прерывается. Это ограничение действует на каждую попытку получения блокировки по отдельности и применяется как к явным запросам блокировки (например, LOCK TABLE или SELECT FOR UPDATE без NOWAIT), так и к неявным. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. При значении, равном нулю (по умолчанию), этот контроль длительности отключается.

В отличие от statement_timeout, этот тайм-аут может произойти только при ожидании блокировки. При ненулевом statement_timeout бессмысленно задавать в lock_timeout такое же или большее значение, так как тайм-аут оператора всегда будет происходить раньше. Если log_min_error_statement имеет значение ERROR или ниже, оператор, прерванный по тайм-ауту, будет записан в журнал.

Устанавливать значение lock_timeout в postgresql.conf не рекомендуется, так как это повлияет на все сеансы.

– idle_in_transaction_session_timeout (integer) – Завершать любые сеансы, в которых открытая транзакция простаивает дольше заданного времени. Это позволяет освободить все блокировки сеанса и вновь задействовать слот подключения; также это позволяет очистить кортежи, видимые только для этой транзакции.

Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. При значении, равном нулю (по умолчанию), этот контроль длительности отключается.

– vacuum_freeze_table_age (integer) – Задаёт максимальный возраст для поля pg_class.relfrozenxid таблицы, при достижении которого VACUUM будет производить агрессивное сканирование. Агрессивное сканирование отличается от обычного сканирования VACUUM тем, что затрагивает все страницы, которые могут содержать незамороженные XID или MXID, а не только те, что могут содержать мёртвые кортежи. Значение по умолчанию – 150 миллионов транзакций. Хотя пользователи могут задать любое значение от нуля до двух миллиардов, в VACUUM введён внутренний предел для действующего значения, равный 95% от autovacuum_freeze_max_age, чтобы периодически запускаемая вручную команда VACUUM имела шансы выполниться, прежде чем для таблицы будет запущена автоочистка для предотвращения зацикливания.

– vacuum_freeze_min_age (integer) – Задаёт возраст для отсечки (в транзакциях), при достижении которого команда VACUUM должна замораживать версии строк при сканировании таблицы. Значение по умолчанию – 50 миллионов транзакций. Хотя пользователи могут задать любое значение от нуля до одного миллиарда, в VACUUM введён внутренний предел для действующего значения, равный половине autovacuum_freeze_max_age, чтобы принудительная автоочистка выполнялась не слишком часто.

– vacuum_multixact_freeze_table_age (integer) – Задаёт максимальный возраст для поля pg_class.relminmxid таблицы, при достижении которого команда VACUUM будет выполнять агрессивное сканирование. Агрессивное сканирование отличается от обычного сканирования VACUUM тем, что затрагивает все страницы, которые могут содержать незамороженные XID или MXID, а не только те, что могут содержать мёртвые кортежи. Значение по умолчанию – 150

миллионов мультитранзакций. Хотя пользователи могут задать любое значение от нуля до двух миллиардов, в VACUUM введён внутренний предел для действующего значения, равный 95% от autovacuum_multixact_freeze_max_age, чтобы периодически запускаемая вручную команда VACUUM имела шансы выполниться, прежде чем для таблицы будет запущена автоочистка для предотвращения зацикливания.

– vacuum_multixact_freeze_min_age (integer) – Задаёт возраст для отсечки (в мультитранзакциях), при достижении которого команда VACUUM должна заменять идентификаторы мультитранзакций новыми идентификаторами транзакций или мультитранзакций при сканировании таблицы. Значение по умолчанию – 5 миллионов мультитранзакций.

– bytea_output (enum) – Задаёт выходной формат для значения типа bytea. Это может быть формат hex (по умолчанию) или escape. Входные значения bytea воспринимаются в обоих форматах, независимо от данного параметра.

– xmlbinary (enum) – Задаёт способ кодирования двоичных данных в XML. Допустимые варианты, определённые в стандарте XML-схем: base64 и hex. Значение по умолчанию – base64.

– xmloption (enum) – Задаёт подразумеваемый по умолчанию тип преобразования между XML и символьными строками (DOCUMENT или CONTENT). Значение по умолчанию – CONTENT (кроме него допускается значение DOCUMENT).

– gin_pending_list_limit (integer) – Задаёт максимальный размер очереди записей GIN, которая используется, когда включён режим fastupdate. Если размер очереди превышает заданный предел, записи из неё массово переносятся в основную структуру данных индекса GIN, и очередь очищается. Если это значение задаётся без единиц измерения, оно считается заданным в килобайтах. Размер по умолчанию – четыре мегабайта (4MB). Этот предел можно переопределить для отдельных индексов GIN, изменив их параметры хранения.

Параметры для настройки языковой среды и форматов:

– DateStyle (string) – Задаёт формат вывода значений даты и времени, а также правила интерпретации неоднозначных значений даты.

– IntervalStyle (enum) – Задаёт формат вывода для значений-интервалов.

– TimeZone (string) – Задаёт часовой пояс для вывода и ввода значений времени. Встроенное значение по умолчанию – GMT, но можно переопределить в postgresql.conf, initdb устанавливает в нём значение, соответствующее системному окружению.

– timezone_abbreviations (string) – Задаёт набор сокращений часовых поясов, которые будут приниматься сервером во вводимых значениях даты и времени. Значение по умолчанию – 'Default', которое представляет набор основных сокращений, принятых в мире.

– extra_float_digits (integer) – Этот параметр корректирует число цифр в текстовом представлении чисел с плавающей точкой, включая значения float4, float8 и геометрических типов.

Если его значение равно 1 (по умолчанию) или больше, числа с плавающей точкой выводятся в кратчайшем точном виде. Выводимое фактически количество цифр зависит от выводимого числа, а не от значения данного параметра. Для значений float8 может потребоваться максимум 17 цифр, а для значений float4 – максимум 9. Данное представление является и

быстрым, и точным, так как оно позволяет при правильном прочтении в точности восстановить двоичное число с плавающей точкой. Ради исторической совместимости этот параметр принимает значения до 3 включительно.

Если его значение равно нулю или отрицательно, выводимое число округляется до заданной десятичной точности. Точность в данном случае определяется обычным количеством цифр для типа (FLT_DIG или DBL_DIG), уменьшенным на значение этого параметра.

– client_encoding (string) – Задаёт кодировку (набор символов) на стороне клиента. По умолчанию выбирается кодировка БД.

– lc_messages (string) – Устанавливает язык выводимых сообщений. Набор допустимых значений зависит от системы. Если эта переменная определена как пустая строка (по умолчанию), то действующее значение получается из среды выполнения сервера, в зависимости от системы.

Изменить этот параметр могут только суперпользователи. Он влияет и на сообщения, которые сервер передаёт клиентам, и на те, что записываются в журнал, поэтому неподходящее значение может сделать серверные журналы нечитаемыми.

– lc_monetary (string) – Устанавливает локаль для форматирования денежных сумм, например с использованием функций семейства to_char. Набор допустимых значений зависит от системы. Если эта переменная определена как пустая строка (по умолчанию), то действующее значение получается из среды выполнения сервера, в зависимости от системы.

– lc_numeric (string) – Устанавливает локаль для форматирования чисел, например с использованием функций семейства to_char. Набор допустимых значений зависит от системы. Если эта переменная определена как пустая строка (по умолчанию), то действующее значение получается из среды выполнения сервера, в зависимости от системы.

– lc_time (string) – Устанавливает локаль для форматирования даты и времени, например с использованием функций семейства to_char. Набор допустимых значений зависит от системы. Если эта переменная определена как пустая строка (по умолчанию), то действующее значение получается из среды выполнения сервера, в зависимости от системы.

– default_text_search_config (string) – Выбирает конфигурацию текстового поиска для тех функций текстового поиска, которым не передаётся аргумент, явно указывающий конфигурацию. Встроенное значение по умолчанию – pg_catalog.simple, но initdb при инициализации записывает в файл конфигурации сервера значение, соответствующее выбранной локали lc_ctype, если удастся найти такую конфигурацию текстового поиска.

3.3.12. Управление блокировками

Параметры для настройки управления блокировками:

– deadlock_timeout (integer) – Время ожидания блокировки, по истечении которого будет выполняться проверка состояния взаимоблокировки. При увеличении значения этого параметра сокращается время, уходящее на ненужные проверки взаимоблокировки, но замедляется реакция на реальные взаимоблокировки. Если это значение задаётся без единиц измерения, оно считается заданным в миллисекундах. Значение по умолчанию – одна секунда (1s), что близко к минимальному значению, которое стоит применять на практике. На сервере с большой нагрузкой имеет смысл увеличить его. В идеале это значение должно превышать типичное время транзакции, чтобы повысить шансы на то, что блокировка всё-таки будет освобождена, прежде

чем ожидающая транзакция решит проверить состояние взаимоблокировки. Изменить этот параметр могут только суперпользователи.

Когда включён параметр log_lock_waits, данный параметр также определяет, спустя какое время в журнал сервера будут записываться сообщения об ожидании блокировки.

– max_locks_per_transaction (integer) – Общая таблица блокировок отслеживает блокировки для max_locks_per_transaction * (max_connections + max_prepared_transactions) объектов (например, таблиц); таким образом, в любой момент времени может быть заблокировано не больше этого числа различных объектов. Этот параметр управляет средним числом блокировок объектов, выделяемым для каждой транзакции; отдельные транзакции могут заблокировать и больше объектов, если все они умещаются в таблице блокировок. Значение по умолчанию, 64. 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.

– max_pred_locks_per_transaction (integer) – Общая таблица предикатных блокировок отслеживает блокировки для max_pred_locks_per_transaction * (max_connections + max_prepared_transactions) объектов (например, таблиц); таким образом, в один момент времени может быть заблокировано не больше этого числа различных объектов. Этот параметр управляет средним числом блокировок объектов, выделяемым для каждой транзакции; отдельные транзакции могут заблокировать и больше объектов, если все они умещаются в таблице блокировок. Значение по умолчанию 64. Этот параметр можно задать только при запуске сервера.

– max_pred_locks_per_relation (integer) – Этот параметр определяет, для скольких страниц или кортежей одного отношения могут устанавливаться предикатные блокировки, прежде чем вместо них будет затребована одна блокировка для всего отношения. Значения, большие или равные нулю, задают абсолютный предел, а с отрицательным значением пределом будет значение max_pred_locks_per_transaction, делённое на модуль данного. По умолчанию действует значение -2. Этот параметр можно задать только в файле postgresql.conf или в командной строке при запуске сервера.

– max_pred_locks_per_page (integer) – Этот параметр определяет, для скольких строк на одной странице могут устанавливаться предикатные блокировки, прежде чем вместо них будет затребована одна блокировка для всей страницы. Этот параметр можно задать только в файле postgresql.conf или в командной строке при запуске сервера.

3.3.13. Обработка ошибок

Параметры для настройки обработки ошибок:

– exit_on_error (boolean) – Если этот параметр включён, любая ошибка приведёт к прерыванию текущего сеанса. По умолчанию он отключён, так что сеанс будет прерываться только при критических ошибках.

– restart_after_crash (boolean) – Когда этот параметр включён (это состояние по умолчанию), PG360 будет автоматически перезагружаться после сбоя серверного процесса. Такой вариант позволяет обеспечить максимальную степень доступности БД.

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

– data_sync_retry (boolean) – При выключенном значении этого параметра (по умолчанию) PG360 будет выдавать ошибку уровня PANIC в случае неудачи при попытке сохранить изменённые данные в файловой системе. В результате сервер БД остановится аварийно. Задать этот параметр можно только при запуске сервера.

В некоторых ОС состояние данных в кеше внутри ядра оказывается неопределённым при ошибке записи. В каких-то случаях эти данные могут быть просто утеряны, и повторять попытку записи небезопасно: вторая попытка может оказаться успешной, тогда как на деле данные не сохранены. В этих обстоятельствах единственный способ избежать потери данных – восстановить их из WAL после такого сбоя, но перед этим желательно выяснить причину проблемы и, возможно, заменить нерабочее оборудование.

Если включить этот параметр, PG360 в случае сбоя при записи выдаст ошибку, но продолжит работу в расчёте повторить операцию сохранения данных при последующей контрольной точке. Включать его следует, только если достоверно известно, как поступает система с данными в буфере при ошибке записи.

3.3.14. Предопределенные параметры

Следующие параметры доступны только для чтения:

– 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) – В Unix-системах этот параметр показывает разрешения для каталога данных (data_directory), определённые при запуске.

– lc_collate (string) – Сообщает локаль, по правилам которой выполняется сортировка текстовых данных. Это значение определяется при создании БД.

– lc_ctype (string) – Сообщает локаль, определяющую классификацию символов. Это значение определяется при создании БД. 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 БД (набор символов). Она определяется при создании БД.

– 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 PG360.

– 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 упреждающей записи. Значение по умолчанию – 16 МБ.

3.4. Аутентификация клиентского приложения

При подключении к серверу БД, клиентское приложение указывает имя пользователя PG360, 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 БД. Следовательно, важно указать на этом этапе, к каким базам пользователь имеет право подключиться.

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

PG360 предлагает несколько различных методов аутентификации клиентов. Метод аутентификации конкретного клиентского соединения может основываться на адресе компьютера клиента, имени БД, имени пользователя.

Имена пользователей БД PG360 не имеют прямой связи с пользователями ОС на которой запущен сервер. Если у всех пользователей БД заведена учётная запись в ОС сервера, то имеет смысл назначить им точно такие же имена для входа в PG360. Однако сервер, принимающий удалённые подключения, может иметь большое количество пользователей БД, у которых нет учётной записи в ОС. В таких случаях не требуется соответствие между именами пользователей БД и именами пользователей ОС.

3.4.1. Файл pg_hba.conf

Аутентификация клиентов управляется конфигурационным файлом, который называется pg_hba.conf и расположен в каталоге с данными кластера БД (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, с помощью конфигурационного параметра hba_file.

Формат файла pg_hba.conf представляет собой набор записей, по одной в строке. Пустые строки игнорируются, как и любой текст комментария после знака #. Записи не продолжаются на

следующей строке. Записи состоят из некоторого количества полей, разделённых между собой пробелом и/или tabs. В полях могут быть использованы пробелы, если они взяты в кавычки. Если в кавычки берётся какое-либо зарезервированное слово в поле БД, пользователя или адресации (например, all или replication), то слово теряет своё особое значение и просто обозначает БД, пользователя или сервер с данным именем.

Каждая запись обозначает тип соединения, диапазон IP-адресов клиента (если он соотносится с типом соединения), имя БД, имя пользователя, и способ аутентификации, который будет использован для соединения в соответствии с этими параметрами. Первая запись с соответствующим типом соединения, адресом клиента, указанной БД 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]
hostssl база пользователь адрес метод-аутентификации [параметры-аутентификации]
hostnossl база пользователь адрес метод-аутентификации [параметрыа-утентификации]
host база пользователь IP-адрес IP-маска метод-аутентификации [параметры-аутентификации]
hostssl база пользователь IP-адрес IP-маска метод-аутентификации [параметры-аутентификации]
hostnossl база пользователь IP-адрес IP-маска метод-аутентификации [параметры-аутентификации]
```

Значения полей:

– local – запись будет соответствовать локальным подключениям, производимым через Unix-сокеты. Если записи такого типа нет, то подключения через Unix-сокеты будут запрещены.

– host – запись соответствует подключениям через TCP/IP, как с SSL так и без. Необходимо обратить внимание на значение параметра listen_addresses — по умолчанию такие соединения будут ожидаться только с localhost. Для того чтобы подключения были возможны с других удаленных узлов, необходимо прописать соответствующие адреса.

– hostssl – запись соответствует только SSL подключениям через TCP/IP. Для использования этой возможность, сервер должен быть собран с поддержкой SSL и включен параметром конфигурации ssl в файле `postgresql.conf;`

– hostnossl – запись соответствует только не SSL (обычным) подключениям через TCP/IP;

– *база* – определяет имена баз данных, доступ к которым описывает данная запись. Значе ниями могут быть:

собственное имя или имена баз данных кластера PG360, разделенные запятой;

@filename – имена баз данных хранятся во внешнем файле filename;

all – все БД кластера PG360;

sameuser – БД с тем же именем, что и пользователь в поле USER запрашиваемой БД;

samerole – БД с тем же именем, что и роль пользователя в поле USER запрашиваемой БД;

replication – определяет, что запись будет соответствовать случаю подключения для репликации.

– пользователь – имя или имена пользователей PG360, правила доступа для которых определяет данная запись. Значениями могут быть:

собственно имя или имена пользователей PG360, разделенные запятой; all –

все пользователи PG360;

@filename – имена пользователей хранятся во внешнем файле filename;

+группа – все пользователи, входящие в указанную группу/группы.

Примечание 2: в PG360 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.

– *адрес* – указывает адрес (или адреса) клиентской машины, которым соответствует данная запись. Данное поле не применимо к записям local. Значениями могут быть:

all – любой адрес;

samehost – IP-адрес самого сервера PG360;

samenet – любой адрес любой подсети, к которой сервер подключён напрямую;

.hostname – суффикс имени узла.

IP адрес или диапазон IP адресов в нотации CIDR

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. Записи в формате IPv6 не будут приниматься, если системная библиотека С не поддерживает адреса Ipv6.

Имя узла (.hostname) сравнивается с результатом обратного преобразования IP-адреса (например, обратного DNS-запроса). Всё, что не является диапазоном IP-адресов или специальным ключевым словом, воспринимается как имя узла.

При сравнении имён узлов регистр не учитывается. Если имена совпали, выполняется прямое преобразование имени (например, прямой DNS-запрос) для проверки, относится ли заданный IP-адрес к адресам, соответствующим имени.

Если двусторонняя проверка пройдена, запись считается соответствующей узлу. Для ускорения проверки (преобразования) имени узла рекомендуется использовать локальный кеш разрешения имён, например, nscd.

Для отображения имя узла клиента вместо IP-адреса в журнале регистрации событий, рекомендуется включить конфигурационный параметр log_hostname (см.п.3.3.8)

– *IP-адрес IP-маска* – эти два поля могут быть использованы как альтернатива записи *IP-адрес*/*длина-маски*. Эти поля не применимы к записям local.

– *метод-аутентификации* – указывает метод аутентификации, который будет использован. Варианты выбора приводятся ниже.

Все значения воспринимаются с учётом регистра и должны быть записаны в нижнем регистре:

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.

ldap – проводит аутентификацию, используя сервер LDAP.

radius – проводит аутентификацию, используя сервер RADIUS.

cert – проводит аутентификацию, используя клиентский сертификат SSL.

pam – проводит аутентификацию, используя службу подключаемых модулей аутентификации (PAM), предоставляемую ОС;

reject – безусловно отклоняет подключение. Используется для «фильтрации» некоторых узлов из группы, например строка reject может запретить конкретному узлу подключение, тогда как следующая строка разрешает подключения для остальных узлов этой сети;

– *параметры-аутентификации* – после поля метод-аутентификации может идти поле (поля) вида имя=значение, определяющее параметры метода аутентификации.

Файлы, включённые в конструкции и начинающиеся с @, читаются, как список имён, разделённых запятыми или пробелами. Для этих файлов допустимы вложенные @ конструкции и комментария после знака #.

Файл pg_hba.conf читается при запуске PG360, либо при получении сигнала SIGHUP. При редактировании файла во время работы, необходимо послать серверу PG360 сигнал SIGHUP, для перечитывания файлов конфигурации – используя pg_ctl reload или вызвав SQL- функцию pg_reload_conf() или выполнив kill -HUP.

Для предварительной проверки изменений в файле pg_hba.conf или для диагностики проблем связанных с применением изменений конфигурации используется системное представление pg_hba_file_rules.

Строки в этом представлении, содержащие в поле error не NULL, указывают на проблемы в соответствующих строках файла.

3.4.2. Методы аутентификации

Набор методов аутентификации пользователей в PG360:

– аутентификация password, требующая ввода пароля пользователем;

– аутентификация LDAP, работающая с сервером аутентификации LDAP;

– аутентификация RADIUS, работающая с сервером аутентификации RADIUS;

– аутентификация по сертификату, требующая использования клиентами SSL-подключения и реализуемая ПО среды эксплуатации, имеющего сертификат соответствия;

– аутентификация PAM, реализуемая с использованием библиотеки PAM.

3.4.3. Аутентификация password

Существует несколько методов аутентификации по паролю. Они работают примерно одинаково, но различаются тем, как пароли пользователей хранятся на сервере и как пароль передаётся от клиента по каналу связи.

– scram-sha-256. С методом scram-sha-256 выполняется аутентификация SCRAM-SHA-256. Она производится по схеме вызов-ответ, которая предотвращает перехват паролей через недоверенные соединения и поддерживает хранение паролей на сервере в виде криптографического хеша, что считается безопасным.

Это наиболее безопасный из существующих на данный момент методов, но он не поддерживается старыми клиентскими библиотеками.

– md5. Для метода md5 реализован менее безопасный механизм вызов-ответ. Он предотвращает перехват паролей и предусматривает хранение паролей на сервере в зашифрованном виде, но не защищает в случае похищения хешей паролей с сервера. Метод md5 несовместим с функциональностью db_user_namespace.

Для облегчения перехода от метода md5 к более новому методу SCRAM, если в качестве метода аутентификации в файле pg_hba.conf указан md5, но пароль пользователя на сервере зашифрован для SCRAM, автоматически будет производиться аутентификация на базе SCRAM.

– password. С методом password пароль передаётся в открытом виде и поэтому является уязвимым для атак с перехватом трафика.

Его следует избегать всегда, если это возможно.

Однако если подключение защищено SSL, метод password может быть безопасен. (Хотя аутентификация по сертификату SSL может быть лучшим выбором когда используется SSL).

Пароли баз данных PG360 отделены от паролей пользователей ОС. Пароли всех пользователей базы данных хранятся в системном каталоге pg_authid.

Управлять паролями можно либо используя SQL-команды CREATE ROLE и ALTER ROLE.

Если пароль для пользователя не задан, вместо него хранится NULL, и пройти аутентификацию по паролю этот пользователь не сможет.

Доступность различных методов аутентификации по паролю зависит от того, как пароли пользователей шифруются на сервере (или, говоря точнее, хешируются). Это определяется параметром конфигурации password_encryption в момент назначения пароля. Если пароль шифруется в режиме scram-sha-256, его можно будет использовать для методов аутентификации scram-sha-256 и password (но в последнем случае он будет передаваться открытым текстом). В случае указания метода аутентификации md5 при этом произойдёт автоматический переход к использованию scram-sha-256. Если пароль шифруется в режиме md5, его можно будет использовать только для методов аутентификации md5 и password (и в последнем случае он так же будет передаваться открытым текстом). Хэши паролей, хранящихся в БД, находятся в системном каталоге pg_authid .

3.4.4. Аутентификация LDAP

Данный метод аутентификации работает сходным с методом password образом, за исключением того, что он использует LDAP как метод подтверждения пароля. LDAP используется только для подтверждения пары «имя пользователя/пароль». Поэтому пользователь должен уже существовать в базе данных до того, как для аутентификации будет использован LDAP.

Аутентификация LDAP может работать в двух режимах. Первый режим называется простое связывание. В ходе аутентификации сервер связывается с характерным именем, составленным следующим образом: prefix username suffix. Обычно, параметр prefix используется для указания cn= или DOMAIN\ в среде Active Directory. suffix используется для указания оставшейся части DN или в среде, отличной от Active Directory.

Во втором режиме, который называется поиск+связывание, сервер сначала связывается с каталогом LDAP с предопределённым именем пользователя и паролем, указанным в ldapbinddn и ldapbindpasswd, и выполняет поиск пользователя, пытающегося подключиться к базе данных. Если имя пользователя и пароль не определены, сервер пытается связаться с каталогом анонимно. Поиск выполняется в поддереве ldapbasedn, при этом проверятся точное соответствие имени пользователя атрибуту ldapsearchattribute. Как только при поиске находится пользователь, сервер отключается и заново связывается с каталогом уже как этот пользователь, с паролем, переданным клиентом, чтобы удостовериться, что учётная запись корректна. Этот же режим используется в схемах LDAP-аутентификации в другом программном обеспечении, например, в pam_ldap и mod_authnz_ldap в Apache. Данный вариант даёт больше гибкости в выборе расположения объектов пользователей, но при этом требует дважды подключаться к серверу LDAP.

3.4.5. Аутентификация RADIUS

Данный метод аутентификации работает сходным с методом password образом, за исключением того, что он использует RADIUS как метод проверки пароля. RADIUS используется только для подтверждения пары «имя пользователя/пароль». Поэтому пользователь должен уже существовать в базе данных до того, как для аутентификации будет использован RADIUS.

3.4.6. Аутентификация по сертификату

Для аутентификации в рамках этого метода используется клиентский сертификат SSL, поэтому данный способ применим только для SSL-подключений. Когда используется этот метод, сервер потребует от клиента предъявления действительного и доверенного сертификата. Пароль у клиента не запрашивается. Атрибут cn (Обычное имя) сертификата сравнивается с запрашиваемым именем пользователя базы данных, и если они соответствуют, вход разрешается. Если cn отличается от имени пользователя базы данных, то может быть использовано сопоставление имён пользователей.

3.4.7. Аутентификация PAM

Данный метод аутентификации работает подобно методу password, но использует в качестве механизма проверки подлинности PAM (Pluggable Authentication Modules, Подключаемые модули аутентификации). По умолчанию имя службы PAM — postgresql. PAM используется только для проверки пар «имя пользователя/пароль» и может дополнительно проверять имя или IP-адрес удалённого компьютера. Поэтому пользователь должен уже существовать в базе данных, чтобы PAM можно было использовать для аутентификации.

3.5. Роли базы данных

PG360 использует концепцию ролей (*roles*) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как пользователя базы данных или как группу пользователей, в зависимости от того, как роль настроена. Роли могут владеть объектами базы данных (например, таблицами и функциями) и выдавать другим ролям разрешения на доступ к этим объектам, управляя тем, кто имеет доступ и к каким объектам. Кроме того, можно предоставить одной роли *членство* в другой роли, таким образом одна роль может использовать права других ролей.

Концепция ролей включает в себя концепцию пользователей («users») и групп («groups»). Любая роль может использоваться в качестве пользователя, группы, и того и другого.

3.5.1. Роли базы данных

Роли базы данных отличаются от пользователей ОС. 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;
```

Здесь *имя* соответствует правилам именования идентификаторов SQL: либо обычное, без специальных символов, либо в двойных кавычках.

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, рассмотрите pg_roles системного каталога:

```
SELECT rolname FROM pg_roles;
```

Метакоманда \du программы psql также полезна для получения списка существующих ролей.

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 ОС, 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. (Поэтому клиент не ограничен только ролью, соответствующей имени пользователя ОС, так же как и имя для входа может не соответствовать реальному имени.) Так как с ролью связан набор прав, доступных для клиента, в многопользовательской среде распределять права следует с осторожностью.

3.5.2. Атрибуты ролей

Роль базы данных может иметь атрибуты, определяющие её полномочия и взаимодействие с системой аутентификации клиентов:

– Право подключения. Только роли с атрибутом LOGIN могут использоваться для начального подключения к базе данных. Роль с атрибутом LOGIN можно рассматривать как пользователя базы данных. Для создания такой роли можно использовать любой из вариантов:

`CREATE ROLE имя LOGIN; или CREATE USER имя;`

– Статус суперпользователя. Суперпользователь базы данных обходит все проверки прав доступа, за исключением права на вход в систему. Это опасная привилегия и она не должна использоваться небрежно. Лучше всего выполнять большую часть работы не как суперпользователь. Для создания нового суперпользователя используется команда:

`CREATE ROLE имя SUPERUSER`

Команду нужно выполнить из под роли, которая также является суперпользователем.

– Создание базы данных. Роль должна явно иметь разрешение на создание базы данных (за исключением суперпользователей, которые пропускают все проверки). Для создания такой роли используется команда:

`CREATE ROLE имя CREATEDB`

– Создание роли. Роль должна явно иметь разрешение на создание других ролей (за исключением суперпользователей, которые пропускают все проверки). Для создания такой роли используется команда:

`CREATE ROLE имя CREATEROLE`

Роль с правом CREATEROLE может не только создавать, но и изменять и удалять другие роли, а также выдавать и отзывать членство в ролях. Однако для создания, изменения, удаления ролей суперпользователей и изменения членства в них требуется иметь статус суперпользователя; права CREATEROLE в таких случаях недостаточно.

– Запуск репликации. Роль должна иметь явное разрешение на запуск потоковой репликации (за исключением суперпользователей, которые пропускают все проверки). Роль, используемая для потоковой репликации, также должна иметь атрибут LOGIN. Для создания такой роли используется команда:

```
CREATE ROLE имя REPLICATION LOGIN
```

– Пароль. Пароль имеет значение, если метод аутентификации клиентов требует, чтобы пользователи предоставляли пароль при подключении к базе данных. Методы аутентификации password и md5 используют пароли. База данных и ОС используют раздельные пароли. Пароль указывается при создании роли:

```
CREATE ROLE имя PASSWORD 'строка'.
```

– Срок действия пароля. Предложение VALID UNTIL устанавливает дату и время, после которого пароль роли перестаёт действовать. Если это предложение отсутствует, срок действия пароля будет неограниченным. Для задания срока действия пароля используется синтаксис:

```
VALID UNTIL 'дата_время'
```

IN ROLE - Указывает одну или несколько существующих ролей (групп), членом которых является новая роль:

```
IN ROLE имя_роли
```

– Наследование прав. По умолчанию роль получает разрешение наследовать права той роли, членом которой она является. Однако можно создать роль и без такого разрешения, выполнив команду:

```
CREATE ROLE имя NOINHERIT
```

– Игнорирование защиты на уровне строк. Разрешение обходить все политики защиты на уровне строк (RLS) нужно давать роли явно. Создать роль с таким разрешением можно, выполнив команду от имени суперпользователя:

```
CREATE ROLE имя BYPASSRLS
```

– Ограничение соединений. Для роли можно ограничить число одновременных соединений. -1 (по умолчанию) означает отсутствие ограничения. Для создания роли с ограничением используется команда:

```
CREATE ROLE name CONNECTION LIMIT 'число'
```

Атрибуты ролей могут быть изменены после создания командой ALTER ROLE.

На уровне ролей можно устанавливать многие конфигурационные параметры времени выполнения.

3.5.3. Членство в роли

Часто бывает удобно сгруппировать пользователей для упрощения управления правами: права можно выдавать для всей группы и у всей группы забирать. В PG360для этого создаётся роль, представляющая группу, а затем членство в этой группе выдаётся ролям индивидуальных пользователей.

Для настройки групповой роли сначала нужно создать саму роль:

```
CREATE ROLE имя;
```

Обычно групповая роль не имеет атрибута LOGIN.

После того как групповая роль создана, в неё можно добавлять или отзывать членов, используя команды GRANT и REVOKE:

```
GRANT групповая_роль TO роль1, ... ;
REVOKE групповая_роль FROM роль1, ... ;
```

Членом роли может быть и другая групповая роль (потому что в действительности нет никаких различий между групповыми и не групповыми ролями). При этом база данных не допускает замыкания членства по кругу. Также не допускается управление членством роли PUBLIC в других ролях.

Члены групповой роли могут использовать её права двумя способами. Во-первых, каждый член группы может явно выполнить SET ROLE, чтобы временно «стать» групповой ролью. В этом состоянии сеанс базы данных использует полномочия групповой роли вместо оригинальной роли, под которой был выполнен вход в систему. При этом для всех создаваемых объектов базы данных владельцем считается групповая, а не оригинальная роль. Во-вторых, роли, имеющие атрибут INHERIT, автоматически используют права всех ролей, членами которых они являются, в том числе и унаследованные этими ролями права.

Атрибуты роли LOGIN, SUPERUSER, CREATEDB и CREATEROLE можно рассматривать как особые права, но они никогда не наследуются как обычные права на объекты базы данных. Чтобы ими воспользоваться, необходимо переключиться на роль, имеющую этот атрибут, с помощью команды SET ROLE.

Для удаления групповой роли используется DROP ROLE:

```
DROP ROLE имя;
```

Любое членство в групповой роли будет автоматически отозвано (в остальном на членов этой роли это никак не повлияет).

3.5.4. Удаление ролей

Так как роли могут владеть объектами баз данных и иметь права доступа к объектам других, удаление роли не сводится к немедленному действию DROP ROLE. Сначала должны быть удалены и переданы другим владельцами все объекты, принадлежащие роли; также должны быть отозваны все права, данные роли.

После того как все ценные объекты будут переданы новым владельцам, все оставшиеся объекты, принадлежащие удаляемой роли, могут быть удалены с помощью команды:

```
DROP OWNED
```

И эта команда не может обращаться к объектам в других базах данных, так что её нужно запускать в каждой базе, которая содержит объекты, принадлежащие роли. DROP OWNED не удаляет табличные пространства или базы данных целиком, так что это необходимо сделать вручную, если роли принадлежат базы или табличные пространства, не переданные новым владельцам.

DROP OWNED также удаляет все права, которые даны целевой роли для объектов, не принадлежащих ей. Так как REASSIGN OWNED такие объекты не затрагивает, обычно необходимо запустить и REASSIGN OWNED, и DROP OWNED (в этом порядке!), чтобы полностью ликвидировать зависимости удаляемой роли.

При попытке выполнить DROP ROLE для роли, у которой сохраняются зависимые объекты, будут выданы сообщения, говорящие, какие объекты нужно передать другому владельцу или удалить.

3.5.5. Предопределённые роли

В PG360 имеется набор предопределённых ролей, которые дают доступ к некоторым часто востребованным, но не общедоступным функциям и данным. Администраторы могут назначать (GRANT) эти роли пользователям и/или ролям в своей среде, таким образом открывая этим пользователям доступ к указанной функциональности и информации.

Предопределённые роли и разрешаемый им доступ:

– pg_read_all_settings – читать все конфигурационные переменные, даже те, что обычно видны только суперпользователям;

– pg_read_all_stats – читать все представления pg_stat_* и использовать различные расширения, связанные со статистикой, даже те, что обычно видны только суперпользователям;

– pg_stat_scan_tables – выполнять функции мониторинга, которые могут устанавливать блокировки ACCESS SHARE в таблицах, возможно, на длительное время;

– pg_monitor – читать/выполнять различные представления и функции для мониторинга. Эта роль включена в роли pg_read_all_ settings , pg_read_all_stats и pg_stat_scan_tables;

– pg_signal_backend – передавать другим обслуживающим процессам сигналы для отмены запроса или завершения сеанса;

– pg_read_server_files – читать файлы в любом месте файловой системы, куда имеет доступ СУБД на сервере, выполняя COPY и другие функции работы с файлами;

– pg_write_server_files – записывать файлы в любом месте файловой системы, куда имеет доступ СУБД на сервере, выполняя COPY и другие функции работы с файлами;

– pg_execute_server_program – выполнять программы на сервере (от имени пользователя, запускающего СУБД) так же, как это делает команда COPY и другие функции, выполняющие программы на стороне сервера.

Роли pg_monitor, pg_read_all_settings, pg_read_all_stats и pg_stat_scan_tables созданы для того, чтобы администраторы могли легко настроить роль для мониторинга сервера БД. Эти роли наделяют своих членов набором общих прав, позволяющих читать различные полезные параметры конфигурации, статистику и другую системную информацию, что обычно доступно только суперпользователям.

Роль pg_signal_backend предназначена для того, чтобы администраторы могли давать доверенным, но не имеющим прав суперпользователя ролям право посылать сигналы другим обслуживающим процессам. В настоящее время эта роль позволяет передавать сигналы для отмены запроса, выполняющегося в другом процессе, или для завершения сеанса. Однако пользователь, включённый в эту роль, не может отправлять сигналы процессам, принадлежащим суперпользователям.

Роли pg_read_server_files, pg_write_server_files и pg_execute_server_program предназначены для того, чтобы администраторы могли выделить доверенные, но не имеющие права суперпользователей роли для доступа к файлам и запуска программ на сервере БД от имени пользователя, запускающего СУБД. Так как эти роли могут напрямую обращаться к любым файлам в файловой системе сервера, они обходят все проверки разрешений на уровне базы данных, а значит, воспользовавшись ими, можно получить права суперпользователя. Поэтому назначать их пользователям следует со всей осторожностью.

Управлять членством в этих ролях следует осмотрительно, чтобы они использовались только по необходимости и только с пониманием, что они открывают доступ к закрытой информации.

Администраторы могут давать пользователям доступ к этим ролям, используя команду GRANT. Например:

```
GRANT pg_signal_backend TO admin_user;
```

3.6. Управление базами данных

Каждый работающий экземпляр сервера PG360 обслуживает одну или несколько баз данных. Поэтому базы данных представляют собой вершину иерархии SQL-объектов («объектов базы данных»).

Некоторые объекты, включая роли, базы данных и табличные пространства, определяются на уровне кластера и сохраняются в табличном пространстве pg_global. Внутри кластера существуют базы данных, которые отделены друг от друга, но могут обращаться к объектам уровня кластера. Внутри каждой базы данных имеются схемы, содержащие такие объекты, как таблицы и функции. Таким образом, полная иерархия выглядит следующим образом: кластер, база данных, схема, таблица (или иной объект, например функция).

При подключении к серверу баз данных клиент должен указать имя базы в запросе подключения. Обращаться к нескольким базам через одно подключение нельзя, однако клиенты могут открыть несколько подключений к одной базе или к разным. Безопасность на уровне базы обеспечивают две составляющие: управление подключениями, которое осуществляется на уровне соединения, и управление доступом к объектам, для которого реализована система прав. Обёртки сторонних данных позволяют создать в одной базе данных объекты, скрывающие за собой объекты в других базах или кластерах. Подобную же функциональность предоставляет и более старый модуль dblink. По умолчанию все пользователи могут подключаться ко всем базам данных, используя все методы подключения.

В случаях, когда в кластере PG360 планируется размещать данные несвязанных проектов или с ним будут работать пользователи, которые в принципе не должны никак взаимодействовать, рекомендуется использовать отдельные базы данных и организовать управление подключением и доступом к объектам соответствующим образом. Если же проекты или пользователи взаимосвязаны и должны иметь возможность использовать ресурсы друг друга, они должны размещаться в одной базе данных, но, возможно, в отдельных схемах. Таким образом будет создана модульная структура с изолированными пространствами имён и управлением доступа.

3.6.1. Создание базы данных

Для создания базы данных сервер PG360должен 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.

Создание баз данных это привилегированная операция.

Поскольку для выполнения команды CREATE DATABASE необходимо подключение к серверу базы данных, возникает вопрос как создать самую первую базу данных. Первая база данных всегда создаётся командой initdb при инициализации пространства хранения данных. Эта база данных называется postgres. Далее для создания первой «обычной» базы данных можно подключиться к postgres.

Вторая база данных template1, также создаётся во время инициализации кластера. При каждом создании новой базы данных в рамках кластера по факту производится клонирование шаблона template1. При этом любые изменения, сделанные в template1 распространяются на все созданные впоследствии базы данных. Следует избегать создания объектов в template1, за исключением ситуации, когда их необходимо автоматически добавлять в новые базы.

Для удобства, есть программа командной строки для создания баз данных, createdb:

```
createdb dbname
```

Программа createdb подключается к базе данных postgres и выполняет ранее описанную SQL-команду CREATE DATABASE. Команда createdb без параметров создаст базу данных с именем текущего пользователя.

Для создания базы данных для другого пользователя и назначения его владельцем, чтобы он мог конфигурировать и управлять ею, необходимо выполнить одну из команд:

```
CREATE DATABASE имя_базы OWNER имя_роли;
```

из среды SQL, или:

```
createdb -O имя_роли имя_базы
```

из командной строки ОС. Лишь суперпользователь может создавать базы данных для других (для ролей, членом которых он не является).

3.6.2. Шаблоны баз данных

По факту команда CREATE DATABASE выполняет копирование существующей базы данных. По умолчанию копируется стандартная системная база template1. Таким образом, template1 это шаблон, на основе которого создаются новые базы. Если добавить объекты в template1, то впоследствии они будут копироваться в новые базы данных. Это позволяет внести изменения в стандартный набор объектов. Например, если в template1 установить процедурный язык PL/Perl, то он будет доступен в новых базах без дополнительных действий.

Также существует вторая системная база template0. При инициализации она содержит те же самые объекты, что и template1, предопределённые в рамках устанавливаемой версии PG360. В template0 не следует вносить никакие изменения после инициализации кластера. Если в команде CREATE DATABASE указать в качестве шаблона template0 вместо template1, можно получить «чистую» пользовательскую базу данных (в которой никаких пользовательских объектов нет, есть только системные объекты в первозданном виде), не содержащую ничего, что могло быть добавлено на месте в template1. Это особенно полезно при восстановлении дампа pg_dump: скрипт выгруженного дампа должен восстанавливаться в чистую базу, чтобы он мог воссоздать нужное содержимое базы, избежав конфликтов с объектами, которые могли быть добавлены в template1.

Другая причина, для копирования template0 вместо template1 заключается в том, что можно указать новые параметры локали и кодировку при копировании template0, в то время как для

копий template1 они не должны меняться. Это связано с тем, что template1 может содержать данные в специфических кодировках и локалях, в отличие от template0.

Для создания базы данных на основе template0 необходимо выполнить одну из команд:

```
CREATE DATABASE dbname TEMPLATE template0;
```

из среды SQL, или:

```
createdb -T template0 dbname
```

из командной строки ОС.

Можно создавать дополнительные шаблоны баз данных, и, более того, можно копировать любую базу данных кластера, если указать её имя в качестве шаблона в команде CREATE DATABASE. При копировании все сессии к копируемой базе данных должны быть закрыты. CREATE DATABASE выдаст ошибку, если есть другие подключения; во время операции копирования новые подключения к этой базе данных не разрешены.

В таблице pg_database есть два полезных флага для каждой базы данных: столбцы datistemplate и datallowconn. datistemplate указывает на факт того, что база данных может выступать в качестве шаблона в команде CREATE DATABASE. Если флаг установлен, то для пользователей с правом CREATEDB клонирование доступно; если флаг не установлен, то лишь суперпользователь и владелец базы данных могут её клонировать. Если datallowconn не установлен, то новые подключения к этой базе не допустимы (однако текущие сессии не закрываются при сбросе этого флага). База template0 обычно помечена как datallowconn = false для избежания любых её модификаций. И template0, и template1 всегда должны быть помечены флагом datistemplate = true.

3.6.3. Конфигурирование баз данных

Сервер PG360 имеет множество параметров конфигурации времени выполнения. Можно выставить специфичные для базы данных значения по умолчанию.

Например, если необходимо выключить GEQO оптимизатор в какой-то из баз, то можно, либо выключить его для всех баз данных одновременно, либо убедиться, что все клиенты заботятся об этом, выполняя команду SET geqo TO off. Для того чтобы это действовало по умолчанию в конкретной базе данных, необходимо выполнить команду:

```
ALTER DATABASE mydb SET geqo TO off;
```

Установка сохраняется, но не применяется тотчас. В последующих подключениях к этой базе данных, эффект будет таким, будто перед началом сессии была выполнена команда SET geqo TO off;. Стоит обратить внимание, что пользователь по-прежнему может изменять этот параметр во время сессии; ведь это просто значение по умолчанию.

Чтобы сбросить такое установленное значение, необходимо выполнить команду:

```
ALTER DATABASE dbname RESET varname;
```

3.6.4. 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.

Невозможно выполнить команду DROP DATABASE пока существует хоть одно подключение к заданной базе. Однако можно подключиться к любой другой, в том числе и template1. template1 может быть единственной возможностью при удалении последней пользовательской базы данных кластера.

Также существует программа командной строки для удаления баз данных dropdb:

```
dropdb dbname
```

(В отличие от команды createdb программа не использует имя текущего пользователя по умолчанию).

3.6.5. Табличные пространства

Табличные пространства в PG360 позволяют администраторам организовать логику размещения файлов объектов базы данных в файловой системе. К однажды созданному табличному пространству можно обращаться по имени на этапе создания объектов.

Табличные пространства позволяют администратору управлять дисковым пространством для инсталляции PG360. Это полезно по двум причинам. Во-первых, это нехватка места в разделе, на котором был инициализирован кластер и невозможность его расширения. Табличное пространство можно создать в другом разделе и использовать его до тех пор, пока не появится возможность переконфигурирования системы.

Во-вторых, табличные пространства позволяют администраторам оптимизировать производительность согласно бизнес-процессам, связанным с объектами базы данных.

Для создания табличного пространства используется команда CREATE TABLESPACE, например:

```
CREATE TABLESPACE fastspace LOCATION '/ssd1/postgresql/data';
```

Каталог должен существовать, быть пустым и принадлежать пользователю ОС, под которым запущен PG360. Все созданные впоследствии объекты, принадлежащие целевому табличному пространству, будут храниться в файлах расположенных в этом каталоге. Каталог не должен размещаться на съёмных или устройствах временного хранения, так как кластер может перестать функционировать из-за потери этого пространства.

Создавать табличное пространство должен суперпользователь базы данных, но после этого можно разрешить обычным пользователям его использовать. Для этого необходимо предоставить привилегию CREATE на табличное пространство.

Таблицы, индексы и целые базы данных могут храниться в отдельных табличных пространствах. Для этого пользователь с правом CREATE на табличное пространство должен указать его имя в качестве параметра соответствующей команды. Например, далее создаётся таблица в табличном пространстве space1:

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

Также можно использовать параметр default_tablespace:

```
SET default_tablespace = space1; CREATE TABLE foo(i int);
```

Когда default_tablespace имеет значение отличное от пустой строки, он будет использоваться неявно в качестве значения параметра TABLESPACE в командах CREATE TABLE и CREATE INDEX, если в самой команде не задано иное.

Существует параметр temp_tablespaces, который указывает на размещение временных таблиц и индексов, а также файлов, создаваемых, например, при операциях сортировки больших

наборов данных. Предпочтительнее, в качестве значения этого параметра, указывать не одно имя, а список из нескольких табличных пространств. Это поможет распределить нагрузку, связанную с временными объектами, по различным табличным пространствам. При каждом создании временного объекта будет случайным образом выбираться имя из указанного списка табличных пространств.

Табличное пространство, связанное с базой данных, также используется для хранения её системных каталогов. Более того, это табличное пространство используется по умолчанию для таблиц, индексов и временных файлов, создаваемых в базе данных, если не указано иное в выражении TABLESPACE, или переменной default_tablespace, или temp_tablespaces (соответственно). Если база данных создана без указания конкретного табличного пространства, то используется пространство, к которому принадлежит копируемый шаблон.

При инициализации кластера автоматически создаются два табличных пространства. Табличное пространство pg_global используется для общих системных каталогов. Табличное пространство pg_default используется по умолчанию для баз данных template1 и template0 (в свою очередь, также является пространством по умолчанию для других баз данных, пока не будет явно указано иное в выражении TABLESPACE команды CREATE DATABASE).

После создания, табличное пространство можно использовать в рамках любой базы данных, при условии, что у пользователя имеются необходимые права. Это означает, что табличное пространство невозможно удалить до тех пор, пока не будут удалены все объекты баз данных, использующих это пространство.

Для удаления пустого табличного пространства необходимо выполнить команду:

```
DROP TABLESPACE
```

Чтобы получить список табличных пространств можно сделать запрос к системному каталогу: pg_tablespace, например, `SELECT spcname FROM pg_tablespace;`

Метакоманда \db программы psql также позволяет отобразить список существующих табличных пространств.

Используются символические ссылки для упрощения реализации табличных пространств. Это означает, что табличные пространства могут использоваться только в системах, поддерживающих символические ссылки.

Каталог $PGDATA/pg_tblspc содержит символические ссылки, которые указывают на внешние табличные пространства кластера. Хоть и не рекомендуется, но возможно регулировать табличные пространства вручную, переопределяя эти ссылки. Ни при каких обстоятельствах эти операции нельзя проводить, пока запущен сервер баз данных.

3.6.6. Привилегии

3.6.6.1. Каждый вид объектов имеет разный набор привилегий.

3.6.6.1.1. Привилегии для таблиц: SELECT — чтение данных; INSERT — вставка данных; UPDATE — изменение строк; REFERENCES — внешний ключ (право ссылаться на таблицу); DELETE — удаление строк; TRUNCATE — очистка таблицы; TRIGGER — создание триггеров.

3.6.6.1.2. Привилегии для представлений: SELECT — право читать представление; TRIGGER — право создавать триггеры.

3.6.6.1.3. Привилегии для последовательности: SELECT — право читать последовательность; UPDATE — право изменять последовательность; USAGE — право использовать последовательность.

3.6.6.1.4. Привилегии для табличных пространств:CREATE — разрешает создавать объекты внутри табличного пространства.

3.6.6.1.5. Привилегии для базы данных: CREATE — разрешает создавать схемы внутри базы данных; CONNECT — даёт возможность подключаться к базе данных; TEMPORARY — разрешает создавать в базе данных временные таблицы.

3.6.6.1.6. Привилегии для схем: CREATE — разрешает создавать объекты внутри конкретной схемы; USAGE — позволяет использовать объекты в конкретной схеме.

3.6.6.1.7. Привилегии для функций: EXECUTE — даёт право выполнять функцию.

3.6.6.2. Для назначения привилегии необходимо вызвать команду GRANT:

```
GRANT <привилегии> ON <объект> to <роль>;
```

3.6.6.3. Для отозва привилегии необходимо вызвать команду REVOKE:

```
REVOKE <привилегии> ON <объект> FROM <роль>;
```

3.6.6.4.Выдача привилегии с правом её передачи выполняется с помощью WITH GRAND OPTION:

```
GRANT <привилегии> ON <объект> to <роль> WITH GRAND OPTION;
```

3.6.6.5. Если привилегия назначена вместе с правом её передачи. А затем роль воспользовалась своим правом и передала привилегию другим ролям. То отозвать эту привилегию можно только каскадно у этой роли и у других ролей с помощью CASCADE:

```
REVOKE <привилегии> ON <объект> FROM <роль> CASCADE;
```

3.6.6.6. Для отзыва только права привилегии необходимо выполнить команду:

```
REVOKE GRANT OPTION FOR <привилегии> ON <объект> FROM <роль>;
```

3.6.6.7. Роль получает привилегии своих групповых ролей. Нужно ли ей будет для получения привилегий выполнять SET ROLE зависит от атрибута роли: INHERIT — атрибут роли, который включает автоматическое наследование привилегий; NOINHERIT — атрибут роли, который требует явное выполнение SET ROLE.

3.6.6.8. Привилегии по умолчанию — это такие привилегии, которые добавятся к каким-то ролям на объект при его создании.

Привилегии по умолчанию создаются командой ALTER DEFAULT PRIVILEGES:

```
ALTER DEFAULT PRIVILEGES [IN SCHEMA <схема>] GRANT <привилегии>
ON <класс_объектов> TO <роль>
```

где <класс_объектов> это может быть, например таблица, функция, представление и т.п.

Для удаления привилегий по умолчанию необходимо вызвать команду:

```
ALTER DEFAULT PRIVILEGES [IN SCHEMA <схема>] REVOKE <привилегии>
ON <класс_объектов> FROM <роль>
```

3.7. Локализация

PG360 поддерживает два средства локализации:

– использование средств локализации ОС 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.

3.7.1. Поддержка языковых стандартов

Поддержка *языковых стандартов* в приложениях относится к культурным предпочтениям, которые касаются алфавита, порядка сортировки, форматирования чисел и т. п. PG360 использует соответствующие стандартам ISO C и POSIX возможности локали, предоставляемые ОС сервера.

3.7.1.1. Обзор

Поддержка локали автоматически инициализируется, когда кластер базы данных создаётся при помощи программы initdb. Программа initdb инициализирует кластер баз данных по умолчанию с локалью из окружения выполнения. Поэтому, если ОС уже использует локаль, которую необходимо выбрать для кластера баз данных, не требуется дополнительно совершать никаких действий. Если необходимо использовать другую локаль (или точно неизвестно, какая локаль используется в системе), можно указать для initdb, какую именно локаль использовать, задав параметр --locale. Например: `initdb --locale=ru_RU`

Данный пример для Unix-систем задаёт русский язык (ru). Другими вариантами могут быть en_US (американский английский) и fr_CA (канадский французский). Если в языковом окружении может использоваться более одного набора символов, значение может принимать вид *language_territory.codeset*. Например, fr_BE.UTF-8 обозначает французский язык (fr), на котором говорят в Бельгии (BE), с кодировкой UTF-8.

То, какие локали и под какими именами доступны в системе, зависит от того, что было включено в ОС производителем и что из этого было установлено. В большинстве Unix-систем команда locale -a выведет список доступных локалей.

Иногда целесообразно объединить правила из различных локалей, например, использовать английские правила сравнения и испанские сообщения. Для этой цели существует набор категорий локали, каждая из которых управляет только определёнными аспектами правил локализации:

LC_COLLATE	Порядок сортировки строк
LC_CTYPE	Классификация символов
LC_MESSAGES	Язык сообщений
LC_MONETARY	Форматирование валютных сумм
LC_NUMERIC	Форматирование чисел
LC_TIME	Форматирование даты и времени

Эти имена категорий initdb принимает в качестве имён соответствующих параметров, позволяющих переопределить выбор локали в определённой категории.

Если необходимо, чтобы система работала без языковой поддержки, нужно использовать специальное имя локали C либо эквивалентное ему POSIX.

Значения некоторых категорий локали должны быть заданы при создании базы данных. Можно использовать различные параметры локали для различных баз данных, но после создания базы уже нет возможности изменить их для этой базы данных. LC_COLLATE и LC_CTYPE являются этими категориями. Они влияют на порядок сортировки в индексах, поэтому они должны быть зафиксированы, иначе индексы на текстовых столбцах могут повредиться. Значения

по умолчанию для этих категорий определяются при запуске initdb, и эти значения используются при создании новых баз данных, если другие значения не указаны явно в команде CREATE DATABASE.

Прочие категории локали можно изменить в любое время, настроив параметры конфигурации сервера, которые имеют такое же имя как и категории локали. Значения, выбранные через initdb, фактически записываются лишь в файл конфигурации postgresql.conf, чтобы использоваться по умолчанию при запуске сервера. Если удалить эти значения из postgresql.conf, сервер получит соответствующие значения из своей среды выполнения.

Необходимо обратить внимание на то, что поведение локали сервера определяется переменными среды, установленными на стороне сервера, а не средой клиента. Таким образом, необходимо правильно сконфигурировать локаль перед запуском сервера. Если же клиент и сервер работают с разными локалями, то сообщения, возможно, будут появляться на разных языках в зависимости от того, где они возникают.

3.7.1.2. Поведение

Локаль влияет на следующий функционал SQL:

– Порядок сортировки в запросах с использованием ORDER BY или стандартных операторах сравнения текстовых данных.

– Функции upper, lower, и initcap.

– Операторы поиска по шаблону (LIKE, SIMILAR TO, и регулярные выражения в стиле POSIX); локаль влияет как на поиск без учёта регистра, так и на классификацию знаков по классам символов регулярных выражений.

– Семейство функций to_char.

– Возможность использовать индексы с предложениями LIKE.

Недостатком использования отличающихся от C или POSIX локалей в PG360 является влияние на производительность. Это замедляет обработку символов и мешает LIKE использовать обычные индексы. По этой причине необходимо использовать локали только в том случае, если они действительно нужны.

3.7.2. Поддержка правил сортировки

Правила сортировки позволяют устанавливать порядок сортировки и особенности классификации символов в отдельных столбцах или даже при выполнении отдельных операций. Это смягчает последствия того, что параметры базы данных LC_COLLATE и LC_CTYPE невозможно изменить после её создания.

3.7.2.1. Основные понятия

Концептуально, каждое выражение с типом данных, к которому применяется сортировка, имеет правила сортировки. (Встроенными сортируемыми типами данных являются text, varchar, и char. Типы, определяемые в базе пользователем, могут также быть отмечены как сортируемые, и, конечно, домен на основе сортируемого типа данных является сортируемым.) Если выражение содержит ссылку на столбец, правила сортировки выражения определяются правилами сортировки столбца. Если выражение — константа, правилами сортировки являются стандартные

правила для типа данных константы. Правила сортировки более сложных выражений являются производной от правил сортировки входящих в него частей, как описано ниже.

Правилами сортировки выражения могут быть правила сортировки «по умолчанию», что означает использование параметров локали, установленных для базы данных. Также возможно, что правила сортировки выражения могут не определиться. В таких случаях операции упорядочивания и другие операции, для которых необходимы правила сортировки, завершатся с ошибкой.

Когда база данных должна выполнить упорядочивание или классификацию символов, она использует правила сортировки выполняемого выражения. Это происходит, к примеру, с предложениями ORDER BY и такими вызовами функций или операторов как <. Правила сортировки, которые применяются в предложении ORDER BY, это просто правила ключа сортировки. Правила сортировки, применяемые к вызову функции или оператора, определяются их параметрами, как описано ниже. В дополнение к операциям сравнения, правила сортировки учитываются функциями, преобразующими регистр символов, такими как lower, upper, и initcap; операторами поиска по шаблону; и функцией to_char и связанными с ней.

При вызове функции или оператора правило сортировки определяется в зависимости от того, какие правила заданы для аргументов во время выполнения данной операции. Если результатом вызова функции или оператора является сортируемый тип данных, правила сортировки также используются во время разбора как определяемые правила сортировки функции или выражения оператора, когда для внешнего выражения требуется знание правил сортировки.

Определение правил сортировки выражения может быть неявным или явным. Это отличие влияет на то, как комбинируются правила сортировки, когда несколько разных правил появляются в выражении. Явное определение правил сортировки возникает, когда используется предложение COLLATE; все прочие варианты являются неявными. Когда необходимо объединить несколько правил сортировки, например, в вызове функции, используются следующие правила:

– Если для одного из выражений-аргументов правило сортировки определено явно, то и для других аргументов явно задаваемое правило должно быть тем же, иначе возникнет ошибка. В случае присутствия явного определения правила сортировки, оно становится результирующим для всей операции.

– В противном случае все входные выражения должны иметь одни и те же неявно определяемые правила сортировки или правила сортировки по умолчанию. Если присутствуют какие– либо правила сортировки, отличные от заданных по умолчанию, получаем результат комбинации правил сортировки. Иначе результатом станут правила сортировки, заданные по умолчанию.

– Если среди входных выражений есть конфликтующие неявные правила сортировки, отличные от заданных по умолчанию, тогда комбинация рассматривается как имеющая неопределённые правила сортировки. Это не является условием возникновения ошибки, если вызываемой конкретной функции не требуются данные о правилах сортировки, которые ей следует применить. Если же такие данные требуются, это приведёт к ошибке во время выполнения.

3.7.2.3. Стандартные правила сортировки

На всех платформах доступны правила сортировки под названием default, C, и POSIX. Дополнительные правила сортировки могут быть доступны в зависимости от поддержки ОС. Правило сортировки default использует значения LC_COLLATE и LC_CTYPE, заданные при создании базы данных. Правила сортировки C и POSIX определяют поведение, характерное для «традиционного C», в котором только знаки кодировки ASCII от «A» до «Z» рассматриваются как буквы, и сортировка осуществляется строго по символьному коду байтов.

Для кодировки UTF8 дополнительно поддерживается имя ucs_basic, определённое в стандарте SQL. Это правило сортировки равнозначно правилу C и производит сортировку по кодам символов Unicode.

3.7.2.4. Предопределённые правила сортировки

Если ОС поддерживает использование нескольких локалей в одной программе (newlocale и связанные функции) или включена поддержка ICU, то при инициализации кластера баз данных программа initdb наполняет системный каталог pg_collation информацией обо всех локалях, которые обнаруживает в этот момент в ОС.

Для просмотра всех имеющихся локалей необходимо выполнить запрос:
`SELECT * FROM pg_collation или команду \dOS+ в psql.`

3.7.2.5. Правила сортировки libc

Стандартный набор правил сортировки, предоставляемый провайдером libc, сопоставляется непосредственно с локалями, установленными в ОС (их можно просмотреть с помощью команды locale -a). В случаях, когда у правила сортировки libc должны быть различные значения LC_COLLATE и LC_CTYPE, или когда в ОС после инициализации СУБД устанавливаются новые локали, создать новое правило сортировки можно с помощью команды CREATE COLLATION. Новые локали ОС можно также импортировать в массовом порядке, воспользовавшись функцией pg_import_system_collations().

В любой базе данных имеют значение только те правила сортировки, которые используют кодировку этой базы данных. Прочие записи в pg_collation игнорируются. Таким образом, усечённое имя правил сортировки, такое как de_DE, может считаться уникальным внутри данной базы данных, даже если бы оно не было уникальным глобально. Использование усечённого имени сортировки рекомендуется, так как при переходе на другую кодировку базы данных придётся выполнить на одно изменение меньше. Однако следует помнить, что правила сортировки default, C и POSIX можно использовать независимо от кодировки базы данных.

В PG360 предполагается, что отдельные объекты правил сортировки несовместимы, даже когда они имеют идентичные свойства. Например,

SELECT a COLLATE "C" < b COLLATE "POSIX" FROM test1;

выведет сообщение об ошибке, несмотря на то, что поведение правил сортировки C и POSIX идентично. По этой причине смешивать усечённые и полные имена правил сортировки не рекомендуется.

3.7.2.6. Правила сортировки ICU

С ICU не представляется разумным перечислять все возможные имена локалей. ICU использует для локалей определённую схему именования, но имён локалей может быть гораздо больше, чем собственно различных локалей. Программа initdb, используя API ICU, извлекает список различных локалей и наполняет начальный набор правил в базе данных. Правила сортировки провайдера ICU создаются с именами, включающими метку языка в формате BCP 47 и указание расширения «для частного использования» (-x-icu), для отличия от локалей libc.

Некоторые (редко используемые) кодировки не поддерживаются ICU. Когда база имеет одну из таких кодировок, записи правил сортировки ICU в pg_collation игнорируются. При попытке использовать их будет выдана ошибка с сообщением вида «правило сортировки "de-x- icu" для кодировки "WIN874" не существует».

3.7.2.7. Создание новых правил сортировки

Если стандартных и предопределённых правил сортировки недостаточно, пользователи могут создавать собственные правила сортировки, используя команду SQL CREATE COLLATION.

Стандартные и предопределённые правила сортировки находятся в схеме pg_catalog, как и все предопределённые объекты. Пользовательские правила сортировки должны создаваться в пользовательских схемах. Помимо прочего, это полезно тем, что их будет выгружать pg_dump.

3.7.2.8. Новые правила сортировки libc

Новые правила сортировки libc могут создаваться следующей командой:

```
CREATE COLLATION german (provider = libc, locale = 'de_DE');
```

Точные значения, которые могут допускаться в предложении locale в этой команде, зависят от ОС. В Unix-подобных системах их список выдаёт команда locale -a.

Так как предопределённый набор правил сортировки libc уже включает все правила сортировки, определённые в ОС в момент инициализации базы данных, необходимость создавать новые правила вручную обычно не возникает. Такая потребность может возникнуть, когда нужно сменить систему именования либо когда ОС была обновлена и в ней появились новые определения локалей (в этом случае см. pg_import_system_collations()).

3.7.2.9. Новые правила сортировки ICU

ICU допускает видоизменения правил сортировки, не ограничивая пользователей наборами язык+страна, которые подготавливает initdb. Пользователи могут свободно создавать собственные объекты-правила сортировки, использующие предоставляемые средства для получения требуемых порядков сортировки. Набор допустимых имён и атрибутов зависит от конкретной версии ICU.

3.7.2.10. Копирование правил сортировки

Команда CREATE COLLATION может также создать новое правило сортировки из существующего, что может быть полезно для использования имён, независимых от ОС, создания

имён для совместимости или использования правил сортировки ICU под более понятными именами. Например:

```
CREATE COLLATION german FROM "de_DE";
CREATE COLLATION french FROM "fr-x-icu";
```

3.7.2.11. Недетерминированные правила сортировки

Правило сортировки может быть либо *детерминированным*, либо *недетерминированным*. С детерминированными правилами сортировки используются детерминированные сравнения, что означает, что сроки считаются равными, только если они состоят из одинаковой последовательности байтов. Недетерминированное же сравнение может признать равными строки, состоящие и из разных байтов. Обычно это требуется при сравнении без учёта регистра или ударения, а также при сравнении строк в различных нормальных формах Unicode. Как именно будут реализованы подобные сравнения, определяется провайдером правил сортировки; флаг детерминированности только отмечает, будет ли вопрос равенства решаться побайтовым сравнением.

Чтобы создать недетерминированное правило сортировки, необходимо указать свойство deterministic = false в команде CREATE COLLATION, например:

```
CREATE COLLATION ndcoll (provider = icu, locale = 'und', deterministic = false);
```

Все стандартные и предопределённые правила сортировки являются детерминированными, и так же детерминированными по умолчанию создаются пользовательские правила. Недетерминированные правила сортировки обеспечивают более «правильное» поведение, особенно в части использования всех возможностей Unicode и обработки множества особых случаев, но они имеют и ряд недостатков. Прежде всего, их применение отрицательно сказывается на производительности. Необходимо обратить внимание, что в B-деревьях с недетерминированным правилами не будет производиться исключение дубликатов. К тому же с недетерминированными правилами сортировки невозможны некоторые операции, например, поиск по шаблону. Поэтому применять их следует только тогда, когда в этом есть явная необходимость.

3.7.3. Поддержка кодировок

Поддержка кодировок в PG360 позволяет хранить текст в различных кодировках, включая однобайтовые кодировки, такие как входящие в семейство ISO 8859 и многобайтовые кодировки, такие как EUC (Extended Unix Code), UTF-8 и внутренний код Mule. Все поддерживаемые кодировки могут прозрачно использоваться клиентами, но некоторые не поддерживаются сервером (в качестве серверной кодировки). Кодировка по умолчанию выбирается при инициализации кластера базы данных PG360 при помощи initdb. Она может быть переопределена при создании базы данных, что позволяет иметь несколько баз данных с разными кодировками.

Важным ограничением, однако, является то, что кодировка каждой базы данных должна быть совместима с параметрами локали базы данных LC_CTYPE (классификация символов) и LC_COLLATE (порядок сортировки строк). Для локали C или POSIX подойдёт любой набор

символов, но для других локалей, предоставляемых библиотекой libc, есть только один набор символов, который будет работать правильно. (Однако в среде Windows кодировка UTF-8 может использоваться с любой локалью.) Если у вас включена поддержка ICU, локали, предоставляемые библиотекой ICU, можно использовать с большинством (но не всеми) кодировками на стороне сервера.

3.7.3.1. Поддерживаемые кодировки

Таблица 2 показывает кодировки, доступные для использования в PG360.

Таблица 2. Кодировки PG360

Имя	**Описание**	**Язык**	**Поддержка на сервере**	**ICU?**	**Байт на символ**	**Псевдонимы**
BIG5	Big Five	Традиционные китайские иероглифы	Нет	Нет	1–2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	Упрощённые китайские иероглифы	Да	Да	1–3	
EUC_JP	Extended UNIX Code-JP	Японский	Да	Да	1–3	
EUC_JIS_2004	Extended UNIX Code-JP, JIS X 0213	Японский	Да	Нет	1–3	
EUC_KR	Extended UNIX Code-KR	Корейский	Да	Да	1–3	
EUC_TW	Extended UNIX Code-TW	Традиционные китайские иероглифы, тайваньский	Да	Да	1–3	
GB18030	Национальный стандарт	Китайский	Нет	Нет	1–4	
GBK	Расширенный национальный стандарт	Упрощённые китайские иероглифы	Нет	Нет	1–2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Латинский/ Кириллица	Да	Да	1	
ISO_8859_6	ISO 8859-6, ECMA 114	Латинский/ Арабский	Да	Да	1	
ISO_8859_7	ISO 8859-7, ECMA 118	Латинский/ Греческий	Да	Да	1	
ISO_8859_8	ISO 8859-8, ECMA 121	Латинский/ Иврит	Да	Да	1	
JOHAB	JOHAB	Корейский (Хангыль)	Нет	Нет	1–3	
KOI8R	KOI8-R	Кириллица (Русский)	Да	Да	1	KOI8
KOI8U	KOI8-U	Кириллица (Украинский)	Да	Да	1	
LATIN1	ISO 8859-1, ECMA 94	Западно европейские	Да	Да	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Центрально европейские	Да	Да	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	Южно европейские	Да	Да	1	ISO88593

Имя	Описание	Язык	Поддержка на сервере	ICU?	Байт на символ	Псевдонимы
LATIN4	ISO 8859-4, ECMA 94	Северо европейские	Да	Да	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Турецкий	Да	Да	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Скандинавские	Да	Да	1	ISO885910
LATIN7	ISO 8859-13	Балтийские	Да	Да	1	ISO885913
LATIN8	ISO 8859-14	Кельтские	Да	Да	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 с европейскими языками и диалектами	Да	Да	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Румынский	Да	Нет	1	ISO885916
MULE_INTERNAL	Внутренний код Mule	Мультиязычны редактор Emacs	йДа	Нет	1–4	
SJIS	Shift JIS	Японский	Нет	Нет	1–2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Японский	Нет	Нет	1–2	
SQL_ASCII	не указан (см. текст)	*any*	Да	Нет	1	
UHC	Унифицированн код Хангыль	ыКойрейский	Нет	Нет	1–2	WIN949, Windows949
UTF8	Unicode, 8-bit	*все*	Да	Да	1–4	Unicode
WIN866	Windows CP866	Кириллица	Да	Да	1	ALT
WIN874	Windows CP874	Тайский	Да	Нет	1	
WIN1250	Windows CP1250	Центрально европейские	Да	Да	1	
WIN1251	Windows CP1251	Кириллица	Да	Да	1	WIN
WIN1252	Windows CP1252	Западно европейские	Да	Да	1	
WIN1253	Windows CP1253	Греческий	Да	Да	1	
WIN1254	Windows CP1254	Турецкий	Да	Да	1	
WIN1255	Windows CP1255	Иврит	Да	Да	1	
WIN1256	Windows CP1256	Арабский	Да	Да	1	
WIN1257	Windows CP1257	Балтийские	Да	Да	1	
WIN1258	Windows CP1258	Вьетнамский	Да	Да	1	ABC, TCVN, TCVN5712, VSCII

Не все клиентские API поддерживают все перечисленные кодировки. Например, драйвер интерфейса JDBC PG360 не поддерживает MULE_INTERNAL, LATIN6, LATIN8 и LATIN10.

Поведение кодировки SQL_ASCII существенно отличается от других. Когда набором символов сервера является SQL_ASCII, сервер интерпретирует байтовые значения 0–127 согласно кодировке ASCII, тогда как значения 128–255 воспринимаются как незначимые. Перекодировка не будет выполнена при выборе SQL_ASCII.

3.7.3.2. Настройка кодировки

Программа initdb определяет кодировку по умолчанию для кластера PG360. Например,

```
initdb -E EUC_JP
```

настраивает кодировку по умолчанию на EUC_JP (Расширенная система кодирования для японского языка). Можно использовать --encoding вместо -E в случае предпочтения более длинных имён параметров. Если параметр -E или --encoding не задан, initdb пытается определить подходящую кодировку в зависимости от указанной или заданной по умолчанию локали.

При создании базы данных можно указать кодировку, отличную от заданной по умолчанию, если эта кодировка совместима с выбранной локалью:

```
createdb -E EUC_KR -T template0 --lc-collate=ko_KR.euckr --lc-ctype=ko_KR.euckr korean
```

Это создаст базу данных с именем korean, которая использует кодировку EUC_KR и локаль ko_KR. Также, получить желаемый результат можно с помощью данной SQL-команды:

```
CREATE DATABASE korean WITH ENCODING 'EUC_KR' LC_COLLATE='ko_KR.euckr'
LC_CTYPE='ko_KR.euckr' TEMPLATE=template0;
```

Приведённые выше команды задают копирование базы данных template0. При копировании любой другой базы данных, параметры локали и кодировку исходной базы изменить нельзя, так как это может привести к искажению данных.

Кодировка базы данных хранится в системном каталоге pg_database. Её можно увидеть при помощи параметра psql -l или команды \l.

3.7.3.3. Автоматическая перекодировка между сервером и клиентом

PG360 поддерживает автоматическое перекодирование символов между сервером и клиентов для многих сочетаний кодировок.

Чтобы включить автоматическую перекодировку символов, необходимо сообщить PG360 кодировку, которую необходимо использовать на стороне клиента. Это можно выполнить несколькими способами:

– Использование команды \encoding в psql. \encoding позволяет оперативно изменять клиентскую кодировку.

– Библиотека libpq имеет функции, для управления клиентской кодировкой.

– Использование команды `SET client_encoding TO`. Клиентская кодировка устанавливается следующей SQL-командой:

```
SET CLIENT_ENCODING TO 'value';
```

Также, можно использовать стандартный синтаксис SQL

```
SET NAMES: SET NAMES 'value';
```

Получить текущую клиентскую кодировку:

```
SHOW client_encoding;
```

Вернуть кодировку по умолчанию:

```
RESET client_encoding;
```

– Использование PGCLIENTENCODING. Если установлена переменная окружения PGCLIENTENCODING, то эта клиентская кодировка выбирается автоматически при подключении к серверу.

– Использование переменной конфигурации client_encoding. Если задана переменная client_encoding, указанная клиентская кодировка выбирается автоматически при подключении к серверу.

Если перекодировка определённого символа невозможна (предположим, выбраны EUC_JP для сервера и LATIN1 для клиента, и передаются некоторые японские иероглифы, не представленные в LATIN1), возникает ошибка.

Если клиентская кодировка определена как SQL_ASCII, перекодировка отключается вне зависимости от кодировки сервера. (Однако если серверная кодировка отлична от SQL_ASCII, сервер будет тем не менее проверять, что входящие данные являются допустимыми для его кодировки; поэтому итоговый результат будет тем же, что и при совпадении клиентской кодировки с серверной.) На сервере же использовать кодировку SQL_ASCII неразумно, кроме случаев, когда все ваши данные полностью вписываются в ASCII.

3.7.3.4. Возможные перекодировки наборов символов

PG360 поддерживает перекодирование между любыми двумя наборами символов, для которых в системном каталоге pg_conversion присутствует функция перекодирования. PG360 включает несколько предопределённых перекодировок, сведённых в Таблице 3 и описанных подробнее в Таблице 4. Кроме этого, есть возможность создать новую перекодировку, используя SQL-команду CREATE CONVERSION. (Чтобы она использовалась для автоматического перекодирования текста между сервером и клиентом, она должна быть помечена как перекодировка «по умолчанию» для своей пары кодировок.)

Таблица 3. Встроенные клиент-серверные перекодировки наборов символов

Серверная кодировка	**Доступные клиентские кодировки**
BIG5	не поддерживается как серверная кодировка
EUC_CN	EUC_CN, MULE_INTERNAL , UTF8
EUC_JP	EUC_JP, MULE_INTERNAL , SJIS, UTF8
EUC_JIS_2004	EUC_JIS_2004, SHIFT_JIS_2004 , UTF8
EUC_KR	EUC_KR, MULE_INTERNAL , UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL , UTF8
GB18030	не поддерживается как серверная кодировка
GBK	не поддерживается как серверная кодировка
ISO_8859_5	ISO_8859_5, KOI8R, MULE_INTERNAL , UTF8, WIN866, WIN1251
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	не поддерживается как серверная кодировка
KOI8R	KOI8R, ISO_8859_5 , MULE_INTERNAL , UTF8, WIN866, WIN1251
KOI8U	KOI8U, UTF8
LATIN1	LATIN1, MULE_INTERNAL , UTF8
LATIN2	LATIN2, MULE_INTERNAL , UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL , UTF8
LATIN4	LATIN4, MULE_INTERNAL , UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8

Серверная кодировка	Доступные клиентские кодировки
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN , EUC_JP , EUC_KR , EUC_TW , ISO_8859_5 , KOI8R, LATIN1 to LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	не поддерживается как серверная кодировка
SHIFT_JIS_2004	не поддерживается как серверная кодировка
SQL_ASCII	любая (перекодировка не будет выполнена)
UHC	не поддерживается как серверная кодировка
UTF8	все поддерживаемые кодировки
WIN866	WIN866, ISO_8859_5 , KOI8R, MULE_INTERNAL , UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL , UTF8
WIN1251	WIN1251, ISO_8859_5 , KOI8R, MULE_INTERNAL , UTF8, WIN866
WIN1252	WIN1252, UTF8
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

Таблица 4. Все встроенные перекодировки наборов символов

Имя преобразования [a]	Исходная кодировка	Целевая кодировка
big5_to_euc_tw	BIG5	EUC_TW
big5_to_mic	BIG5	MULE_INTERNAL
big5_to_utf8	BIG5	UTF8
euc_cn_to_mic	EUC_CN	MULE_INTERNAL
euc_cn_to_utf8	EUC_CN	UTF8
euc_jp_to_mic	EUC_JP	MULE_INTERNAL
euc_jp_to_sjis	EUC_JP	SJIS
euc_jp_to_utf8	EUC_JP	UTF8
euc_kr_to_mic	EUC_KR	MULE_INTERNAL
euc_kr_to_utf8	EUC_KR	UTF8
euc_tw_to_big5	EUC_TW	BIG5
euc_tw_to_mic	EUC_TW	MULE_INTERNAL
euc_tw_to_utf8	EUC_TW	UTF8
gb18030_to_utf8	GB18030	UTF8
gbk_to_utf8	GBK	UTF8
iso_8859_10_to_utf8	LATIN6	UTF8
iso_8859_13_to_utf8	LATIN7	UTF8

Имя преобразования [a]	**Исходная кодировка**	**Целевая кодировка**
iso_8859_14_to_utf8	LATIN8	UTF8
iso_8859_15_to_utf8	LATIN9	UTF8
iso_8859_16_to_utf8	LATIN10	UTF8
iso_8859_1_to_mic	LATIN1	MULE_INTERNAL
iso_8859_1_to_utf8	LATIN1	UTF8
iso_8859_2_to_mic	LATIN2	MULE_INTERNAL
iso_8859_2_to_utf8	LATIN2	UTF8
iso_8859_2_to_windows_1250	LATIN2	WIN1250
iso_8859_3_to_mic	LATIN3	MULE_INTERNAL
iso_8859_3_to_utf8	LATIN3	UTF8
iso_8859_4_to_mic	LATIN4	MULE_INTERNAL
iso_8859_4_to_utf8	LATIN4	UTF8
iso_8859_5_to_koi8_r	ISO_8859_5	KOI8R
iso_8859_5_to_mic	ISO_8859_5	MULE_INTERNAL
iso_8859_5_to_utf8	ISO_8859_5	UTF8
iso_8859_5_to_windows_1251	ISO_8859_5	WIN1251
iso_8859_5_to_windows_866	ISO_8859_5	WIN866
iso_8859_6_to_utf8	ISO_8859_6	UTF8
iso_8859_7_to_utf8	ISO_8859_7	UTF8
iso_8859_8_to_utf8	ISO_8859_8	UTF8
iso_8859_9_to_utf8	LATIN5	UTF8
johab_to_utf8	JOHAB	UTF8
koi8_r_to_iso_8859_5	KOI8R	ISO_8859_5
koi8_r_to_mic	KOI8R	MULE_INTERNAL
koi8_r_to_utf8	KOI8R	UTF8
koi8_r_to_windows_1251	KOI8R	WIN1251
koi8_r_to_windows_866	KOI8R	WIN866
koi8_u_to_utf8	KOI8U	UTF8
mic_to_big5	MULE_INTERNAL	BIG5
mic_to_euc_cn	MULE_INTERNAL	EUC_CN
mic_to_euc_jp	MULE_INTERNAL	EUC_JP
mic_to_euc_kr	MULE_INTERNAL	EUC_KR
mic_to_euc_tw	MULE_INTERNAL	EUC_TW
mic_to_iso_8859_1	MULE_INTERNAL	LATIN1
mic_to_iso_8859_2	MULE_INTERNAL	LATIN2
mic_to_iso_8859_3	MULE_INTERNAL	LATIN3
mic_to_iso_8859_4	MULE_INTERNAL	LATIN4
mic_to_iso_8859_5	MULE_INTERNAL	ISO_8859_5
mic_to_koi8_r	MULE_INTERNAL	KOI8R
mic_to_sjis	MULE_INTERNAL	SJIS
mic_to_windows_1250	MULE_INTERNAL	WIN1250
mic_to_windows_1251	MULE_INTERNAL	WIN1251
mic_to_windows_866	MULE_INTERNAL	WIN866

Имя преобразования [a]	**Исходная кодировка**	**Целевая кодировка**
sjis_to_euc_jp	SJIS	EUC_JP
sjis_to_mic	SJIS	MULE_INTERNAL
sjis_to_utf8	SJIS	UTF8
windows_1258_to_utf8	WIN1258	UTF8
uhc_to_utf8	UHC	UTF8
utf8_to_big5	UTF8	BIG5
utf8_to_euc_cn	UTF8	EUC_CN
utf8_to_euc_jp	UTF8	EUC_JP
utf8_to_euc_kr	UTF8	EUC_KR
utf8_to_euc_tw	UTF8	EUC_TW
utf8_to_gb18030	UTF8	GB18030
utf8_to_gbk	UTF8	GBK
utf8_to_iso_8859_1	UTF8	LATIN1
utf8_to_iso_8859_10	UTF8	LATIN6
utf8_to_iso_8859_13	UTF8	LATIN7
utf8_to_iso_8859_14	UTF8	LATIN8
utf8_to_iso_8859_15	UTF8	LATIN9
utf8_to_iso_8859_16	UTF8	LATIN10
utf8_to_iso_8859_2	UTF8	LATIN2
utf8_to_iso_8859_3	UTF8	LATIN3
utf8_to_iso_8859_4	UTF8	LATIN4
utf8_to_iso_8859_5	UTF8	ISO_8859_5
utf8_to_iso_8859_6	UTF8	ISO_8859_6
utf8_to_iso_8859_7	UTF8	ISO_8859_7
utf8_to_iso_8859_8	UTF8	ISO_8859_8
utf8_to_iso_8859_9	UTF8	LATIN5
utf8_to_johab	UTF8	JOHAB
utf8_to_koi8_r	UTF8	KOI8R
utf8_to_koi8_u	UTF8	KOI8U
utf8_to_sjis	UTF8	SJIS
utf8_to_windows_1258	UTF8	WIN1258
utf8_to_uhc	UTF8	UHC
utf8_to_windows_1250	UTF8	WIN1250
utf8_to_windows_1251	UTF8	WIN1251
utf8_to_windows_1252	UTF8	WIN1252
utf8_to_windows_1253	UTF8	WIN1253
utf8_to_windows_1254	UTF8	WIN1254
utf8_to_windows_1255	UTF8	WIN1255
utf8_to_windows_1256	UTF8	WIN1256
utf8_to_windows_1257	UTF8	WIN1257
utf8_to_windows_866	UTF8	WIN866
utf8_to_windows_874	UTF8	WIN874
windows_1250_to_iso_8859_2	WIN1250	LATIN2

Имя преобразования [a]	**Исходная кодировка**	**Целевая кодировка**
windows_1250_to_mic	WIN1250	MULE_INTERNAL
windows_1250_to_utf8	WIN1250	UTF8
windows_1251_to_iso_8859_5	WIN1251	ISO_8859_5
windows_1251_to_koi8_r	WIN1251	KOI8R
windows_1251_to_mic	WIN1251	MULE_INTERNAL
windows_1251_to_utf8	WIN1251	UTF8
windows_1251_to_windows_866	WIN1251	WIN866
windows_1252_to_utf8	WIN1252	UTF8
windows_1256_to_utf8	WIN1256	UTF8
windows_866_to_iso_8859_5	WIN866	ISO_8859_5
windows_866_to_koi8_r	WIN866	KOI8R
windows_866_to_mic	WIN866	MULE_INTERNAL
windows_866_to_utf8	WIN866	UTF8
windows_866_to_windows_1251	WIN866	WIN
windows_874_to_utf8	WIN874	UTF8
euc_jis_2004_to_utf8	EUC_JIS_2004	UTF8
utf8_to_euc_jis_2004	UTF8	EUC_JIS_2004
shift_jis_2004_to_utf8	SHIFT_JIS_2004	UTF8
utf8_to_shift_jis_2004	UTF8	SHIFT_JIS_2004
euc_jis_2004_to_shift_jis_2004	EUC_JIS_2004	SHIFT_JIS_2004
shift_jis_2004_to_euc_jis_2004	SHIFT_JIS_2004	EUC_JIS_2004

[a]Имена преобразований следуют стандартной схеме именования. К официальному названию исходной кодировки, в котором все не алфавитно-цифровые символы заменяются подчёркиваниями, добавляется _to_, а за ним аналогично подготовленное имя целевой кодировки.

3.8. Регламентные задачи обслуживания базы данных

В PG360 для достижения оптимальной производительности нужно регулярно выполнять следующие процедуры:

– регулярное создание резервных копий данных, которые необходимы для восстановления системы после сбоя (сбой диска, удаление важной таблицы по ошибке);

– периодическая «очистка» базы данных;

– управление файлами журнала.

Для контроля состояния базы данных и для отслеживания нестандартных ситуаций используется программа check_postgres.

3.8.1. Регламентная очистка

Базы данных PG360 требуют периодического проведения процедуры обслуживания, которая называется *очисткой*. Во многих случаях очистку достаточно выполнять с помощью *демона автоочистки,* также можно дополнить или заменить действие этого демона командами VACUUM (обычно они выполняются по расписанию в заданиях cron или Планировщика задач).

3.8.1.1. Основные принципы очистки

Команды VACUUM в PG360 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 PG360;

– для обновления карты видимости, которая ускоряет сканирование только индекса;

– для предотвращения потери очень старых данных из-за зацикливания идентификаторов *транзакций* или *мультитранзакций*.

Разные причины диктуют выполнение действий VACUUM с разной частотой и в разном объёме.

Существует два варианта VACUUM: обычный VACUUM и VACUUM FULL. Команда VACUUM FULL может высвободить больше дискового пространства, однако работает медленнее. Кроме того, обычная команда VACUUM может выполняться параллельно с использованием производственной базы данных. (При этом такие команды как SELECT, INSERT, UPDATE и DELETE будут выполняться нормально, хотя нельзя будет изменить определение таблицы командами типа ALTER TABLE.) Команда VACUUM FULL требует блокировки обрабатываемой таблицы в режиме ACCESS EXCLUSIVE и поэтому не может выполняться параллельно с другими операциями с этой таблицей. По этой причине администраторы, как правило, должны стараться использовать обычную команду VACUUM и избегать VACUUM FULL.

Команда VACUUM порождает существенный объём трафика ввода/вывода, который может стать причиной низкой производительности в других активных сеансах. Это влияние фоновой очистки можно регулировать, настраивая параметры конфигурации.

3.8.1.2. Демон автоочистки

В PG360 имеется не обязательная, но настоятельно рекомендуемая к использованию функция, называемая автоочисткой, предназначение которой — автоматизировать выполнение команд VACUUM и ANALYZE. Когда автоочистка включена, она проверяет, в каких таблицах было вставлено, изменено или удалено много строк. При этих проверках используются средства сбора статистики; поэтому автоочистка будет работать, только если параметр track_counts имеет значение true. В конфигурации по умолчанию автоочистка включена и соответствующие параметры имеют подходящие значения.

«Демон автоочистки» состоит из нескольких процессов. Существует постоянный фоновый процесс, называемый процессом запуска автоочистки, который отвечает за запуск рабочих процессов автоочистки для всех баз данных. Этот контролирующий процесс распределяет работу по времени, стараясь запускать рабочий процесс для каждой базы данных каждые autovacuum_naptime секунд. (Следовательно, если всего имеется N баз данных, новый рабочий процесс будет запускаться каждые autovacuum_naptime/N секунд.) Одновременно могут выполняться до autovacuum_max_workers рабочих процессов. Если число баз данных, требующих обработки, превышает autovacuum_max_workers, обработка следующей базы начинается сразу по

завершении первого рабочего процесса. Каждый рабочий процесс проверяет все таблицы в своей базе данных и в случае необходимости выполняет VACUUM и/или ANALYZE. Для отслеживания действий рабочих процессов можно установить параметр log_autovacuum_min_duration.

3.8.2. 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.

Команда REINDEX проста и безопасна для использования в любых случаях. Эта команда по умолчанию затребует блокировку ACCESS EXCLUSIVE, поэтому её обычно лучше выполнять с указанием CONCURRENTLY, с которым затребуется только SHARE UPDATE EXCLUSIVE.

3.8.3. Обслуживание журнала

Журнал сервера базы данных необходимо сохранять где-либо, а не просто сбрасывать его в /dev/ null. Этот журнал бесценен при диагностике проблем.

Журнал сервера может быть очень объёмным (особенно при высоких уровнях отладки), 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. Этот подход также можно использовать для получения содержимого журнала в формате CSV (значения, разделённые запятыми).

Можно использовать внешнюю программу для ротации журнальных файлов, например logrotate.

Также существует программа pgBadger — инструмент для сложного анализа файлов журнала. Кроме того, check_postgres может посылать уведомления в Nagios, когда в журнале появляются важные сообщения, а также при обнаружении других нестандартных ситуаций.

3.9. Резервное копирование и восстановление

Базы данных PG360 следует регулярно сохранять в резервной копии. Существует три разных подхода к резервному копированию данных в PG360:

– выгрузка в SQL;

– копирование на уровне файлов;

– непрерывное архивирование.

3.9.1. Выгрузка в SQL

При выгрузке в SQL выполняется 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. PG360 предоставляет для этой цели вспомогательную программу pg_dump. Применение этой программы выглядит следующим образом:

```
pg_dump имя_базы > файл_дампа
```

Программа pg_dump является для PG360 клиентским приложением. Это означает, что можно выполнять процедуру резервного копирования с любого удалённого компьютера, если имеется доступ к нужной базе данных.

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 PG360, pg_dump по умолчанию будет подключаться к базе данных с именем пользователя, совпадающим с именем текущего пользователя ОС. Чтобы переопределить имя, нужно либо добавить параметр -U, либо установить переменную окружения PGUSER. Программа pg_dump подключается к серверу через механизмы проверки подлинности клиента.

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.)

3.9.1.1. 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 необходимо создать ее 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 программой 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.

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 получится только частично восстановленная база 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, необходимо учитывать, 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.

Благодаря способности pg_dump и psql писать и читать каналы ввода/вывода, можно скопировать базу данных непосредственно с одного сервера на другой, например:

```
pg_dump -h host1 имя_базы | psql -h host2 имя_базы
```

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

3.9.1.2. 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. 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.

3.9.1.3. Управление большими базами данных

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

– использование сжатых файлов дампов. Можно 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
```

– использование 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 следующим образом:

```
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.

– использование специального формата дампа pg_dump. Если при сборке PG360 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. 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
```

Для очень больших баз данных может понадобиться сочетать split с одним из двух других методов.

– использование возможности параллельной выгрузки в pg_dump. Чтобы ускорить выгрузку большой БД, можно 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
```

Также можно 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.

3.9.2. Резервное копирование на уровне файлов

Альтернативной стратегией резервного копирования является непосредственное копирование файлов, в которых PG360 хранит содержимое базы данных. Можно использовать любой способ копирования файлов, 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:

– сервер баз данных *должен* быть остановлен переда копированием и восстановлением данных. 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);

– нельзя копировать копированать или восстановливать только отдельных таблицы или баз данных кластера баз данных. Т.к. информацию, содержащуюся в таких скопированных файлах, нельзя использовать без файлов журналов транзакций, pg_xact/*, которые содержат состояние всех транзакций. 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.

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.

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

В процессе всего времени работы PG360 ведёт журнал упреждающей записи (WAL), который расположен в подкаталоге pg_wal/ каталога с данными кластера баз данных. В этот журнал записываются все изменения, вносимые в файлы данных. Журнал используется 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, можно 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 можно 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 необходима непрерывная последовательность заархивированных файлов WAL, начинающаяся не позже, чем с момента начала копирования. Так что для начала нужно настроить и протестировать процедуру архивирования файлов WAL *до того*, как получить первую базовую копию.

3.9.3.1. Настройка архивирования WAL

Чтобы включить архивирование WAL, необходимо установить в файле postgresql.conf в параметре конфигурации wal_level уровень replica (или выше), в archive_mode — значение on, и

задать желаемую команду оболочки в параметре archive_command. В archive_command символы %p заменяются полным путём к файлу, подлежащему архивации, а %f заменяются только именем файла. (Путь задаётся относительно текущего рабочего каталога, то есть каталога данных кластера). Если в команду нужно включить сам символ %, необходимо записать %%. Например:

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

Она будет копировать архивируемые сегменты WAL в каталог /mnt/server/archivedir. После замены параметров %p и %f фактически запускаемая команда может выглядеть так:

```
test ! -f /mnt/server/archivedir/000000010000000A900000065 && cp
pg_wal/000000010000000A900000065
/mnt/server/archivedir/000000010000000A900000065
```

Подобная команда будет генерироваться для каждого следующего архивируемого файла.

Команда архивирования будет запущена от имени того же пользователя, от имени которого работает сервер PG360.

В случае успешного завершения команда архивирования возвращает нулевой код, и сервер PG360 удалит его или переработает. Ненулевой код состояния скажет PG360, что файл не заархивирован; попытки заархивировать его будут периодически повторяться, до успешного заверешения.

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

3.9.3.2. Создание базовой резервной копии

Для получения базовой резервной копию, используйте программу pg_basebackup. Эта программа сохраняет базовую копию в виде обычных файлов или в архиве tar.

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 0000000100001234000055CD, файл истории резервного копирования получит имя 0000000100001234000055CD.007C9330.backup.

Файл истории резервного копирования – это текстовый файл, в который записывается метка, которая была передана pg_basebackup, а также время и текущие сегменты WAL в момент начала и завершения резервной копии. Если с данной меткой связан соответствующий файл дампа, то заархивированного файла истории достаточно, чтобы найти файл дампа, необходимый 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.

3.9.3.3. Восстановление непрерывной архивной копии

Порядок действий при восстановлении базы данных из резервной копии:

1. остановить сервер баз данных, если он запущен;

2. скопировать весь текущий каталог кластера баз данных и все табличные пространства во временный каталог на случай, если они вам понадобятся. При этом необходимо, 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. удалить все существующие файлы и подкаталоги из каталога кластера и из корневых каталогов используемых табличных пространств;

4. восстановить 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 используются табличные пространства, нужно убедиться также, что символьные ссылки в pg_tblspc/ восстановились корректно;

5. удалить все файлы из pg_wal/. Если каталог pg_wal/ не архивировался, необходимо создайть этот каталог с правильными правами доступа;

6. если на шаге 2 были сохранены незаархивированные файлы с сегментами WAL, скопировать их в pg_wal/;

7. установить параметры восстановления в postgresql.conf и создать файл recovery.signal в каталоге данных кластера;

8. запустить 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. просмотреть содержимое базы данных, чтобы убедиться, что она вернулась к желаемому состоянию. Если это не так, перейти на шаг 1.

В процессе процедуры создается конфигурация восстановления, описывающая, как будет выполняться восстановление и до какой точки. Для получения из архива файл-сегменты WAL необходимо выполнить команду restore_command. 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 %, необходимо написать %%. Например:

```
restore_command = 'cp /mnt/server/archivedir/%f %p'
```

Эта команда копирует заархивированные ранее сегменты WAL из каталога /mnt/server/ archivedir.

В случае ошибки команда возвращает ненулевой код возврата. 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.

3.10. Отказоустойчивость, балансировка нагрузки и репликация Серверы

базы данных могут работать совместно для обеспечения возможности быстрого переключения на другой сервер в случае отказа первого (отказоустойчивость) или для обеспечения возможности нескольким серверам БД обрабатывать один набор данных (балансировка нагрузки). В идеале, серверы БД могут работать вместе прозрачно для клиента. Веб-серверы, обрабатывающие статические страницы, можно совместить достаточно легко

посредством простого распределения запросов на несколько машин. Фактически серверы баз данных только для чтения тоже могут быть совмещены достаточно легко. К сожалению, большинство серверов баз данных получают смешанные запросы на чтение/запись, а серверы с доступом на чтение/запись совместить гораздо сложнее. Это объясняется тем, что данные только для чтения достаточно единожды разместить на каждом сервере, а запись на любой из серверов должна распространиться на все остальные серверы, чтобы будущие запросы на чтение возвращали согласованные результаты.

Проблема синхронизации является главным препятствием для совместной работы серверов. Так как единственного решения, устраняющего проблему синхронизации во всех случаях, не существует, предлагается несколько решений. Разные решения подходят к проблеме по-разному и минимизируют её влияние в разных рабочих условиях.

Некоторые решения применяют синхронизацию, позволяя только одному серверу изменять данные. Сервер, который может изменять данные, называется сервером чтения/записи, *ведущим* или *главным* сервером. Сервер, который отслеживает изменения на ведущем, называется *ведомым* или *резервным* сервером. Резервный сервер, к которому нельзя подключаться до тех пор, пока он не будет повышен до главного, называется сервером *тёплого резерва*, а тот, который может принимать соединения и обрабатывать запросы только на чтение, называется сервером *горячего резерва*.

Некоторые решения являются синхронными, при которых транзакция, модифицирующая данные, не считается подтверждённой, пока все серверы не подтвердят транзакцию. Это гарантирует, что при отработке отказа не произойдёт потеря данных и что все балансирующие серверы возвращают целостные данные вне зависимости от того, к какому серверу был запрос. Асинхронное решение, напротив, допускает некоторую задержку между временем подтверждения транзакции и её передачей на другие серверы, допуская возможность, что некоторые транзакции могут быть потеряны в момент переключения на резервный сервер и что балансирующие серверы могут вернуть слегка устаревшие данные. Асинхронная передача используется, когда синхронная будет слишком медленной.

Решения могут так же разделяться по степени детализации. Некоторые решения работают только на уровне всего сервера БД целиком, в то время как другие позволяют работать на уровне таблиц или уровне БД.

В любом случае необходимо принимать во внимание быстродействие. Обычно выбирается компромисс между функциональностью и производительностью. Например, полностью синхронное решение в медленной сети может снизить производительность больше чем наполовину, в то время как асинхронное решение будет оказывать минимальное воздействие.

Далее рассматриваются различные решения по организации отказоустойчивости, репликации и балансировки нагрузки.

3.10.1. Сравнение различных решений

3.10.1.1 Отказоустойчивость на разделяемых дисках

Отказоустойчивость на разделяемых дисках позволяет избежать избыточности синхронизации путём задействования только одной копии базы данных. Она использует единственный дисковый массив, который разделяется между несколькими серверами. Если основной сервер

БД откажет, резервный сервер может подключиться и запустить базу данных, что позволит восстановить БД после аварии. Это обеспечивает быстрое переключение без потери данных.

Функциональность разделяемого оборудования обычно реализована в сетевых устройствах хранения. Так же возможно применение сетевой файловой системы; особое внимание следует уделить тому, чтобы поведение системы полностью соответствовало POSIX. Существенное ограничение этого метода состоит в том, что в случае отказа или порчи разделяемого дискового массива оба сервера: главный и резервный — станут нерабочими. Другая особенность — резервный сервер никогда не получает доступ к разделяемым дискам во время работы главного.

3.10.1.2 Репликация на уровне файловой системы (блочного устройства)

Видоизменённая версия функциональности разделяемого оборудования представлена в виде репликации на уровне файловой системы, когда все изменения в файловой системе отражаются в файловой системе другого компьютера. Единственное ограничение: синхронизация должна выполняться методом, гарантирующим целостность копии файловой системы на резервном сервере — в частности, запись на резервном сервере должна происходить в том же порядке, что и на главном. DRBD является популярным решением на основе репликации файловой системы для Linux.

3.10.1.3 Трансляция журнала предзаписи

Серверы тёплого и горячего резерва могут так же поддерживаться актуальными путём чтения потока записей из журнала изменений (WAL). Если основной сервер отказывает, резервный содержит почти все данные с него и может быть быстро преобразован в новый главный сервер БД. Это можно сделать синхронно или асинхронно, но может быть выполнено только на уровне сервера БД целиком.

Резервный сервер может быть реализован с применением трансляции файлов журналов, или потоковой репликации, или их комбинацией.

3.10.1.4 Логическая репликация

В схеме с логической репликацией сервер баз данных может передавать поток изменений данных на другой сервер. Механизм логической репликации в PG360 формирует поток логических изменений данных, обрабатывая WAL. Логическая репликация позволяет переносить изменения, происходящие только в отдельных таблицах. Для логической репликации не требуется, чтобы за определённым сервером закреплялась роль главного или реплицирующего; напротив, данные могут передаваться в разных направлениях. Используя интерфейс логического декодирования, подобную функциональность могут предоставлять и сторонние расширения.

3.10.1.5 Репликация главный-резервный на основе триггеров

При репликации главный-резервный все запросы, изменяющие данные, пересылаются главному серверу. Главный сервер, в свою очередь, асинхронно пересылает изменённые данные резервному. Резервный сервер может обрабатывать запросы только на чтение при работающем главном. Такой резервный сервер идеален для обработки запросов к хранилищам данных.

Slony-I является примером подобного типа репликации, действующей на уровне таблиц, и поддерживает множество резервных серверов. Так как обновления на резервных серверах происходят асинхронно (в пакетах), возможна потеря данных во время отказа.

3.10.1.6 Репликация SQL в среднем слое

В схеме с репликацией SQL в среднем слое, средний слой перехватывает каждый SQL–запрос и пересылает его на один или все серверы. Каждый сервер работает независимо. Модифицирующие запросы должны быть направлены всем серверам, чтобы каждый из них получал все изменения. Но читающие запросы могут посылаться только одному серверу, что позволяет перераспределить читающую нагрузку между всеми серверами.

Если запросы просто перенаправлять без изменений, функции подобные random(), CURRENT_TIMESTAMP и последовательности могут получить различные значения на разных

серверах. Это происходит потому что каждый сервер работает независимо, а эти запросы неизбирательные (и действительно не изменяют строки). Если такая ситуация недопустима, или средний слой, или приложение должны запросить подобные значения с одного сервера, затем использовать его в других пишущих запросах. Другим способом является применения этого вида репликации совместно с другим традиционным набором репликации главный– резервный, то есть изменяющие данные запросы посылаются только на главный сервер, а затем применяются на резервном в процессе этой репликации, но не с помощью реплицирующего среднего слоя. Следует иметь в виду, что все транзакции фиксируются или прерываются на всех серверах, возможно с применением двухфазной фиксации (см. PREPARE TRANSACTION и COMMIT PREPARED). Репликацию такого типа реализуют, например Pgpool-II и Continuent Tungsten.

3.10.1.7 Асинхронная репликация с несколькими ведущими

Если серверы не находятся постоянно в единой сети или связаны низкоскоростным каналом, как например, ноутбуки или удалённые серверы, обеспечение согласованности данных между ними представляет проблему. Когда используется асинхронная репликация с несколькими ведущими серверами, каждый из них работает независимо и периодически связывается с другими серверами для определения конфликтующих транзакций. Конфликты могут урегулироваться пользователем или по правилам их разрешения.

Итоговая таблица 5 возможностей различных решений приведена ниже.

Таблица 5. Таблица свойств отказоустойчивости, балансировки нагрузки и репликации

Тип	**Разделяемый диск**	**Репл. файловой системы**	**Трансляц журнала упреждающей записи**	**Логическая репл.**	**Триггерная репл.**	**Репликация SQL в среднем сло**	**Асинхр. репл. с н. в.**
Не требуется специального оборудовани		•	•	•	•	•	•
Допускается несколько ведущих серверов				•		•	•
Нет доп. нагрузки ведущего сервера	•		•	•		•	
Нет задержки при нескольких серверах	•		без синхр.	без синхр.	•		•
Отказ ведущего сервера не может привести к потере данных	•	•	с синхр.	с синхр.		•	
Сервер реплики принимает читающие запросы			с горячим резервом	•	•	•	•
Репликация на уровне таблиц				•	•		•
Не требуется разрешение конфликтов	•	•	•		•	•	

3.10.1.8 Секционирование данных

При секционировании таблицы расщепляются на наборы данных. Каждый из наборов может быть изменён только на одном сервере.

3.10.1.9 Выполнение параллельных запросов на нескольких серверах

Многие из описанных выше решений позволяют обрабатывать несколько запросов на нескольких серверах, но ни одно из них не поддерживает выполнение одного запроса на нескольких серверах, что позволило бы его ускорить. Данное же решение позволяет нескольким

серверам обрабатывать один запрос одновременно. Для этого обычно данные разделяются между серверами, серверы выполняют свои части запросов, выдают результаты центральному серверу, а он, в свою очередь, объединяет полученные данные и выдаёт итоговый результат пользователю. Это решение может быть реализовано с применением набора средств PL/Proxy.

3.10.2. Трансляция журналов на резервные серверы

Постоянная архивация может использоваться для создания кластерной конфигурации *высокой степени доступности* (HA) с одним или несколькими *резервными серверами*, способными заменить ведущий сервер в случае выхода его из строя. Такую реализацию отказоустойчивости часто называют *тёплый резерв* или *трансляция журналов*.

Ведущий и резервный серверы работают совместно для обеспечения этой возможности, при этом они связаны опосредованно. Ведущий сервер работает в режиме постоянной архивации изменений, в то время как каждый резервный сервер работает в режиме постоянного приёма архивных изменений, получая файлы WAL от ведущего. Для обеспечения этой возможности не требуется вносить изменения в таблицы БД, поэтому администрировать данное решение репликации проще, чем ряд других. Так же такая конфигурация относительно слабо влияет на производительность ведущего сервера.

Непосредственную передачу записей WAL с одного сервера БД на другой обычно называют трансляцией журналов (или доставкой журналов). PG360 реализует трансляцию журналов на уровне файлов, передавая записи WAL по одному файлу (сегменту WAL) единовременно. Файлы WAL (размером 16 МБ) можно легко и эффективно передать на любое расстояние, будь то соседний сервер, другая система в местной сети или сервер на другом краю света. Требуемая пропускная способность при таком подходе определяется скоростью записи транзакций на ведущем сервере. Трансляция журналов на уровне записей более фрагментарная операция, при которой изменения WAL передаются последовательно через сетевое соединение.

Трансляция журналов асинхронна, то есть записи WAL доставляются после завершения транзакции. В результате образуется окно, когда возможна потеря данных при отказе сервера: будут утеряны ещё не переданные транзакции. Размер этого окна при трансляции файлов журналов может быть ограничен параметром archive_timeout, который может принимать значение меньше нескольких секунд. Тем не менее подобные заниженные значения могут потребовать существенного увеличения пропускной способности, необходимой для трансляции файлов. При потоковой репликации окно возможности потери данных гораздо меньше.

Скорость восстановления достаточно высока, обычно резервный сервер становится полностью доступным через мгновение после активации. В результате такое решение называется тёплым резервом, что обеспечивает отличную отказоустойчивость. Восстановление сервера из архивной копии базы и применение изменений обычно происходит существенно дольше. Поэтому такие действия обычно требуются при восстановлении после аварии, не для отказоустойчивости. Так же резервный сервер может обрабатывать читающие запросы. В этом случае он называется сервером горячего резерва.

3.10.2.1. Планирование

Необходимо подбирать ведущий и резервный серверы так, чтобы они были максимально похожи, с точки зрения базы данных (одинаковые версии СУБД, пути монтировани для табличных пространств и др.) разрядности ОС и др.

Учитывайте, что если CREATE TABLESPACE выполнена на ведущем сервере, новая точка монтирования для этой команды уже должна существовать на резервных серверах до её выполнения. Аппаратная часть не должна быть в точности одинаковой, но опыт показывает, что сопровождать идентичные системы легче, чем две различные на протяжении жизненного цикла приложения и системы. В любом случае архитектура оборудования должна быть одинаковой – например, трансляция журналов с 32-битной на 64- битную систему не будет работать.

В общем случае трансляция журналов между серверами с различными основными версиями PG360 невозможна. Особенность в том, чтобы не вносить изменения в дисковые форматы при обновлениях корректирующей версии, таким образом, ведущий и резервный серверы, имеющие разные корректирующие версии, могут работать успешно. Тем не менее формально такая возможность не поддерживается, поэтому рекомендуется поддерживать одинаковую версию ведущего и резервных серверов, насколько это возможно. При обновлении корректирующей версии безопаснее будет в первую очередь обновить резервные серверы – новая корректирующая версия с большей вероятностью прочитает файл WAL предыдущей корректирующей версии, чем наоборот.

3.10.2.2. Работа резервного сервера

Сервер переходит в режим ожидания, если в каталоге данных при запуске сервера есть файл standby.signal.

Сервер, работающий в режиме резервного, последовательно применяет файлы WAL, полученные от главного. Резервный сервер может читать файлы WAL из архива WAL (см. restore_command) или напрямую с главного сервера по соединению TCP (потоковая репликация). Резервный сервер также будет пытаться восстановить любой файл WAL, найденный в кластере резервного в каталоге pg_wal. Это обычно происходит после перезапуска сервера, когда он применяет заново файлы WAL, полученные от главного сервера перед перезапуском. Но можно и вручную скопировать файлы в каталог pg_wal, чтобы применить их в любой момент времени.

В момент запуска резервный сервер начинает восстанавливать все доступные файлы WAL, размещённые в архивном каталоге, указанном в команде restore_command. По достижении конца доступных файлов WAL или при сбое команды restore_command сервер пытается восстановить все файлы WAL, доступные в каталоге pg_wal. Если это не удаётся и потоковая репликация настроена, резервный сервер пытается присоединиться к ведущему и начать закачивать поток WAL с последней подтверждённой записи, найденной в архиве или pg_wal. Если это действие закончилось неудачей, или потоковая репликация не настроена, или соединение позднее разорвалось, резервный сервер возвращается к шагу 1 и пытается восстановить файлы из архива вновь. Цикл обращения за файлами WAL к архиву, pg_wal, и через потоковую репликацию продолжается до остановки сервера или переключения его роли, вызванного файлом-триггером.

Режим резерва завершается и сервер переключается в обычный рабочий режим при получении команды pg_ctl promote, в результате вызова pg_promote() или при обнаружении файла-

триггера (promote_trigger_file). Перед переключением сервер восстановит все файлы WAL, непосредственно доступные из архива или pg_wal, но пытаться подключиться к главному серверу он больше не будет.

3.10.2.3. Подготовка главного сервера для работы с резервными

Настройка постоянного архивирования на ведущем сервере в архивный каталог, доступный с резервного, описана в п. 3.9.3. Расположение архива должно быть доступно с резервного сервера даже при отключении главного, то есть его следует разместить на резервном или другом доверенном, но не на главном сервере.

При использовании потоковой репликации следует настроить режим аутентификации на ведущем сервере, чтобы разрешить соединения с резервных. Для этого создать роль и обеспечить подходящую запись в файле pg_hba.conf в разделе доступа к БД replication. Так же следует убедиться, что для параметра max_wal_senders задаётся достаточно большое значение в конфигурационном файле ведущего сервера. При использовании слотов для репликации также достаточно большое значение нужно задать для max_replication_slots.

Создание базовой резервной копии, необходимой для запуска резервного сервера, описано в п. 3.9.3.2.

3.10.2.4. Настройка резервного сервера

Для запуска резервного сервера необходимо восстановить резервную копию, снятую с ведущего. Затем нужно создать файл standby.signal в каталоге данных кластера резервного сервера. Задать в restore_command команду копирования файлов из архива WAL. Если планируется несколько резервных серверов в целях отказоустойчивости, значением параметра recovery_target_timeline должно быть latest (значение по умолчанию), чтобы резервный сервер переходил на новую линию времени, образуемую при обработке отказа и переключении на другой сервер.

При необходимости потоковой репликации необходимо задать в primary_conninfo параметры строки соединения для libpq, включая имя (или IP-адрес) сервера и всё, что требуется для подключения к ведущему серверу. Если ведущий требует пароль для аутентификации, пароль также должен быть указан в primary_conninfo.

Если резервный сервер настраивается в целях отказоустойчивости, на нём следует настроить архивацию WAL, соединения и аутентификацию, как на ведущем сервере, потому что резервный сервер станет ведущим после отработки отказа.

При использовании архива WAL его размер может быть уменьшен с помощью команды, задаваемой в archive_cleanup_command, если она будет удалять файлы, ставшие ненужными для резервного сервера. Утилита pg_archivecleanup разработана специально для использования в archive_cleanup_command при типичной конфигурации с одним резервным сервером (см. pg_archivecleanup). Если архив используется в целях резервирования, необходимо сохранять все файлы, требующиеся для восстановления как минимум с последней базовой резервной копии, даже если они не нужны для резервного сервера.

Пример конфигурации:

```
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo
password=foopass options=''-c wal_sender_timeout=5000'''
```

```
restore_command = 'cp /path/to/archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /path/to/archive %r'
```

Можно поддерживать любое количество резервных серверов, но при применении потоковой репликации необходимо убедиться, что значение max_wal_senders на ведущем достаточно большое, чтобы все они могли подключиться одновременно.

3.10.2.5. Потоковая репликация

При потоковой репликации резервный сервер может работать с меньшей задержкой, чем при трансляции файлов. Резервный сервер подключается к ведущему, который передаёт поток записей WAL резервному в момент их добавления, не дожидаясь окончания заполнения файла WAL.

Потоковая репликация асинхронна по умолчанию, то есть имеется небольшая задержка между подтверждением транзакции на ведущем сервере и появлением этих изменений на резервном. Тем не менее эта задержка гораздо меньше, чем при трансляции файлов журналов, обычно в пределах одной секунды, если резервный сервер достаточно мощный и справляется с нагрузкой. При потоковой репликации настраивать archive_timeout для уменьшения окна потенциальной потери данных не требуется.

При потоковой репликации без постоянной архивации на уровне файлов, сервер может избавиться от старых сегментов WAL до того, как резервный получит их. В этом случае резервный сервер потребует повторной инициализации из новой базовой резервной копии. Этого можно избежать, установив для wal_keep_size достаточно большое значение, при котором сегменты WAL будут защищены от ранней очистки, либо настроив слот репликации для резервного сервера. Если с резервного сервера доступен архив WAL, этого не требуется, так как резервный может всегда обратиться к архиву для восполнения пропущенных сегментов.

Чтобы настроить потоковую репликацию, сначала необходимо настроить резервный сервер в режиме передачи журналов в виде файлов. Затем переключить его в режим потоковой репликации, установив в primary_conninfo строку подключения, указывающую на ведущий. Настроить listen_addresses и параметры аутентификации (см. pg_hba.conf) на ведущем сервере таким образом, чтобы резервный смог подключиться к псевдобазе replication ведущего.

В системах, поддерживающих параметр сокета keepalive, подходящие значения tcp_keepalives_idle, tcp_keepalives_interval и tcp_keepalives_count помогут ведущему вовремя заметить разрыв соединения.

Установить максимальное количество одновременных соединений с резервных серверов (с помощью параметра max_wal_senders).

При запуске резервного сервера с правильно установленным primary_conninfo резервный подключится к ведущему после воспроизведения всех файлов WAL, доступных из архива. При успешном установлении соединения можно увидеть walreceiver на резервном сервере и соответствующий процесс walsender на ведущем.

3.10.2.5.1. Аутентификация

Право использования репликации очень важно ограничить так, чтобы только доверенные пользователи могли читать поток WAL, так как из него можно извлечь конфиденциальную

информацию. Резервный сервер должен аутентифицироваться на главном от имени пользователя с правом REPLICATION или от имени суперпользователя. Настоятельно рекомендуется создавать выделенного пользователя с правами REPLICATION и LOGIN специально для репликации. Хотя право REPLICATION даёт очень широкие полномочия, оно не позволяет модифицировать данные в ведущей системе, тогда как с правом SUPERUSER это можно делать.

Список аутентификации клиентов для репликации содержится в файле pg_hba.conf в записях с установленным значением replication в поле *database*.

Например, если резервный сервер запущен на компьютере с IP-адресом 192.168.1.100 и учётная запись для репликации foo, администратор может добавить следующую строку в файл pg_hba.conf ведущего:

```
# Разрешить пользователю "foo" с компьютера 192.168.1.100 подключаться к этому # серверу в качестве партнёра репликации, если был передан правильный пароль. #
# TYPE DATABASEUSER ADDRESS METHOD
host replication foo 192.168.1.100/32 md5
```

Имя компьютера и номер порта для ведущего, имя подключающегося пользователя и пароль указываются в primary_conninfo. Пароль также может быть задан в файле ~/.pgpass на резервном сервере (в поле *database* нужно указать replication). Например, если ведущий принимает подключения по IP-адресу 192.168.1.50, через порт 5432, пользователя для репликации foo с паролем foopass, администратор может добавить следующую строку в файл postgresql.conf на резервном сервере:

```
# Резервный сервер подключается к ведущему, работающему на компьютере 192.168.1.50 # (порт 5432), от имени пользователя "foo" с паролем "foopass".
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=foopass'
```

3.10.2.5.2. Наблюдение

Важным индикатором стабильности работы потоковой репликации является количество записей WAL, созданных на ведущем, но ещё не применённых на резервном сервере. Задержку можно подсчитать, сравнив текущую позиции записи WAL на ведущем с последней позицией WAL, полученной на резервном сервере. Эти позиции можно узнать, воспользовавшись функциями pg_current_wal_lsn на ведущем и pg_last_wal_receive_lsn на резервном, соответственно. Последняя полученная позиция WAL на резервном сервере также выводится в состоянии процесса-приёмника WAL, которое показывает команда ps.

Список процессов-передатчиков WAL можно получить через представление pg_stat_replication. Большая разница между pg_current_wal_lsn и полем sent_lsn этого представления может указывать на то, что главный сервер работает с большой нагрузкой, тогда как разница между sent_lsn и pg_last_wal_receive_lsn на резервном может быть признаком задержек в сети или большой нагрузки резервного сервера.

На сервере горячего резерва состояние процесса-приёмника WAL можно получить через представление pg_stat_wal_receiver. Большая разница между pg_last_wal_replay_lsn и полем flushed_lsn свидетельствует о том, что WAL поступает быстрее, чем удаётся его воспроизвести.

3.10.2.6. Слоты репликации

Слоты репликации автоматически обеспечивают механизм сохранения сегментов WAL, пока они не будут получены всеми резервными и главный сервер не будет удалять строки, находящиеся в статусе recovery conflict даже при отключении резервного.

Вместо использования слотов репликации для предотвращения удаления старых сегментов WAL можно применять wal_keep_size или сохранять сегменты в архиве с помощью команды archive_command. Тем не менее эти методы часто приводят к тому, что хранится больше сегментов WAL, чем необходимо, в то время как для слотов репликации сохраняются только те сегменты, которые нужны. С другой стороны, для слотов репликации может потребоваться так много сегментов WAL, что они заполнят всё пространство, отведённое для pg_wal; объём файлов WAL, сохраняемых для слотов репликации, ограничивается параметром max_slot_wal_keep_size.

Подобным образом, параметры hot_standby_feedback и vacuum_defer_cleanup_age позволяют защитить востребованные строки от удаления при очистке, но первый параметр не защищает в тот промежуток времени, когда резервный сервер не подключён, а для последнего часто нужно задавать большое значение, чтобы обеспечить должную защиту. Слоты репликации решают эти проблемы.

Каждый слот репликации обладает именем, состоящим из строчных букв, цифр и символов подчёркивания. Имеющиеся слоты репликации и их статус можно просмотреть в представлении pg_replication_slots. Слоты могут быть созданы и удалены как с помощью протокола потоковой репликации, так и посредством функций SQL.

3.10.2.7. Каскадная репликация

Свойство каскадной репликации позволяет резервному серверу принимать соединения репликации и потоки WAL от других резервных, выступающих посредниками. Это может быть полезно для уменьшения числа непосредственных подключений к главному серверу, а также для уменьшения накладных расходов при передаче данных в интрасети.

Резервный сервер, выступающий как получатель и отправитель, называется каскадным резервным сервером. Резервные серверы, стоящие ближе к главному, называются серверами верхнего уровня, а более отдалённые — серверами нижнего уровня. Каскадная репликация не накладывает ограничений на количество или организацию последующих уровней, а каждый резервный соединяется только с одним сервером вышестоящего уровня, который в конце концов соединяется с единственным главным/ведущим сервером.

Резервный сервер каскадной репликации не только получает записи WAL от главного, но так же восстанавливает их из архива. Таким образом, даже если соединение с сервером более высокого уровня разорвётся, потоковая репликация для последующих уровней будет продолжаться до исчерпания доступных записей WAL.

Каскадная репликация в текущей реализации асинхронна. Параметры синхронной репликации в настоящее время не оказывают влияние на каскадную репликацию.

Распространение обратной связи горячего резерва работает от нижестоящего уровня к вышестоящему уровню вне зависимости от способа организации связи.

Если вышестоящий резервный сервер будет преобразован в новый главный, нижестоящие серверы продолжат получать поток с нового главного при условии, что recovery_target_timeline имеет значение latest (по умолчанию).

Для использования каскадной репликации необходимо настроить резервный каскадный сервер на приём соединений репликации (то есть установить max_wal_senders и hot_standby, настроить host-based authentication). Так же может быть необходимо настроить на нижестоящем резервном значение primary_conninfo на каскадный резервный сервер.

3.10.2.8. Синхронная репликация

3.10.2.8.1. По умолчанию в PG360 потоковая репликация асинхронна. Если ведущий сервер выходит из строя, некоторые транзакции, которые были подтверждены, но не переданы на резервный, могут быть потеряны. Объём потерянных данных пропорционален задержке репликации на момент отработки отказа.

Синхронная репликация предоставляет возможность гарантировать, что все изменения, внесённые в транзакции, были переданы одному или нескольким синхронным резервным серверам. Это повышает стандартный уровень надёжности, гарантируемый при фиксации транзакции. Этот уровень защиты соответствует второму уровню безопасности репликации из теории вычислительной техники, или групповой безопасности первого уровня (безопасности групповой и уровня 1), когда выбран режим synchronous_commit remote_write.

При синхронной репликации каждая фиксация пишущей транзакции ожидает подтверждения того, что запись фиксации помещена в журнал предзаписи на диске на обоих серверах: ведущем и резервном. При таком варианте потеря данных может произойти только в случае одновременного выхода из строя ведущего и резервного серверов. Это обеспечивает более высокий уровень надёжности, при условии продуманного подхода системного администратора к вопросам размещения и управления этими серверами. Ожидание подтверждения увеличивает уверенность в том, что данные не будут потеряны во время сбоя сервера, но при этом увеличивает время отклика для обработки транзакции. Минимальное время ожидания равно времени передачи данных от ведущего к резервному и обратно.

Транзакции только для чтения и откат транзакции не требуют ожидания для ответа с резервного сервера. Промежуточные подтверждения не ожидают ответа от резервного сервера, только подтверждение верхнего уровня. Долгие операции вида загрузки данных или построения индекса не ожидают финального подтверждения. Но все двухфазные подтверждения требуют ожидания, включая подготовку и непосредственно подтверждение.

Синхронным резервным сервером может быть резервный сервер при физической репликации или подписчик при логической репликации. Это также может быть другой потребитель потока логической или физической репликации, способный отправлять в ответ требуемые сообщения. Помимо встроенных систем логической и физической репликации, к таким потребителям относятся специальные программы, pg_receivewal и pg_recvlogical, а также некоторые сторонние системы репликации и внешние программы. Подробнее об организации синхронной репликации с их использованием можно узнать в соответствующей документации.

3.10.2.8.2. Базовая настройка

При настроенной потоковой репликации установка синхронной репликации требует только дополнительной настройки: необходимо выставить synchronous_standby_names в непустое значение. Так же необходимо установить synchronous_commit в значение on, но так как это значение по умолчанию, обычно действий не требуется. В такой конфигурации каждая транзакция будет ожидать подтверждение того, что на резервном сервере произошла запись транзакции в надёжное хранилище. Значение synchronous_commit может быть выставлено для отдельного пользователя, может быть прописано в файле конфигурации, для конкретного пользователя или БД или динамически изменено приложением для управления степенью надёжности на уровне отдельных транзакций.

После сохранения записи о фиксации транзакции на диске ведущего сервера эта запись WAL передаётся резервному серверу. Резервный сервер отвечает подтверждающим сообщением после сохранения каждого нового блока данных WAL на диске, если только wal_receiver_status_interval на нём не равен нулю. В случае, когда выбран режим synchronous_commit remote_apply, резервный сервер передаёт подтверждение после воспроизведения записи фиксации, когда транзакция становится видимой. Если резервный сервер выбран на роль синхронного резервного в соответствии со значением synchronous_standby_names на ведущем, подтверждающие сообщения с этого сервера, в совокупности с сообщениями с других синхронных серверов, будут сигналом к завершению ожидания при фиксировании транзакций, требующих подтверждения сохранения записи фиксации. Эти параметры позволяют администратору определить, какие резервные серверы будут синхронными резервными. Настройка синхронной репликации в основном осуществляется на главном сервере. Перечисленные в списке резервных серверы должны быть подключены к нему непосредственно; он ничего не знает о резервных серверах, подключённых каскадно, через промежуточные серверы.

Если synchronous_commit имеет значение remote_write, то в случае подтверждения транзакции ответ от резервного сервера об успешном подтверждении будет передан, когда данные запишутся в ОС, но не когда данные будет реально сохранены на диске. При таком значении уровень надёжности снижается по сравнению со значением on. Резервный сервер может потерять данные в случае падения ОС, но не в случае падения PG360. Тем не менее этот вариант полезен на практике, так как позволяет сократить время отклика для транзакции. Потеря данных может произойти только в случае одновременного сбоя ведущего и резервного, осложнённого повреждением БД на ведущем.

Если synchronous_commit имеет значение remote_apply, то для завершения фиксирования транзакции потребуется дождаться, чтобы текущие синхронные резервные серверы сообщили, что они воспроизвели транзакцию и её могут видеть запросы пользователей. В простых случаях это позволяет обеспечить обычный уровень согласованности и распределение нагрузки.

Пользователи прекратят ожидание в случае запроса на быструю остановку сервера. В то время как при использовании асинхронной репликации сервер не будет полностью остановлен, пока все исходящие записи WAL не переместятся на текущий присоединённый резервный сервер.

3.10.2.8.3. Несколько синхронных резервных серверов

Синхронная репликация поддерживает применение одного или нескольких синхронных резервных серверов; транзакции будут ждать, пока все резервные серверы, считающиеся синхронными, не подтвердят получение своих данных. Число синхронных резервных серверов, от которых транзакции должны ждать подтверждения, задаётся в параметре synchronous_standby_names. В этом параметре также задаётся список имён резервных серверов и метод (FIRST или ANY) выбора синхронных из заданного списка.

С методом FIRST производится синхронная репликация на основе приоритетов, когда транзакции фиксируются только после того, как их записи в WAL реплицируются на заданное число синхронных резервных серверов, выбираемых согласно приоритетам. Серверы, имена которых идут в начале списка, имеют больший приоритет и выбираются на роль синхронных. Другие резервные серверы, идущие в этом списке за ними, считаются потенциальными синхронными. Если один из текущих синхронных резервных серверов по какой-либо причине отключается, он будет немедленно заменён следующим по порядку резервным сервером.

С методом ANY производится синхронная репликация на основе кворума, когда транзакции фиксируются только после того, как их записи в WAL реплицируются на *как минимум* заданное число синхронных серверов в списке.

Состояние синхронности резервных серверов можно увидеть в представлении pg_stat_replication.

3.10.2.8.4. Планирование производительности

Организуя синхронную репликацию, обычно нужно обстоятельно обдумать конфигурацию и размещение резервных серверов, чтобы обеспечить приемлемую производительность приложений. Ожидание не потребляет системные ресурсы, но блокировки транзакций будут сохраняться до подтверждения передачи. Как следствие, непродуманное использование синхронной репликации приведёт к снижению производительности БД из-за увеличения времени отклика и числа конфликтов.

PG360 позволяет разработчикам выбрать требуемый уровень надёжности, обеспечиваемый при репликации. Он может быть установлен для системы в целом, для отдельного пользователя или соединения или даже для отдельной транзакции.

Например, в рабочей нагрузке приложения 10% изменений могут относиться к важным данным клиентов, а 90% — к менее критичным данным, потеряв которые, бизнес вполне сможет выжить (например, это могут быть текущие разговоры пользователей между собой).

При настройке уровня синхронности репликации на уровне приложения (на ведущем) можно задать синхронную репликацию для большинства важных изменений без замедления общего рабочего ритма. Возможность настройки на уровне приложения является важным и практичным средством для получения выгод синхронной репликации при высоком быстродействии.

Следует иметь в виду, что пропускная способность сети должна быть больше скорости генерирования данных WAL.

3.10.2.8.5. Планирование отказоустойчивости

В synchronous_standby_names задаётся количество и имена синхронных резервных серверов, от которых будет ожидаться подтверждение при фиксировании транзакции, когда параметру synchronous_commit присвоено значение on, remote_apply или remote_write. Фиксирование транзакции в таком режиме может не завершиться никогда, если один из синхронных резервных серверов выйдет из строя.

Поэтому для высокой степени доступности лучше всего обеспечить наличие синхронных резервных серверов в должном количестве. Для этого можно перечислить несколько потенциальных резервных серверов в строке synchronous_standby_names.

При синхронной репликации на основе приоритетов синхронными резервными серверами станут серверы, имена которых стоят в этом списке первыми. Следующие за ними серверы будут становиться синхронными резервными при отказе одного из текущих.

При синхронной репликации на основе кворума кандидатами на роль синхронных резервных будут все серверы в списке. И если один из них откажет, другие серверы будут продолжать исполнять эту роль.

Когда к ведущему серверу впервые присоединяется резервный, он ещё не будет полностью синхронизированным. Это называется состоянием навёрстывания. Как только отставание резервного от ведущего сервера сократится до нуля в первый раз, система перейдет в состояние потоковой передачи в реальном времени. Сразу после создания резервного сервера навёрстывание может быть длительным. В случае выключения резервного сервера длительность этого процесса увеличится соответственно продолжительности простоя. Резервный сервер может стать синхронным только по достижении состояния потоковой передачи. Это состояние можно проследить в представлении pg_stat_replication.

Если ведущий сервер перезапускается при наличии зафиксированных транзакций, ожидающих подтверждения, эти транзакции будут помечены как полностью зафиксированные после восстановления ведущего. При этом нельзя гарантировать, что все резервные серверы успели получить все текущие данные WAL к моменту падения ведущего. Таким образом, некоторые транзакции могут считаться незафиксированными на резервном сервере, даже если они считаются зафиксированными на ведущем. Гарантия, которую мы можем дать, состоит в том, что приложение не получит явного подтверждения успешной фиксации, пока не будет уверенности, что данные WAL получены всеми синхронными резервными серверами.

Если запустить синхронные резервные серверы в указанном количестве не удаётся, вам следует уменьшить число синхронных серверов, подтверждения которых требуются для завершения фиксации транзакций, в параметре synchronous_standby_names (или вовсе отключить его) и перезагрузить файл конфигурации на ведущем сервере.

В случае если ведущий сервер стал недоступным для оставшихся резервных, следует переключиться на наиболее подходящий из имеющихся резервных серверов.

Если необходимо пересоздать резервный сервер при наличии ожидающей подтверждения транзакции необходимо убедиться, что команды pg_start_backup() и pg_stop_backup() запускаются в сессии с установленным synchronous_commit = off, в противном случае эти запросы на подтверждение будут бесконечными для вновь возникшего резервного сервера.

3.10.2.9. Непрерывное архивирование на резервном сервере

Когда на резервном сервере применяется непрерывное архивирование WAL, возможны два различных сценария:

– архив WAL может быть общим для ведущего и резервного сервера;

– резервный сервер может иметь собственный архив WAL.

Когда резервный сервер работает с собственным архивом WAL, необходимо установить в archive_mode значение always, будет вызываться команду архивации для каждого сегмента WAL, который он получает при восстановлении из архива или потоковой репликации.

В случае с общим архивом можно поступить аналогично, и команда archive_command проверяет, нет ли в архиве файла, идентичного архивируемому. Таким образом, команда archive_command заботиться о том, чтобы существующий файл не был заменён файлом с другим содержимым, а в случае попытки повторного архивирования сообщает об успешном выполнении.

Если в archive_mode установлено значение on, архивация в режиме восстановления или резерва не производится. В случае повышения резервного сервера, он начнёт архивацию после повышения, но в архив не попадут те файлы WAL или файлы истории линии времени, которые генерировал не он сам. Поэтому, чтобы в архиве оказался полный набор файлов WAL, необходимо обеспечить архивацию всех файлов WAL до того, как они попадут на резервный сервер. Это естественным образом происходит при трансляции файлов журналов, так как резервный сервер может восстановить только файлы, которые находятся в архиве, однако при потоковой репликации это не так. Когда сервер работает не в режиме резерва, различий между режимами on и always нет.

3.10.3. Отработка отказа

Если ведущий сервер отказывает, резервный должен начать процедуры отработки отказа.

Если отказывает резервный сервер, никакие действия по отработке отказа не требуются. Если резервный сервер будет перезапущен, даже через некоторое время, немедленно начнётся операция восстановления, благодаря возможности возобновляемого восстановления. Если вернуть резервный сервер в строй невозможно, необходимо создать полностью новый экземпляр резервного сервера.

Когда ведущий сервер отказывает и резервный сервер становится новым ведущим, а затем старый ведущий включается снова, необходим механизм для предотвращения возврата старого к роли ведущего.

Во многих отказоустойчивых конструкциях используются всего две системы: ведущая и резервная, с некоторым контрольным механизмом, который постоянно проверяет соединение между ними и работоспособность ведущей. Также возможно применение третьей системы (называемой следящим сервером) для исключения некоторых вариантов нежелательной отработки отказа, но эта дополнительная сложность оправдана, только если вся схема достаточно хорошо продумана и тщательно протестирована.

PG360 не предоставляет системного программного обеспечения, необходимого для определения сбоя на ведущем и уведомления резервного сервера баз данных. Имеется множество подобных инструментов, которые хорошо интегрируются со средствами ОС, требуемыми для успешной отработки отказа, например, для миграции IP-адреса.

Когда происходит переключение на резервный сервер, только один сервер продолжает работу. Это состояние называется ущербным. Бывший резервный сервер теперь является ведущим, а бывший ведущий отключён и может оставаться отключённым. Для возвращения к нормальному состоянию необходимо запустить новый резервный сервер, либо на бывшем ведущем, либо в третьей, возможно, новой системе. Ускорить этот процесс в больших кластерах позволяет утилита pg_rewind. По завершении этого процесса можно считать, что ведущий и резервный сервер поменялись ролями. Некоторые используют третий сервер в качестве запасного для нового ведущего, пока не будет воссоздан новый резервный сервер, хотя это, очевидно, усложняет конфигурацию системы и рабочие процедуры.

Таким образом, переключение с ведущего сервера на резервный может быть быстрым, но требует некоторого времени для повторной подготовки отказоустойчивого кластера. Регулярные переключения с ведущего сервера на резервный полезны, так как при этом появляется плановое время для отключения и проведения обслуживания. Это также позволяет убедиться в работоспособности механизма отработки отказа и гарантировать, что он действительно будет работать, когда потребуется. Эти административные процедуры рекомендуется документировать письменно.

Чтобы сделать ведущим резервный сервер, принимающий журналы, необходимо выполнить команду pg_ctl promote, вызвать функцию pg_promote() или создать файл-триггер с именем и путём, заданным в параметре promote_trigger_file. Если для переключения планируется использовать команду pg_ctl promote или функцию pg_promote(), файл promote_trigger_file не требуется. Если резервный сервер применяется для анализа данных, чтобы только разгрузить ведущий, выполняя запросы на чтение, а не обеспечивать отказоустойчивость, повышать его до ведущего не понадобится.

3.10.4. Другие методы трансляции журнала

В качестве альтернативы встроенному режиму резерва можно использовать команду restore_command, следящую за содержимым архива. Команда restore_command исполняется единожды для каждого файла WAL, но процесс, запускаемый restore_command, создаётся и завершается для каждого файла, так что это не служба и не серверный процесс, и применить сигналы и реализовать их обработчик в нём нельзя. Поэтому restore_command не подходит для отработки отказа. Можно организовать переключение по тайм-ауту, в частности, связав его с известным значением archive_timeout на ведущем. Однако это не очень надёжно, так как переключение может произойти и из-за проблем в сети или загруженности ведущего сервера. Для этого следует использовать механизм уведомлений, например, явно создавать файл-триггер, если это возможно.

3.10.4.1. Реализация

Для процедуры настройки для резервного сервера с применением альтернативного метода необходимо:

– развернуть ведущую и резервную системы, сделав их максимально одинаковыми, включая две одинаковые копии PG360;

– настроить постоянную архивацию с ведущего сервера в каталог архивов WAL на резервном. Убедиться, что archive_mode, archive_command и archive_timeout установлены в соответствующие значения на ведущем;

– создать базовую копию данных ведущего сервера и восстановить её на резервном;

– запустить восстановление на резервном сервере из локального архива WAL с помощью команды restore_command.

Поток восстановления только читает архив WAL, поэтому, как только файл WAL скопирован на резервную систему, его можно копировать на ленту в то время, как его читает резервный сервер. Таким образом, работа резервного сервера в целях отказоустойчивости может быть совмещена с долговременным сохранением файлов для восстановления после катастрофических сбоев.

Для целей тестирования возможен запуск ведущего и резервного сервера в одной системе. Это не обеспечивает надёжность серверов, так же как и не подходит под описание высокой доступности.

3.10.4.2. Построчная трансляция журнала

Так же возможна реализация построчной трансляции журналов с применением альтернативного метода, хотя это требует дополнительных доработок, а изменения будут видны для запросов на сервере горячего резерва только после передачи полного файла WAL.

Внешняя программа может вызвать функцию pg_walfile_name_offset() для поиска имени файла и точного смещения в нём от текущего конца WAL. Можно получить доступ к файлу WAL напрямую и скопировать данные из последнего известного окончания WAL до текущего окончания на резервном сервере. При таком подходе интервал возможной потери данных определяется временем цикла работы программы копирования, что может составлять очень малую величину. Так же не потребуется напрасно использовать широкую полосу пропускания для принудительного архивирования частично заполненного файла сегмента. Следует отметить, что на резервном сервере скрипт команды restore_command работает только с файлом WAL целиком, таким образом, копирование данных нарастающим итогом не может быть выполнено на резервном обычными средствами. Это используется только в случае отказа ведущего — когда последний частично сформированный файл WAL предоставляется резервному непосредственно перед переключением. Корректная реализация этого процесса требует взаимодействия скрипта команды restore_command с данными из программы копирования.

3.10.5. Горячий резерв

3.10.5.1. Термин «горячий резерв» используется для описания возможности подключаться к серверу и выполнять запросы на чтение, в то время как сервер находится в режиме резерва или восстановления архива. Это полезно и для целей репликации, и для восстановления желаемого состояния из резервной копии с высокой точностью. Так же термин «горячий резерв» описывает способность сервера переходить из режима восстановления к обычной работе, в то время как пользователи продолжают выполнять запросы и/или их соединения остаются открытыми.

В режиме горячего резерва запросы выполняются примерно так же, как и в обычном режиме, с некоторыми отличиями в использовании и администрировании, описанными ниже.

Когда параметр hot_standby на резервном сервере установлен в true, то он начинает принимать соединения сразу, как только система придёт в согласованное состояние в процессе восстановления. Для таких соединений будет разрешено только чтение, запись невозможна даже во временные таблицы.

Для того, чтобы данные с ведущего сервера были получены на резервном, требуется некоторое время. Таким образом, имеется измеряемая задержка между ведущим и резервным серверами. Поэтому запуск одинаковых запросов примерно в одно время на ведущем и резервном серверах может вернуть разный результат. Можно сказать, что данные на резервном сервере *в конечном счёте согласуются* с ведущим. После того как запись о зафиксированной транзакции воспроизводится на резервном сервере, изменения, совершённые в этой транзакции, становится видны в любых последующих снимках данных на резервном сервере. Снимок может быть сделан в начале каждого запроса или в начале каждой транзакции в зависимости от уровня изоляции транзакции.

Транзакции, запущенные в режиме горячего резерва, могут выполнять следующие команды:

Доступ к данным: SELECT, COPY TO;

Команды для работы с курсором: DECLARE, FETCH, CLOSE;

Параметры: SHOW, SET, RESET;

Команды явного управления транзакциями:

BEGIN, END, ABORT, START TRANSACTION

SAVEPOINT, RELEASE, ROLLBACK TO SAVEPOINT;

Блок EXCEPTION и другие внутренние подчиненные транзакции;

LOCK TABLE, только когда исполняется в явном виде в следующем режиме: ACCESS SHARE, ROW SHARE или ROW EXCLUSIVE;

Планы и ресурсы: PREPARE, EXECUTE, DEALLOCATE, DISCARD;

Дополнения и расширения: LOAD;

UNLISTEN.

Транзакции, запущенные в режиме горячего резерва, никогда не получают ID транзакции и не могут быть записаны в журнал предзаписи. Поэтому при попытке выполнить следующие действия возникнут ошибки:

– Команды манипуляции данными (DML): INSERT, UPDATE, DELETE, COPY FROM, TRUNCATE. Следует отметить, что нет разрешённых действий, которые приводили бы к срабатыванию триггера во время исполнения на резервном сервере. Это ограничение так же касается и временных таблиц, так как строки таблицы не могут быть прочитаны или записаны без обращения к ID транзакции, что в настоящее время не возможно в среде горячего резерва.

– Команды определения данных (DDL): CREATE, DROP, ALTER, COMMENT. Эти ограничения так же относятся и к временным таблицам, так как операции могут потребовать обновления таблиц системных каталогов.

– SELECT ... FOR SHARE | UPDATE, так как блокировка строки не может быть проведена без обновления соответствующих файлов данных.

– Правила для выражений SELECT, которые приводят к выполнению команд DML.

– LOCK которая явно требует режим более строгий чем ROW EXCLUSIVE MODE.

– LOCK в короткой форме с умолчаниями, так как требует ACCESS EXCLUSIVE MODE.

– Команды управления транзакциями, которые в явном виде требуют режим не только для чтения:

BEGIN READ WRITE, START TRANSACTION READ WRITE

SET TRANSACTION READ WRITE, SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE

SET transaction_read_only = off

– Команды двухфазной фиксации: PREPARE TRANSACTION, COMMIT PREPARED, ROLLBACK PREPARED, так как даже транзакции только для чтения нуждаются в записи в WAL на этапе подготовки (первая фаза двухфазной фиксации).

– Обновление последовательностей: nextval(), setval()

– LISTEN, NOTIFY

При обычной работе транзакции «только для чтения» могут использовать команды LISTEN и NOTIFY; таким образом, сеансы горячего резерва работают с несколько большими ограничениями, чем обычные только читающие сеансы. Возможно, что некоторые из этих ограничений будут ослаблены в следующих выпусках.

В режиме горячего резерва параметр transaction_read_only всегда имеет значение true и изменить его нельзя. Но если не пытаться модифицировать содержимое БД, подключение к серверу в этом режиме не отличается от подключений к обычным базам данных. При отработке отказа или переключении ролей база данных переходит в обычный режим работы. Когда сервер меняет режим работы, установленные сеансы остаются подключёнными. После выхода из режима горячего резерва становится возможным запускать пишущие транзакции (даже в сеансах, начатых ещё в режиме горячего резерва).

Пользователи могут узнать, активен ли режим горячего резерва в их сеансе, выполнив команду SHOW transaction_read_only. Кроме того, пользователи могут получить информацию о резервном сервере, используя ряд функций (см. Таблицу 9.86). Эти средства позволяют как создавать программы, учитывающие текущий статус базы данных, так и отслеживать процесс восстановления или разрабатывать более сложные программы для восстановления баз данных в нужном состоянии.

3.10.5.2. Обработка конфликтов запросов

Ведущий и резервный серверы связаны между собой многими слабыми связями. События на ведущем сервере оказывают влияние на резервный. В результате имеется потенциальная возможность отрицательного влияния или конфликта между ними. Наиболее простой для понимания конфликт — быстродействие: если на ведущем происходит загрузка очень большого объёма данных, то происходит создание соответствующего потока записей WAL на резервный сервер. Таким образом, запросы на резервном конкурируют за системные ресурсы, например, ввод-вывод.

Так же может возникнуть дополнительный тип конфликта на сервере горячего резерва. Этот конфликт называется *жёстким конфликтом*, оказывает влияние на запросы, приводя к их отмене, а в некоторых случаях и к обрыву сессии для разрешения конфликтов. Пользователям предоставлен набор средств для обработки подобных конфликтов.

Случаи конфликтов включают:

– Установку эксклюзивной блокировки на ведущем сервере, как с помощью явной команды LOCK, так и при различных DDL, что приводит к конфликту доступа к таблицам на резервном.

– Удаление табличного пространства на ведущем сервере приводит к конфликту на резервном когда запросы используют это пространство для хранения временных рабочих файлов.

– Удаление базы данных на ведущем сервере конфликтует с сессиями, подключёнными к этой БД на резервном.

– Приложение очистки устаревших транзакций из WAL конфликтует с транзакциями на резервном сервере, которые используют снимок данных, который всё ещё видит какие-то из очищенных на ведущем строк.

– Приложение очистки устаревших транзакций из WAL конфликтует с запросами к целевой странице на резервном сервере вне зависимости от того, являются ли данные удалёнными или видимыми.

В этих случаях на ведущем сервере просто происходит ожидание; пользователю следует выбрать какую их конфликтующих сторон отменить. Тем не менее на резервном нет выбора: действия из WAL уже произошли на ведущем, поэтому резервный обязан применить их. Более того, позволять обработчику WAL ожидать неограниченно долго может быть крайне нежелательно, так как отставание резервного сервера от ведущего может всё возрастать. Таким образом, механизм обеспечивает принудительную отмену запросов на резервном сервере, которые конфликтуют с применяемыми записями WAL.

Примером такой проблемы может быть ситуация: администратор на ведущем сервере выполнил команду DROP TABLE для таблицы, которая участвует в запросе на резервном. Этот запрос нельзя будет выполнять дальше, если команда DROP TABLE применится на резервном. Если бы этот запрос выполнялся на ведущем, команда DROP TABLE ждала бы его окончания. Но когда на ведущем выполняется только команда DROP TABLE, ведущий сервер не знает, какие запросы выполняются на резервном, поэтому он не может ждать завершения подобных запросов. Поэтому если записи WAL с изменением прибудут на резервный сервер, когда запрос будет продолжать выполняться, возникнет конфликт. В этом случае резервный сервер должен либо задержать применение этих записей WAL (и всех остальных, следующих за ними), либо отменить конфликтующий запрос, чтобы можно было применить DROP TABLE.

Если конфликтный запрос короткий, обычно желательно разрешить ему завершиться, ненадолго задержав применение записей WAL, но слишком большая задержка в применении WAL обычно нежелательна. Поэтому механизм отмены имеет параметры max_standby_archive_delay и max_standby_streaming_delay, которые определяют максимально допустимое время задержки применения WAL. Конфликтующие запросы будут отменены, если они длятся дольше допустимого времени задержки применения очередных записей WAL. Два параметра существуют для того, чтобы можно было задать разные значения для чтения записей WAL из архива (то есть при начальном восстановлении из базовой копии либо при «навёрстывании» ведущего сервера в случае большого отставания) и для получения записей WAL при потоковой репликации.

На резервном сервере, созданном преимущественно для отказоустойчивости, лучше выставлять параметры задержек относительно небольшими, чтобы он не мог сильно отстать от ведущего из-за задержек, связанных с ожиданием запросов горячего резерва. Однако если резервный сервер предназначен для выполнения длительных запросов, то высокое значение или

даже бесконечное ожидание могут быть предпочтительнее. Тем не менее следует иметь в виду, что длительные запросы могут оказать влияние на другие сессии на резервном сервере в виде отсутствия последних изменений от ведущего из-за задержки применения записей WAL.

В случае, если задержка, определённая max_standby_archive_delay или max_standby_streaming_delay будет превышена, конфликтующий запрос будет отменён. Обычно это выражается в виде ошибки отмены, но в случае проигрывания команды DROP DATABASE обрывается вся конфликтная сессия. Так же, если конфликт произошел при блокировке, вызванной транзакцией в состоянии IDLE, конфликтная сессия разрывается (это поведение может изменить в будущем).

Отменённые запросы могут быть немедленно повторены (конечно после старта новой транзакции). Так как причина отмены зависит от природы проигрываемых записей WAL, запрос, который был отменён, может быть успешно выполнен вновь.

Следует учесть, что параметры задержки отсчитываются от времени получения резервным сервером данных WAL. Таким образом, период дозволенной работы для запроса на резервном сервере никогда не может быть длиннее параметра задержки и может быть существенно короче, если резервный уже находится в режиме задержки в результате ожидания предыдущего запроса или результат не доступен из-за высокой нагрузки обновлений.

Наиболее частой причиной конфликтов между запросами на резервном сервере и проигрыванием WAL является преждевременная очистка. Обычно PG360 допускает очистку старых версий записей при условии что ни одна из транзакций их не видит согласно правилам видимости данных для MVCC. Тем не менее эти правила применяются только для транзакций, выполняемых на главном сервере. Таким образом, допустима ситуация, когда на главном запись уже очищена, но эта же запись всё ещё видна для транзакций на резервном сервере.

Для опытных пользователей следует отметить, что как очистка старых версий строк, так и заморозка версии строки могут потенциально вызвать конфликт с запросами на резервном сервере. Ручной запуск команды VACUUM FREEZE может привести к конфликту, даже в таблице без обновленных и удалённых строк.

Пользователи должны понимать, что регулярное и активное изменение данных в таблицах на ведущем сервере чревато отменой длительных запросов на резервном. В таком случае установка конечного значения для max_standby_archive_delay или max_standby_streaming_delay действует подобно ограничению statement_timeout.

В случае, если количество отменённых запросов на резервном сервере получается неприемлемым, существует ряд дополнительных возможностей. Первая возможность — установить параметр hot_standby_feedback, который не даёт команде VACUUM удалять записи, ставшие недействительными недавно, что предотвращает конфликты очистки. При этом следует учесть, что это вызывает задержку очистки мёртвых строк на ведущем, что может привести к нежелательному распуханию таблицы. Тем не менее в итоге ситуация будет не хуже, чем если бы запросы к резервному серверу исполнялись непосредственно на ведущем, но при этом сохранится положительный эффект от разделения нагрузки. В случае, когда соединение резервных серверов с ведущим часто разрывается, следует скорректировать период, в течение которого обратная связь через hot_standby_feedback не обеспечивается. Например, следует подумать об увеличении max_standby_archive_delay, чтобы запросы отменялись не сразу при конфликтах с архивом WAL в

период разъединения. Также может иметь смысл увеличить max_standby_streaming_delay для предотвращения быстрой отмены запросов из-за полученных записей WAL после восстановления соединения.

Другая возможность — увеличение vacuum_defer_cleanup_age на ведущем сервере таким образом, чтобы мёртвые записи не очищались бы так быстро, как при обычном режиме работы. Это даёт запросам на резервном сервере больше времени на выполнение, прежде чем они могут быть отменены, без увеличения задержки max_standby_streaming_delay. Тем не менее при таком подходе очень трудно обеспечить какое-то определённое окно по времени, так как vacuum_defer_cleanup_age измеряется в количестве транзакций, выполняемых на ведущем сервере.

Количество отменённых запросов и причины отмены можно просмотреть через системное представление pg_stat_database_conflicts на резервном сервере. Системное представление pg_stat_database так же содержит итоговую информацию.

3.10.5.3. Администрирование

Если в файле postgresql.conf параметр hot_standby имеет значение on (значение по умолчанию) и существует файл standby.signal, сервер запустится в режиме горячего резерва. Однако может пройти некоторое время, прежде чем к нему можно будет подключиться, так как он не будет принимать подключения, пока не произведёт восстановление до согласованного состояния, подходящего для выполнения запросов (информация о согласованности состояния записывается на ведущем сервере в контрольной точке). В течение этого периода клиенты при попытке подключения будут получать сообщение об ошибке. Убедиться, что сервер включился в работу, можно либо повторяя попытки подключения из приложения до успешного подключения, либо дождавшись появления в журналах сервера таких сообщений:

```
LOG: entering standby mode

... then some time later ...

LOG: consistent recovery state reached
LOG: database system is ready to accept read only connections
```

Включить горячий резерв нельзя, если WAL был записан в период, когда на ведущем сервере параметр wal_level имел значение, отличное от replica и logical. Достижение согласованного состояния также может быть отсрочено, если имеют место оба этих условия:

– пишущая транзакция имеет более 64 подтранзакций;

– очень длительные пишущие транзакции.

Если применяется файловая репликация журналов («тёплый резерв»), возможно, придётся ожидать прибытия следующего файла WAL (максимальное время ожидания задаётся параметром archive_timeout на ведущем сервере).

Значения некоторых параметров на резервном сервере необходимо изменить при модификации их на ведущем. Для таких параметров значения на резервном сервере должны быть не меньше значений на ведущем. Таким образом, если необходимо увеличить их, сначала необходимо сделать это на резервных серверах, а затем применить изменения на ведущем. И наоборот, если необходимо их уменьшить, сначала необходимо сделать это на ведущем сервере, а потом применить изменения на всех резервных. Если параметры имеют недостаточно большие

значения, резервный сервер не сможет начать работу. В этом случае можно увеличить их и повторить попытку запуска сервера, чтобы он возобновил восстановление. Это касается следующих параметров:

– max_connections;

– max_prepared_transactions;

– max_locks_per_transaction;

– max_wal_senders;

– max_worker_processes.

Очень важно для администратора выбрать подходящие значения для max_standby_archive_delay и max_standby_streaming_delay. Оптимальное значение зависит от приоритетов. Например, если основное назначение сервера — обеспечение высокой степени доступности, то следует установить короткий период, возможно даже нулевой, хотя это очень жёсткий вариант. Если резервный сервер планируется как дополнительный сервер для аналитических запросов, то приемлемой будет максимальная задержка в несколько часов или даже -1, что означает бесконечное ожидание окончания запроса.

Вспомогательные биты статуса транзакций, записанные на ведущем, не попадают в WAL, так что они, скорее всего, будут перезаписаны на нём при работе с данными. Таким образом, резервный сервер будет производить запись на диск, даже если все пользователи только читают данные, ничего не меняя. Кроме того, пользователи будут записывать временные файлы при сортировке больших объёмов и обновлять файлы кеша. Поэтому в режиме горячего резерва ни одна часть базы данных фактически не работает в режиме «только чтение». Следует отметить, что также возможно выполнить запись в удалённую базу данных с помощью модуля dblink и другие операции вне базы данных с применением PL-функций, несмотря на то, что транзакции по- прежнему смогут только читать данные.

Следующие типы административных команд недоступны в течение режима восстановления:

– команды определения данных (DDL): CREATE INDEX и т. п.;

– команды управления правами и назначения владельца: GRANT, REVOKE, REASSIGN;

– команды обслуживания: ANALYZE, VACUUM, CLUSTER, REINDEX.

Следует отметить, что некоторые из этих команд фактически доступны на ведущем сервере для транзакций в режиме только для чтения.

В результате нельзя создать дополнительные индексы или статистику, чтобы они существовали только на резервном. Если подобные административные команды нужны, то их следует выполнить на ведущем сервере, затем эти изменения будут распространены на резервные серверы.

Функции pg_cancel_backend() и pg_terminate_backend() работают на стороне пользователя, но не для процесса запуска, который обеспечивает восстановление. Представление pg_stat_activity не показывает восстанавливаемые транзакции как активные. Поэтому представление pg_prepared_xacts всегда пусто в ходе восстановления. Если требуется разобрать сомнительные подготовленные транзакции, следует обратиться к pg_prepared_xacts на ведущем и выполнить команды для разбора транзакций там либо разобрать их по окончании восстановления.

Функция pg_locks отображает блокировки, происходящие в процессе работы сервера как обычно. pg_locks так же показывает виртуальные транзакции, обработанные процессом запуска,

которому принадлежат все AccessExclusiveLocks, наложенные транзакциями в режиме восстановления. Следует отметить, что процесс запуска не запрашивает блокировки, чтобы внести изменения в базу данных, поэтому блокировки, отличные от AccessExclusiveLocks не показываются в pg_locks для процесса запуска, подразумевается их существование.

Команды управления файлами WAL, например pg_start_backup, pg_switch_wal и т. д. не будут работать во время восстановления.

Динамически загружаемые модули работать будут, включая pg_stat_statements.

Рекомендательная блокировка работает обычно при восстановлении, включая обнаружение взаимных блокировок. Следует отметить, что рекомендательная блокировка никогда не попадает в WAL, таким образом для рекомендательной блокировки как на ведущем сервере, так и на резервном, невозможен конфликт с проигрыванием WAL. Но возможно получение рекомендательной блокировки на ведущем сервере, а затем получение подобной рекомендательной блокировки на резервном. Рекомендательная блокировка относится только к серверу, на котором она получена.

Системы репликации на базе триггеров, подобные Slony, Londiste и Bucardo не могут запускаться на резервном сервере вовсе, хотя они превосходно работают на ведущем до тех пор, пока не будет подана команда не пересылать изменения на резервный. Проигрывание WAL не основано на триггерах, поэтому поток WAL нельзя транслировать с резервного сервера в другую систему, которая требует дополнительной записи в БД или работает на основе триггеров.

Новые OID не могут быть выданы, хотя, например генераторы UUID смогут работать, если они не пытаются записывать новое состояние в базу данных.

В настоящий момент создание временных таблиц недопустимо при транзакции только для чтения, в некоторых случаях существующий скрипт будет работать неверно. Это ограничение может быть ослаблено в следующих выпусках. Это одновременно требование SQL стандарта и техническое требование.

Команда DROP TABLESPACE может быть выполнена только если табличное пространство пусто. Некоторые пользователи резервного сервера могут активно использовать табличное пространство через параметр temp_tablespaces. Если имеются временные файлы в табличных пространствах, все активные запросы отменяются для обеспечения удаления временных файлов, затем табличное пространство может быть удалено и продолжено проигрывание WAL.

Выполнение команды DROP DATABASE или ALTER DATABASE ... SET TABLESPACE на ведущем сервере приводит к созданию записи в WAL, которая вызывает принудительное отключение всех пользователей, подключённых к этой базе данных на резервном. Это происходит немедленно, вне зависимости от значения max_standby_streaming_delay. Следует отметить, что команда ALTER DATABASE ... RENAME не приводит к отключению пользователей, так что обычно она действует незаметно, хотя в некоторых случаях возможны сбои программ, которые зависят от имени базы данных.

Если в обычном режиме (не в режиме восстановления) выполнить DROP USER или DROP ROLE для роли с возможностью подключения, в момент, когда этот пользователь подключён, на данном пользователе это никак не отразится — он останется подключённым. Однако переподключиться он уже не сможет. Это же поведение действует в режиме восстановления — если выполнить DROP USER на ведущем сервере, пользователь не будет отключён от резервного.

Сборщик статистики работает во время восстановления. Все операции сканирования, чтения, блоки, использование индексов и т. п. будут записаны обычным образом на резервном сервере. Действия, происходящие при проигрывании, не будут дублировать действия на ведущем сервере, то есть проигрывание команды вставки не увеличит значение столбца Inserts в представлении pg_stat_user_tables. Файлы статистики удаляются с началом восстановления, таким образом, статистика на ведущем сервере и резервном будет разной. Это является особенностью, не ошибкой.

Автоматическая очистка не работает во время восстановления. Она запустится в обычном режиме после завершения восстановления.

Во время восстановления работает процесс контрольных точек и процесс фоновой записи. Процесс контрольных точек обрабатывает точки перезапуска (подобные контрольным точкам на ведущем сервере), а процесс фоновой записи выполняет обычные операции по очистке блоков. В том числе он может обновлять вспомогательные биты, сохранённые на резервном сервере. Во время восстановления принимается команда CHECKPOINT, но она производит точку перезапуска, а не создаёт новую точку восстановления.

3.10.5.4. Ссылки на параметры горячего резерва

На ведущем могут применяться параметры wal_level и vacuum_defer_cleanup_age. Параметры max_standby_archive_delay и max_standby_streaming_delay на ведущем не действуют.

На резервном сервере могут применяться параметры hot_standby, max_standby_archive_delay и max_standby_streaming_delay. Параметр vacuum_defer_cleanup_age на нём не действует, пока сервер остаётся в режиме резервного сервера. Но если он станет ведущим, его значение вступит в силу.

3.10.5.5. Ограничения

Имеются следующие ограничения горячего резерва:

– Требуется информация о всех запущенных транзакциях перед тем как будет создан снимок данных. Транзакции, использующие большое количество подтранзакций (в настоящий момент больше 64), будут задерживать начало соединения только для чтения до завершения самой длинной пишущей транзакции. При возникновении этой ситуации поясняющее сообщение будет записано в журнал сервера.

– Подходящие стартовые точки для запросов на резервном сервере создаются при каждой контрольной точке на главном. Если резервный сервер отключается, в то время как главный был в отключённом состоянии, может оказаться невозможным возобновить его работу в режиме горячего резерва, до того, как запустится ведущий и добавит следующие стартовые точки в журналы WAL. Подобная ситуация не является проблемой для большинства случаев, в которых она может произойти. Обычно, если ведущий сервер выключен и больше не доступен, это является следствием серьёзного сбоя и в любом случае требует преобразования резервного в новый ведущий. Так же в ситуации, когда ведущий отключён намеренно, проверка готовности резервного к преобразованию в ведущий тоже является обычной процедурой.

– В конце восстановления блокировки AccessExclusiveLocks, вызванные подготовленными транзакциями, требуют удвоенное, в сравнении с нормальным, количество блокировок записей

таблицы. Если планируется использовать либо большое количество конкурирующих подготовленных транзакций, обычно вызывающие AccessExclusiveLocks, либо большие транзакции с применением большого количества AccessExclusiveLocks, то рекомендуется выбрать большое значение параметра max_locks_per_transaction, возможно в два раза большее, чем значение параметра на ведущем сервере. Всё это не имеет значения, когда max_prepared_transactions равно 0.

– Уровень изоляции транзакции Serializable в настоящее время недоступен в горячем резерве. Попытка выставить для транзакции такой уровень изоляции в режиме горячего резерва вызовет ошибку.

3.11. Мониторинг работы СУБД

Для мониторинга работы СУБД и анализа её производительности существуют различные инструменты, среди которых и команды мониторинга Unix, такие как ps, top, iostat, и vmstat. Кроме того, после обнаружения запроса с низкой производительностью может потребоваться дополнительное исследование с использованием PG360 команды EXPLAIN.

3.11.1. Сборщик статистики

Сборщик статистики в PG360 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, PG360 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.

3.11.1.1. 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.

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. (Для того чтобы обычные пользователи не скрывали свою работу от

администратора СУБД, изменять эти параметры с помощью команды SET могут только суперпользователи.)

Сборщик статистики использует временные файлы для передачи собранной информации другим процессам PG360. 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, все статистические данные счётчиков сбрасываются.

3.11.1.2. Просмотр статистики

Для просмотра текущего состояния системы предназначены несколько предопределённых представлений:

– pg_stat_activity – содержит одну строка для каждого серверного процесса с информацией о текущей активности процесса, включая его состояние и текущий запрос.

– pg_stat_replication – содержит по одной строке для каждого процесса-передатчика WAL со статистикой по репликации на ведомом сервере, к которому подключён этот процесс.

– pg_stat_wal_receiver – содержит только одну строка со статистикой приёмника WAL, полученной с сервера, на котором работает приёмник.

– pg_stat_subscription – содержит как минимум одну строка для подписки, сообщающая о рабочих процессах подписки.

– pg_stat_ssl – содержит одну строку для каждого подключения (обычного и реплицирующего), в которой показывается информация об использовании SSL для данного подключения.

– pg_stat_progress_analyze – содержит по одной строке с текущим состоянием для каждого обслуживающего процесса (включая рабочие процессы автоочистки), в котором работает ANALYZE.

– pg_stat_progress_create_index – содержит по одной строке с текущим состоянием для каждого обслуживающего процесса, в котором выполняется CREATE INDEX или REINDEX.

– pg_stat_progress_vacuum – содержит по одной строке с текущим состоянием для каждого обслуживающего процесса (включая рабочие процессы автоочистки), в котором работает VACUUM.

– pg_stat_progress_cluster – содержит по одной строке с текущим состоянием для каждого обслуживающего процесса, в котором выполняется CLUSTER или VACUUM FULL.

– pg_stat_progress_basebackup – содержит по одной строке с текущим состоянием для каждого передающего WAL процесса, транслирующего базовую копию.

В дополнение к ним есть несколько других представлений, позволяющих просмотреть результаты сбора статистики:

– pg_stat_archiver – содержит только одну строку со статистикой работы процесса архивации WAL.

– pg_stat_bgwriter – содержит только одну строку со статистикой работы фонового процесса записи.

– pg_stat_database – содержит одну строку для каждой базы данных со статистикой на уровне базы.

– pg_stat_database_conflicts – содержит 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 – содержит по одной строке на каждую таблицу в текущей базе данных со статистикой по обращениям к этой таблице.

– pg_stat_sys_tables – содержит данные аналогично pg_stat_all_tables, за исключением того, что отображаются только системные таблицы.

– pg_stat_user_tables – содержит данные аналогично pg_stat_all_tables, за исключением того, что отображаются только пользовательские таблицы.

– pg_stat_xact_all_tables – содержит данные аналогично 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 – содержит данные аналогично pg_stat_xact_all_tables , за исключением того, что отображаются только системные таблицы.

– pg_stat_xact_user_tables – содержит данные аналогично pg_stat_xact_all_tables, за исключением того, что отображаются только пользовательские таблицы.

– pg_stat_all_indexes – содержит по одной строке для каждого индекса в текущей базе данных со статистикой по обращениям к этому индексу.

– pg_stat_sys_indexes – содержит данные аналогично pg_stat_all_indexes, за исключением того, что показываются только индексы по системным таблицам.

– pg_stat_user_indexes – содержит данные аналогично pg_stat_all_indexes , за исключением того, что показываются только индексы по пользовательским таблицам.

– pg_statio_all_tables – содержит по одной строке для каждой таблицы в текущей базе данных со статистикой по операциям ввода/вывода с этой таблицей.

– pg_statio_sys_tables – содержит данные аналогично pg_statio_all_tables, за исключением того, что показываются только системные таблицы.

– pg_statio_user_tables – содержит данные аналогично pg_statio_all_tables, за исключением того, что показываются только пользовательские таблицы.

– pg_statio_all_indexes – содержит по одной строке для каждого индекса в текущей базе данных со статистикой по операциям ввода/вывода для этого индекса.

– pg_statio_sys_indexes – содержит данные аналогично pg_statio_all_indexes, за исключением того, что показываются только индексы по системным таблицам.

– pg_statio_user_indexes – содержит данные аналогично pg_statio_all_indexes, за исключением того, что показываются только индексы по пользовательским таблицам.

– pg_statio_all_sequences – содержит по одной строке для каждой последовательности в текущей базе данных со статистикой по операциям ввода/вывода с этой последовательностью.

– pg_statio_sys_sequences – содержит данные аналогично pg_statio_all_sequences, за исключением того, что показываются только системные последовательности.

– pg_statio_user_sequences – содержит данные аналогично pg_statio_all_sequences, за исключением того, что показываются только пользовательские последовательности.

– pg_stat_user_functions – содержит по одной строке для каждой отслеживаемой функции со статистикой по выполнениям этой функции.

– pg_stat_xact_user_functions – содержит данные 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 – содержит одну строку со статистикой работы для каждого SLRU-кеша.

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).

Часть информации в представлениях с динамическими статистическими данными скрыта по соображениям безопасности. Обычные пользователи могут получать информацию только о своих собственных сеансах (сеансах, принадлежащих роли, членами которой они являются). В строках, относящимся к другим сеансам, многие столбцы будут содержать NULL. При этом информация о самом сеансе и его общих свойствах, например, текущем пользователе и базе данных, доступна всем пользователям. Суперпользователи и члены встроенной роли pg_read_all_stats могут получить всю информацию о любом сеансе.

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

Представления pg_statio_ используются 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: PG360 обрабатывает дисковый ввод/вывод так, что данные, не находящиеся в буферном кеше PG360, 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 PG360 рекомендуется использовать сборщик статистики PG360 в сочетании с программами ОС, которые дают более полное представление о том, как ядро осуществляет ввод/вывод.

Подробное описание представлений приведено в Приложении А.

3.11.1.3. Статистические функции

Статистическую информацию можно просматривать и другими способами. Для этого можно написать запросы, использующие те же функции доступа к статистике, что лежат в основе описанных выше стандартных представлений. В качестве аргумента функции, предоставляющие доступ к статистике на уровне базы, принимают OID базы данных, по которой должна быть выдана информация. Функции, которые работают на уровне таблиц и индексов, принимают в качестве аргумента OID таблицы или индекса. Аргументом для функции, предоставляющей статистику на уровне функций, является OID функции. Необходимо обратить внимание, что с

помощью этих функций можно получить информацию по таблицам, индексам и функциям исключительно в текущей базе данных.

Дополнительные функции, связанные со сбором статистики, перечислены в Таблице 6.

Таблица 6. Дополнительные статистические функции

Функция	**Описание**
pg_backend_pid() → integer	Выдаёт идентификатор серверного процесса, обслуживающего текущий сеанс
pg_stat_get_activity(integer) → setof record	Возвращает запись с информацией о серверном процессе с заданным ID или по одной строке для каждого активного серверного процесса, если был указан NULL. Возвращаемые поля являются подмножеством столбцов представления pg_stat_ activity .
pg_stat_get_snapshot_timestamp() → timestamp with time zone	Возвращает время текущего снимка статистики.
pg_stat_clear_snapshot( ) → void	Сбрасывает текущий снимок статистики.
pg_stat_reset() → void	Обнуляет все статистические счётчики в текущей базе данных. По умолчанию доступ к этой функции имеют только суперпользователи, но право на её выполнение (EXECUTE) можно дать и другим пользователям.
pg_stat_reset_shared(text) → void	Обнуляет некоторые статистические счётчики на уровне кластера, в зависимости от аргумента. Аргумент может принимать значения bgwriter и archiver, с которыми обнуляются все счётчики в представлении pg_stat_bgwriter или pg_stat_archiver , соответственно. По умолчанию доступ к этой функции имеют только суперпользователи, но право на её выполнение (EXECUTE) можно дать и другим пользователям.
pg_stat_reset_single_table_counters (oid) → void	Обнуляет статистику по отдельной таблице или индексу в текущей базе данных. По умолчанию доступ к этой функции имеют только суперпользователи, но право на её выполнение (EXECUTE) можно дать и другим пользователям.
pg_stat_reset_single_function_counters (oid) → void	Обнуляет статистику для отдельной функции в текущей базе данных. По умолчанию доступ к этой функции имеют только суперпользователи, но право на её выполнение (EXECUTE) можно дать и другим пользователям.
pg_stat_reset_slru (text) → void	Обнуляет статистику для отдельного SLRU-кеша или для всех SLRU-кешей в кластере. Если в аргументе передаётся NULL, обнуляются все счётчики, показанные в представлении pg_stat_slru , для всех SLRU-кешей. Также в аргументе можно передать текстовое значение CommitTs, MultiXactMember, MultiXactOffset, Notify, Serial, Subtrans или Xact, чтобы сбросить счётчики только для указанного кеша. Если в аргументе передаётся other (на самом деле это может быть любая нераспознанная строка), обнулены будут все счётчики для всех кешей, кроме перечисленных поимённо выше. По умолчанию доступ к этой функции имеют только суперпользователи, но право на её выполнение (EXECUTE) можно дать и другим пользователям.

Функция pg_stat_get_activity, на которой основано представление pg_stat_activity, возвращает набор строк, содержащих всю доступную информацию о каждом серверном процессе. Иногда более удобным оказывается получение только части этой информации. В таких случаях можно использовать набор более старых функций, дающих доступ к статистике на уровне серверных процессов. Эти функции используют идентификатор серверного процесса, значение которого варьируется от единицы до числа активных в настоящий момент серверных процессов. Функция pg_stat_get_backend_idset генерирует по одной строке для каждого активного серверного процесса, что необходимо для вызова этих функций. Например, для того, чтобы отобразить значения PID и текущие запросы всех серверных процессов, нужно выполнить команду:

```
    SELECT pg_stat_get_backend_pid(s.backendid) AS pid,
pg_stat_get_backend_activity(s.backendid) AS query
    FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

Таблица 7. Статистические функции на уровне серверных процессов

Функция	**Описание**
pg_stat_get_backend_idset() → setof integer	Выдаёт идентификаторы активных в настоящий момент серверных процессов (от 1 до числа активных процессов).
pg_stat_get_backend_activity(integer) → text	Выдаёт текст последнего запроса этого серверного процесса.
pg_stat_get_backend_activity_start(integer) → timestamp with time zone	Выдаёт время начала выполнения последнего запроса в рамках данного процесса.
pg_stat_get_backend_client_addr(integer) → inet	Выдаёт IP-адрес клиента, подключённого к этому серверному процессу.
pg_stat_get_backend_client_port(integer) → integer	Выдаёт номер TCP-порта, который клиент использует для взаимодействия с сервером.
pg_stat_get_backend_dbid(integer) → oid	Выдаёт OID базы данных, к которой подключён этот серверный процесс.
pg_stat_get_backend_pid(integer) → integer	Выдаёт идентификатор (PID) этого серверного процесса.
pg_stat_get_backend_start(integer) → timestamp with time zone	Выдаёт время, когда был запущен данный процесс.
pg_stat_get_backend_userid(integer) → oid	Выдаёт OID пользователя, подключённого к этому серверному процессу.
pg_stat_get_backend_wait_event_type(integer) → text	Выдаёт имя типа ожидаемого события, если серверный процесс сейчас находится в состоянии ожидания, или NULL в противном случае.
pg_stat_get_backend_wait_event(integer) → text	Выдаёт имя ожидаемого события, если серверный процесс сейчас находится в состоянии ожидания, или NULL в противном случае.
pg_stat_get_backend_xact_start(integer) → timestamp with time zone	Выдаёт время начала текущей транзакции в данном процессе.

3.11.2. 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 PG360.

– 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 так же то, как меняется конкуренция в зависимости от загруженности базы.

3.11.3. Отслеживание выполнения

В PG360 имеется возможность отслеживать выполнение определённых команд. В настоящее время такое отслеживание поддерживается только для команд ANALYZE, CLUSTER,

CREATE INDEX, VACUUM и BASE_BACKUP (то есть для команды репликации, которую выполняет pg_basebackup).

Подробное описание отслеживания выполнения для команд ANALYZE, CLUSTER, CREATE INDEX, VACUUM, BASE_BACKUP приведено в Приложении Б,

3.12. Мониторинг использования диска

3.12.1. Определение использования диска

Для каждой таблицы создаётся первичный дисковый файл кучи (heap), в котором хранится большая часть данных. Если в таблице есть столбцы с потенциально большими значениями, то также может быть и ассоциированный с этой таблицей файл TOAST, который используется для хранения слишком больших значений, не умещающихся в главной таблице. На каждую таблицу TOAST, если она существует, будет существовать один индекс. Также там могут быть и индексы, ассоциированные с базовой таблицей. Каждая таблица и индекс хранятся в отдельном дисковом файле — возможно более чем в одном файле, если размер этого файла превышает один гигабайт.

Можно осуществлять мониторинг дискового пространства тремя способами:

– используя SQL-функции,

– используя модуль oid2name или

– просматривая системные каталоги вручную.

SQL-функции являются наиболее простыми и рекомендуются для использования.

Используя программу psql после применения команд VACUUM или ANALYZE, можно выполнить такой запрос, чтобы увидеть сколько дискового пространства использует какая-либо таблица:

```
SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'customer';

pg_relation_filepath | relpages
----------------------+----------
base/16384/16806     |     60
(1 row)
```

Каждая страница обычно равна 8kb. (Помните, что relpages обновляется только командами VACUUM, ANALYZE, и несколькими командами DDL, такими как CREATE INDEX). Путь к файлу представляет интерес, если необходимо проанализировать непосредственно файл на диске.

Чтобы посмотреть пространство, используемое таблицами TOAST, используется следующий запрос:

```
SELECT relname, relpages FROM pg_class,
(SELECT reltoastrelid FROM pg_class
WHERE relname = 'customer') AS ss WHERE oid = ss.reltoastrelid OR
oid = (SELECT indexrelid FROM pg_index
WHERE indrelid = ss.reltoastrelid) ORDER BY relname;

relname                   | relpages
----------------------+----------
pg_toast_16806       |     0
```

```
pg_toast_16806_index |   1
```

Можно посмотреть размеры индексов, выполнив запрос:

```
SELECT c2.relname, c2.relpages
FROM pg_class c, pg_class c2, pg_index i WHERE c.relname =
'customer' AND
c.oid = i.indrelid AND
c2.oid = i.indexrelid ORDER BY c2.relname;

relname             | relpages
--------------------+----------
customer_id_index | 26
```

Также можно найти самые большие таблицы и индексы, используя эти запросы:

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC;

relname | relpages
-----------------------+-------------
bigtable | 3290
customer | 3144
```

3.12.2. Ошибка переполнения диска

Основная цель мониторинга дисков для администратора БД – предотвратить их переполнение. Если переполнится диск с данными, повреждения данных не произойдёт, но могут не выполняться полезные действия. Если же переполнится диск, содержащий файлы WAL, это может привести к аварийному сбою сервера с последующим его отключением.

Если нельзя освободить дополнительное пространство на диске, удалив какие-либо другие файлы, то можно перенести часть файлов базы данных на другие файловые системы, с помощью создания табличных пространств.

3.13. Журнал упреждающей записи

3.13.1 *Журнал упреждающей записи* (WAL) – это стандартный метод обеспечения целостности данных. Принцип действия WAL состоит в том, что изменения в файлах с данными (где находятся таблицы и индексы) должны записываться только после того, как эти изменения были занесены в журнал, т. е. после того как записи журнала, описывающие данные изменения, будут сохранены на постоянное устройство хранения. Если следовать этой процедуре, то записывать страницы данных на диск после подтверждения каждой транзакции нет необходимости, потому что если случится сбой, то будет возможность восстановить базу данных с помощью журнала: любые изменения, которые не были применены к страницам с данными, могут быть воссозданы из записей журнала.

Результатом использования WAL является значительное уменьшение количества запросов записи на диск, потому что для гарантии, что транзакция подтверждена, в записи на диск нуждается только файл журнала, а не каждый файл данных изменённый в результате транзакции.

Файл журнала записывается последовательно и таким образом, затраты на синхронизацию журнала намного меньше, чем затраты на запись страниц с данными. Это особенно справедливо для серверов, которые обрабатывают много маленьких транзакций, изменяющих разные части хранилища данных. Таким образом, когда сервер обрабатывает множество мелких конкурентных транзакций, для подтверждения многих транзакций достаточно одного вызова fsync на файл журнала.

WAL также делает возможным поддержку онлайнового резервного копирования и восстановления на определённый момент времени. С помощью архивирования данных WAL поддерживается возврат к любому моменту времени, который доступен в данных WAL: мы просто устанавливаем предыдущую физическую резервную копию базы данных и воспроизводим журнал WAL до нужного момента времени. Более того, физическая резервная копия не должна быть мгновенным снимком состояния баз данных — если она была сделана некоторое время назад, воспроизведение журнала WAL за этот период исправит все внутренние несоответствия.

3.13.2 Настройка WAL

Существует несколько конфигурационных параметров относящихся к WAL, которые влияют на производительность СУБД.

Контрольные точки— это точки в последовательности транзакций, в которых гарантируется, что файлы с данными и индексами были обновлены всей информацией записанной перед контрольной точкой. Во время контрольной точки, все «грязные» страницы данных, находящиеся в памяти, сохраняются на диск, а в файл журнала записывается специальная запись контрольной точки. (Сделанные изменения были перед этим записаны в файлы WAL.) В случае краха процедура восстановления ищет последнюю запись контрольной точки, чтобы определить эту точку в журнале (называемую записью REDO), от которой процедура должна начать операцию воспроизведения изменений. Любые изменения файлов данных перед этой точкой гарантированно находятся уже на диске. Таким образом, после контрольной точки, сегменты журнала, которые предшествуют записи воспроизведения, больше не нужны и могут быть удалены или пущены в циклическую перезапись. (Когда архивирование WAL будет завершено, сегменты журнала должны быть архивированы перед их удалением или циклической перезаписи.)

Запись всех «грязных» страниц данных из памяти на диск, которая требуется для контрольной точки, может вызвать значительную нагрузку на дисковый ввод/вывод. По этой причине, активность записи по контрольной точке регулируется так, что ввод/вывод начинается при старте контрольной точки и завершается перед стартом следующей контрольной точки; это минимизирует потерю производительности во время прохождения контрольных точек.

Отдельный серверный процесс контрольных точек автоматически выполняет контрольные точки с заданной частотой. Контрольные точки производятся каждые checkpoint_timeout секунд либо при приближении к пределу max_wal_size, если это имеет место раньше. Значения по умолчанию: 5 минут и 1 Гбайт, соответственно. Если после предыдущей контрольной точки новые записи WAL не добавились, следующие контрольные точки будут пропущены, даже если проходит время checkpoint_timeout. (Если применяется архивация WAL и необходимо установить нижний предел для частоты архивации, чтобы ограничить потенциальную потерю данных,

следует настраивать параметр archive_timeout, а не параметры контрольных точек.) Также можно выполнить контрольную точку принудительно, воспользовавшись SQL-командой CHECKPOINT.

Уменьшение значений checkpoint_timeout и/или max_wal_size приводит к учащению контрольных точек. Это позволяет ускорить восстановление после краха (поскольку для воспроизведения нужно меньше данных), но с другой стороны нужно учитывать дополнительную нагрузку, возникающую вследствие более частого сброса «грязных» страниц данных на диск. Если включён режим full_page_writes (по умолчанию это так), нужно учесть и ещё один фактор. Для обеспечения целостности страницы данных, при первом изменении страницы данных после контрольной точки эта страница записывается в журнал целиком. В данном случае, чем меньше интервал между контрольными точками, тем больше объём записи в журнал WAL, так что это частично дискредитирует идею уменьшения интервала записи, и в любом случае приводит к увеличению объёма обмена с диском

Контрольные точки довольно дороги с точки зрения ресурсов, во-первых, потому что они требуют записи всех «грязных» буферов из памяти на диск, и во-вторых потому что они создают дополнительный трафик WAL, о чём говорилось выше. Таким образом, необходимо установить параметры контрольных точек так, чтобы контрольные точки не выполнялись слишком часто. Для простой проверки параметров контрольной точки можно установить параметр checkpoint_warning. Если промежуток времени между контрольными точками будет меньше чем количество секунд, заданное параметром checkpoint_warning, то в журнал сервера будет выдано сообщение с рекомендацией увеличить max_wal_size. Эпизодическое появление такого сообщения не является поводом для беспокойства. Но если оно появляется часто, необходимо увеличить значения параметров управления контрольными точками. Массовые операции, такие как COPY с большим объёмом данных, могут привести к появлению нескольких таких предупреждений, если был установлен параметр max_wal_size достаточно большим.

На платформах Linux и POSIX параметр checkpoint_flush_after позволяет принудить ОС к сбросу страниц, записываемых во время контрольной точки, при накоплении заданного количества байт. Если его не настроить, эти страницы могут оставаться в кеше страниц ОС, что повлечёт затормаживание при выполнении fsync в конце контрольной точки. Этот параметр часто помогает уменьшить задержки транзакций, но может оказать и негативное влияние на производительность; особенно, когда объём нагрузки больше shared_buffers, но меньше кеша страниц в ОС.

Число файлов сегментов WAL в каталоге pg_wal зависит от min_wal_size, max_wal_size и объёма WAL, сгенерированного в предыдущих циклах контрольных точек. Когда старые файлы сегментов оказываются не нужны, они удаляются или перерабатываются (то есть переименовываются, чтобы стать будущими сегментами в нумерованной последовательности). Если вследствие кратковременного скачка интенсивности записи в журнал, предел max_wal_size превышается, ненужные файлы сегментов будут удаляться, пока система не опустится ниже этого предела. Оставаясь ниже этого предела, система перерабатывает столько файлов WAL, сколько необходимо для покрытия ожидаемой потребности до следующей контрольной точки, и удаляет остальные. Значение min_wal_size ограничивает снизу число файлов WAL, которые будут переработаны для будущего использования; такой объём WAL всегда будет перерабатываться,

даже если система простаивает и оценка использования говорит, что нужен совсем небольшой WAL.

Вне зависимости от max_wal_size, последние файлы WAL в объёме wal_keep_size мегабайт и ещё один дополнительный файл WAL сохраняются в любом случае. Кроме того, если применяется архивация WAL, старые сегменты не могут быть удалены или переработаны, пока они не будут заархивированы. Если WAL архивируется медленнее, чем генерируется, либо если команда archive_command постоянно даёт сбои, старые файлы WAL будут накапливаться в pg_wal, пока ситуация не будет разрешена. Медленно работающий или отказавший ведомый сервер, использующий слот репликации, даст тот же эффект.

В режиме восстановления архива или горячего резерва сервер периодически выполняет *точки перезапуска*, которые похожи на контрольные точки в обычном режиме работы: сервер принудительно сбрасывает своё состояние на диск, обновляет файл pg_control, чтобы показать, что уже обработанные данные WAL не нужно сканировать снова, и затем перерабатывает все старые файлы сегментов журнала в каталоге pg_wal. Точки перезапуска не могут выполняться чаще, чем контрольные точки на главном сервере, так как они могут происходить только в записях контрольных точек. Точка перезапуска производится, когда достигается запись контрольной точки и после предыдущей точки перезапуска прошло не меньше checkpoint_timeout секунд или размер WAL может превысить max_wal_size. Однако из-за того, что на время выполнения точек перезапуска накладываются ограничения, max_wal_size часто превышается при восстановлении, вплоть до объёма WAL, записываемого в цикле между контрольными точками. (Значение max_wal_size никогда и не было жёстким пределом, так что всегда следует оставлять приличный запас сверху, чтобы не остаться без свободного места на диске.)

Наиболее часто используются две связанные с WAL внутренние функции: XLogInsertRecord и XLogFlush. XLogInsertRecord применяется для добавления записи в буферы WAL в разделяемой памяти. Если места для новой записи недостаточно, XLogInsertRecord придётся записать (переместить в кеш ядра) несколько заполненных буферов WAL. Это нежелательно, так как XLogInsertRecord используется при каждом изменении в базе данных на низком уровне (например, при добавлении строки) в момент, когда установлена исключительная блокировка задействованных страниц данных, поэтому данная операция должна быть максимально быстрой. Что ещё хуже, запись буферов WAL может также повлечь создание нового сегмента журнала, что займёт ещё больше времени. Обычно буферы WAL должны записываться и сохраняться на диске в функции XLogFlush, которая вызывается, по большей части, при фиксировании транзакции, чтобы результаты транзакции сохранились в надёжном хранилище. В системах с интенсивной записью в журнал вызовы XLogFlush могут иметь место не так часто, чтобы XLogInsertRecord не приходилось производить запись. В таких системах следует увеличить число буферов WAL, изменив параметр wal_buffers. Когда включён режим full_page_writes и система очень сильно загружена, увеличение wal_buffers поможет сгладить скачки во времени ответа в период сразу после каждой контрольной точки.

Параметр commit_delay определяет, на сколько микросекунд будет засыпать ведущий процесс группы, записывающий в журнал, после получения блокировки в XLogFlush, пока подчинённые формируют очередь на запись. Во время этой задержки другие серверные процессы смогут добавлять записи в WAL буферы журнала, чтобы все эти записи сохранились на диск в

результате одной операции синхронизации, которую выполнит ведущий. Ведущий процесс не засыпает, если отключён режим fsync, либо число сеансов с активными транзакциями меньше commit_siblings, так как маловероятно, что какой-либо другой сеанс зафиксирует транзакцию в ближайшее время. На некоторых платформах, разрешение этого таймера сна составляет 10 миллисекунд, так что любое значение параметра commit_delay от 1 до 10000 микросекунд будет действовать одинаково. Кроме того, в некоторых системах состояние сна может продлиться несколько дольше, чем требует параметр.

Так как цель commit_delay состоит в том, чтобы позволить стоимости каждой операции синхронизации амортизироваться через параллельную фиксацию транзакций (потенциально за счёт задержки транзакции), необходимо определить количество той стоимости, прежде чем урегулирование сможет быть выбрано разумно. Чем выше стоимость, тем более эффективный будет commit_delay в увеличении пропускной способности транзакций в какой-то степени. Программа pg_test_fsync может использоваться, чтобы измерить среднее время в микросекундах, которое занимает одиночная работа сброса WAL на диск. Значение половины среднего времени сообщаемого программой рекомендуется в качестве отправной точки для использования значения в параметре commit_delay при оптимизации для конкретного объёма работы, и говорит о том, сколько нужно времени для синхронизации сброса единственной операции записи 8 Кбайт. Настройка параметра commit_delay особенно полезна в случае хранения WAL в хранилище с высокоскоростными дисками, такими как твердотельные накопители (SSD) или RAID-массивы с кешем записи и аварийным питанием на батарее; но это определённо должно тестироваться на репрезентативной рабочей нагрузке. Более высокие значения commit_siblings должны использоваться в таких случаях, тогда как меньшие значения commit_siblings часто полезны на носителях с большими задержками. Увеличение значения параметра commit_delay может увеличить задержку транзакции настолько, что пострадает общая производительность транзакций.

Даже если commit_delay равен нулю (значение по умолчанию), групповая фиксация все равно может произойти, но группа будет состоять только из тех сеансов, которым понадобилось сбросить записи о фиксации на диск за то время, пока происходил предыдущий сброс. Чем больше сеансов, тем чаще это происходит даже при нулевом commit_delay, поэтому увеличение этого параметра может и не оказать заметного действия. Установка commit_delay имеет смысл в двух случаях: (1) когда несколько транзакций одновременно фиксируют изменения, (2) либо когда частота фиксаций ограничена пропускной способностью дисковой подсистемы. Однако при задержке из-за низкой скорости вращения диска, эта настройка может оказаться полезной даже всего при двух сеансах.

Параметр wal_sync_method определяет, как PG360 будет обращаться к ядру, чтобы принудительно сохранить WAL на диск. Все методы должны быть одинаковыми в плане надёжности, за исключением fsync_writethrough, который может иногда принудительно сбрасывать кеш диска, даже если другие методы не делают этого. Однако какой из них самый быстрый, во многом определяется платформой; можно протестировать скорость, используя модуль pg_test_fsync. Данный параметр не имеет значения, если fsync выключен.

Включение параметра конфигурации wal_debug (предоставляется, если PG360 был скомпилирован с его поддержкой) будет приводить к тому, что все вызовы связанных с WAL

функций XLogInsertRecord и XLogFlush будут протоколироваться в журнале сервера. В будущем данный параметр может быть заменён более общим механизмом.

3.14. Логическая репликация

Логическая репликация – это метод репликации объектов данных и изменений в них, использующий репликационные идентификаторы (обычно это первичный ключ). Эта репликация называется «логической», в отличие от физической, которая построена на точных адресах блоков и побайтовом копировании. PG360 поддерживает оба механизма одновременно. Логическая репликация позволяет более детально управлять репликацией данных и аспектами безопасности.

В логической репликации используется модель публикаций/подписок с одним или несколькими подписчиками, которые подписываются на одну или несколько публикаций на публикующем узле. Подписчики получают данные из публикаций, на которые они подписаны, и могут затем повторно опубликовать данные для организации каскадной репликации или более сложных конфигураций.

Для осуществления логической репликации необходимо установить несколько параметров конфигурации.

На публикующем сервере параметр wal_level должен иметь значение logical, а в max_replication_slots должно быть задано число не меньше ожидаемого числа подписчиков плюс некоторый резерв для синхронизации таблиц. А в max_wal_senders должно быть значение как минимум равное max_replication_slots плюс число возможных физических реплик, работающих одновременно.

Также на стороне подписчика необходимо установить параметр max_replication_slots, задающий число источников репликации, которые могут отслеживаться. Он должен быть не меньше числа подписок, на которые будет подписываться данный подписчик. В max_logical_replication_workers необходимо установить значение не меньше числа подписок плюс некоторый резерв для синхронизации таблиц. Кроме того, может потребоваться изменить max_worker_processes, чтобы это число включало дополнительные рабочие процессы для репликации (как минимум max_logical_replication_workers + 1). При этом некоторые расширения и параллельные запросы также занимают слоты из числа max_worker_processes.

Необходимо установить параметры конфигурации в файле postgresql.conf:

```
wal_level = logical
```

Другие необходимые параметры по умолчанию имеют значения, подходящие для базовой настройки.

В файл pg_hba.conf необходимо внести изменения, чтобы разрешить репликацию (конкретные значения будут зависеть от фактической конфигурации сети и имени пользователя, с которым необходимо подключаться):

```
host all repuser 0.0.0.0/0 md5
```

Затем в базе данных публикации выполнить команду:

```
CREATE PUBLICATION mypub FOR TABLE users, departments;
```

В базе данных подписчика выполнить команду:

```
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=foo host=bar user=repuser' PUBLICATION mypub;
```

Показанная выше команда запустит процесс репликации, который вначале синхронизирует исходное содержимого таблиц users и departments, а затем начнёт перенос инкрементальных изменений в этих таблицах.

3.15. Порядок действий в случае сбоев, отказа сервера, при уничтожении или модификации программ и служебных данных СУБД PG360

3.15.2. При аппаратном отказе сервера необходимо выполнить следующие действия:

– заменить сервер;

– установить необходимое общесистемное программное обеспечение;

– установить СУБД PG360, руководствуясь документом "Руководство программиста";

– восстановить данные из резервной копии.

3.15.3. При уничтожении или модификации программ и служебных данных СУБД PG360 оператор - администратор СУБД PG360 должен выполнить следующие действия:

– установить СУБД PG360, руководствуясь документом "Руководство программиста";

– восстановить данные из резервной копии.

3.16. Квалификация оператора

3.16.1. Администратор сервера должен иметь навыки администрирования ОС Linux.

3.16.2. Администратор СУБД PG360 должен изучить документ "Система управления базами данных "PG360". Руководство оператора".

4. СООБЩЕНИЯ ОПЕРАТОРУ

4.1. Во время сеанса работы сервера PG360 могут выдаваться различные сообщения, которым назначены пятисимвольные коды ошибок, соответствующие кодам «SQLSTATE», описанным в стандарте SQL. Согласно стандарту, первые два символа кода ошибки обозначают класс ошибок, а последние три символа обозначают определённое условие в этом классе.

В Таблице 4.1 перечислены все коды ошибок и классы ошибок. Для каждого класса ошибок имеется код ошибки с последними тремя символами 000 – это ошибки, которые относятся к некоторому классу, но не имеют более определённого кода.

Символ, указанный в столбце «Имя условия», определяет условие в PL/pgSQL. Имена условий могут записываться в верхнем или нижнем регистре.

Таблица 4.1. Коды и классы ошибок PG360

Код ошибки	**Имя условия**
Класс 00 — Успешное завершение	
00000	successful_completion
Класс 01 — Предупреждение	
01000	warning
0100C	dynamic_result_sets_returned
01008	implicit_zero_bit_padding
01003	null_value_eliminated_in_set_function
01007	privilege_not_granted
01006	privilege_not_revoked
01004	string_data_right_truncation
01P01	deprecated_feature
Класс 02 — Нет данных (это также класс предупреждений согласно стандарту SQL)	
02000	no_data
02001	no_additional_dynamic_result_sets_returned
Класс 03 — SQL-оператор ещё не завершён	
03000	sql_statement_not_yet_complete
Класс 08 — Исключение, связанное с подключением	
08000	connection_exception
08003	connection_does_not_exist
08006	connection_failure
08001	sqlclient_unable_to_establish_sqlconnection
08004	sqlserver_rejected_establishment_of_sqlconnection
08007	transaction_resolution_unknown
08P01	protocol_violation
Класс 09 — Исключение с действием триггера	
09000	triggered_action_exception
Класс 0A — Неподдерживаемая функциональность	
0A000	feature_not_supported

Код ошибки	Имя условия
Класс 0B — Неверное начало транзакции	
0B000	invalid_transaction_initiation
Класс 0F — Исключение с указателем на данные	
0F000	locator_exception
0F001	invalid_locator_specification
Класс 0L — Неверный праводатель	
0L000	invalid_grantor
0LP01	invalid_grant_operation
Класс 0P — Неверное указание роли	
0P000	invalid_role_specification
Класс 0Z — Исключение диагностики	
0Z000	diagnostics_exception
0Z002	stacked_diagnostics_accessed_without_active_handler
Класс 20 — Case не найден	
20000	case_not_found
Класс 21 — Нарушение количества	
21000	cardinality_violation
Класс 22 — Исключение в данных	
22000	data_exception
2202E	array_subscript_error
22021	character_not_in_repertoire
22008	datetime_field_overflow
22012	division_by_zero
22005	error_in_assignment
2200B	escape_character_conflict
22022	indicator_overflow
22015	interval_field_overflow
2201E	invalid_argument_for_logarithm
22014	invalid_argument_for_ntile_function
22016	invalid_argument_for_nth_value_function
2201F	invalid_argument_for_power_function
2201G	invalid_argument_for_width_bucket_function
22018	invalid_character_value_for_cast
22007	invalid_datetime_format
22019	invalid_escape_character
2200D	invalid_escape_octet
22025	invalid_escape_sequence
22P06	nonstandard_use_of_escape_character
22010	invalid_indicator_parameter_value
22023	invalid_parameter_value
22013	invalid_preceding_or_following_size

Код ошибки	**Имя условия**
2201B	invalid_regular_expression
2201W	invalid_row_count_in_limit_clause
2201X	invalid_row_count_in_result_offset_clause
2202H	invalid_tablesample_argument
2202G	invalid_tablesample_repeat
22009	invalid_time_zone_displacement_value
2200C	invalid_use_of_escape_character
2200G	most_specific_type_mismatch
22004	null_value_not_allowed
22002	null_value_no_indicator_parameter
22003	numeric_value_out_of_range
2200H	sequence_generator_limit_exceeded
22026	string_data_length_mismatch
22001	string_data_right_truncation
22011	substring_error
22027	trim_error
22024	unterminated_c_string
2200F	zero_length_character_string
22P01	floating_point_exception
22P02	invalid_text_representation
22P03	invalid_binary_representation
22P04	bad_copy_file_format
22P05	untranslatable_character
2200L	not_an_xml_document
2200M	invalid_xml_document
2200N	invalid_xml_content
2200S	invalid_xml_comment
2200T	invalid_xml_processing_instruction
22030	duplicate_json_object_key_value
22031	invalid_argument_for_sql_json_datetime_function
22032	invalid_json_text
22033	invalid_sql_json_subscript
22034	more_than_one_sql_json_item
22035	no_sql_json_item
22036	non_numeric_sql_json_item
22037	non_unique_keys_in_a_json_object
22038	singleton_sql_json_item_required
22039	sql_json_array_not_found
2203A	sql_json_member_not_found
2203B	sql_json_number_not_found
2203C	sql_json_object_not_found
2203D	too_many_json_array_elements

Код ошибки	**Имя условия**
2203E	too_many_json_object_members
2203F	sql_json_scalar_required
Класс 23 — Нарушение ограничения целостности	
23000	integrity_constraint_violation
23001	restrict_violation
23502	not_null_violation
23503	foreign_key_violation
23505	unique_violation
23514	check_violation
23P01	exclusion_violation
Класс 24 — Неверное состояние курсора	
24000	invalid_cursor_state
Класс 25 — Неверное состояние транзакции	
25000	invalid_transaction_state
25001	active_sql_transaction
25002	branch_transaction_already_active
25008	held_cursor_requires_same_isolation_level
25003	inappropriate_access_mode_for_branch_transaction
25004	inappropriate_isolation_level_for_branch_transaction
25005	no_active_sql_transaction_for_branch_transaction
25006	read_only_sql_transaction
25007	schema_and_data_statement_mixing_not_supported
25P01	no_active_sql_transaction
25P02	in_failed_sql_transaction
25P03	idle_in_transaction_session_timeout
Класс 26 — Неверное имя SQL-оператора	
26000	invalid_sql_statement_name
Класс 27 — Нарушение при изменении данных в триггере	
27000	triggered_data_change_violation
Класс 28 — Неверное указание авторизации	
28000	invalid_authorization_specification
28P01	invalid_password
Класс 2B — Зависимые описания привилегий всё ещё существуют	
2B000	dependent_privilege_descriptors_still_exist
2BP01	dependent_objects_still_exist
Класс 2D — Неверное завершение транзакции	
2D000	invalid_transaction_termination
Класс 2F — Исключение в подпрограмме SQL	
2F000	sql_routine_exception
2F005	function_executed_no_return_statement
2F002	modifying_sql_data_not_permitted
2F003	prohibited_sql_statement_attempted

Код ошибки	Имя условия
2F004	reading_sql_data_not_permitted
Класс 34 — Неверное имя курсора	
34000	invalid_cursor_name
Класс 38 — Исключение во внешней подпрограмме	
38000	external_routine_exception
38001	containing_sql_not_permitted
38002	modifying_sql_data_not_permitted
38003	prohibited_sql_statement_attempted
38004	reading_sql_data_not_permitted
Класс 39 — Исключение при вызове внешней подпрограммы	
39000	external_routine_invocation_exception
39001	invalid_sqlstate_returned
39004	null_value_not_allowed
39P01	trigger_protocol_violated
39P02	srf_protocol_violated
39P03	event_trigger_protocol_violated
Класс 3B — Исключение точки сохранения	
3B000	savepoint_exception
3B001	invalid_savepoint_specification
Класс 3D — Неверное имя каталога	
3D000	invalid_catalog_name
Класс 3F — Неверное имя схемы	
3F000	invalid_schema_name
Класс 40 — Откат транзакции	
40000	transaction_rollback
40002	transaction_integrity_constraint_violation
40001	serialization_failure
40003	statement_completion_unknown
40P01	deadlock_detected
Класс 42 — Ошибка синтаксиса или нарушение правила доступа	
42000	syntax_error_or_access_rule_violation
42601	syntax_error
42501	insufficient_privilege
42846	cannot_coerce
42803	grouping_error
42P20	windowing_error
42P19	invalid_recursion
42830	invalid_foreign_key
42602	invalid_name
42622	name_too_long
42939	reserved_name
42804	datatype_mismatch

Код ошибки	Имя условия
42P18	indeterminate_datatype
42P21	collation_mismatch
42P22	indeterminate_collation
42809	wrong_object_type
428C9	generated_always
42703	undefined_column
42883	undefined_function
42P01	undefined_table
42P02	undefined_parameter
42704	undefined_object
42701	duplicate_column
42P03	duplicate_cursor
42P04	duplicate_database
42723	duplicate_function
42P05	duplicate_prepared_statement
42P06	duplicate_schema
42P07	duplicate_table
42712	duplicate_alias
42710	duplicate_object
42702	ambiguous_column
42725	ambiguous_function
42P08	ambiguous_parameter
42P09	ambiguous_alias
42P10	invalid_column_reference
42611	invalid_column_definition
42P11	invalid_cursor_definition
42P12	invalid_database_definition
42P13	invalid_function_definition
42P14	invalid_prepared_statement_definition
42P15	invalid_schema_definition
42P16	invalid_table_definition
42P17	invalid_object_definition
Класс 44 — Нарушение WITH CHECK OPTION	
44000	with_check_option_violation
Класс 53 — Нехватка ресурсов	
53000	insufficient_resources
53100	disk_full
53200	out_of_memory
53300	too_many_connections
53400	configuration_limit_exceeded
Класс 54 — Превышение ограничения программы	
54000	program_limit_exceeded

Код ошибки	Имя условия
54001	statement_too_complex
54011	too_many_columns
54023	too_many_arguments
Класс 55 — Объект не в требуемом состоянии	
55000	object_not_in_prerequisite_state
55006	object_in_use
55P02	cant_change_runtime_param
55P03	lock_not_available
55P04	unsafe_new_enum_value_usage
Класс 57 — Вмешательство оператора	
57000	operator_intervention
57014	query_canceled
57P01	admin_shutdown
57P02	crash_shutdown
57P03	cannot_connect_now
57P04	database_dropped
Класс 58 — Ошибка системы (ошибка, внешняя по отношению к PG360)	
58000	system_error
58030	io_error
58P01	undefined_file
58P02	duplicate_file
Класс 72 — Ошибка снимка	
72000	snapshot_too_old
Класс F0 — Ошибка файла конфигурации	
F0000	config_file_error
F0001	lock_file_exists
Класс HV — Ошибка обёртки сторонних данных (SQL/MED)	
HV000	fdw_error
HV005	fdw_column_name_not_found
HV002	fdw_dynamic_parameter_value_needed
HV010	fdw_function_sequence_error
HV021	fdw_inconsistent_descriptor_information
HV024	fdw_invalid_attribute_value
HV007	fdw_invalid_column_name
HV008	fdw_invalid_column_number
HV004	fdw_invalid_data_type
HV006	fdw_invalid_data_type_descriptors
HV091	fdw_invalid_descriptor_field_identifier
HV00B	fdw_invalid_handle
HV00C	fdw_invalid_option_index
HV00D	fdw_invalid_option_name

Код ошибки	**Имя условия**
HV090	fdw_invalid_string_length_or_buffer_length
HV00A	fdw_invalid_string_format
HV009	fdw_invalid_use_of_null_pointer
HV014	fdw_too_many_handles
HV001	fdw_out_of_memory
HV00P	fdw_no_schemas
HV00J	fdw_option_name_not_found
HV00K	fdw_reply_handle
HV00Q	fdw_schema_not_found
HV00R	fdw_table_not_found
HV00L	fdw_unable_to_create_execution
HV00M	fdw_unable_to_create_reply
HV00N	fdw_unable_to_establish_connection
Класс P0 — Ошибка PL/pgSQL	
P0000	plpgsql_error
P0001	raise_exception
P0002	no_data_found
P0003	too_many_rows
P0004	assert_failure
Класс XX — Внутренняя ошибка	
XX000	internal_error
XX001	data_corrupted
XX002	index_corrupted

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

В настоящем документе приняты следующие сокращения:

БД	– база данных
ОС	– операционная система
ПК	– персональный компьютер
СУБД	– система управления базами данных
УЦ	– удостоверяющий центр

Приложение А

Перечень представлений

A.1 pg_stat_activity

В представлении pg_stat_activity для каждого серверного процесса будет присутствовать по одной строке с информацией, относящейся к текущей деятельности этого процесса.

Таблица A.1. Представление pg_stat_activity

Тип столбца	**Описание**
datid oid	OID базы данных, к которой подключён этот серверный процесс
datname name	Имя базы данных, к которой подключён этот серверный процесс
pid integer	Идентификатор процесса этого серверного процесса
leader_pid integer	Идентификатор ведущего процесса группы, если текущий процесс является исполнителем параллельного запроса. NULL, если этот процесс является ве дущим или не задействован в параллельном запросе.
usesysid oid	OID пользователя, подключённого к этому серверному процессу
usename name	Имя пользователя, подключённого к этому серверному процессу
application_name text	Название приложения, подключённого к этому серверному процессу
client_addr inet	IP-адрес клиента, подключённого к этому серверному процессу. Значение null в этом поле означает, что клиент подключён через сокет Unix на сторо не сервера или что это внутренний процесс, например, автоочистка.
client_hostname text по IP и только при включённом режиме log_hostname.	Имя компьютера для подключённого клиента, получаемое в результате обратного поиска в DNS по client_addr . Это поле будет отлично от null только в случае соединений
client_port integer	Номер TCP-порта, который используется клиентом для соединения с этим обслуживающим процессом, или -1, если используется сокет Unix. Если поле содержит NULL, это означает, что это внутренний серверный процесс.
backend_start timestamp with time zone	Время запуска процесса. Для процессов, обслуживающих клиентов, это вре мя подключения клиента к серверу.
xact_start timestamp with time zone	Время начала текущей транзакции в этом процессе или null при отсутствии активной транзакции. Если текущий запрос был первым в своей транзакции, то значение в этом столбце совпадает со значением столбца query_start .
query_start timestamp with time zone	Время начала выполнения активного в данный момент запроса, или, если state не active, то время начала выполнения последнего запроса
state_change timestamp with time zone	Время последнего изменения состояния (поля state)
wait_event_type text	Тип события, которого ждёт обслуживающий процесс, если это ожидание имеет место; в противном случае — NULL.
wait_event text	Имя ожидаемого события, если обслуживающий процесс находится в состо янии ожидания, а в противном случае — NULL.
state text	Общее текущее состояние этого серверного процесса. Возможные значения: active: серверный процесс выполняет запрос. idle: серверный процесс ожидает новой команды от клиента. idle in transaction: серверный процесс находится внутри транзакции, но в на стоящее время не выполняет никакой запрос. idle in transaction (aborted) : Это состояние подобно idle in transaction, за ис ключением того, что один из операторов в транзакции вызывал ошибку. fastpath function call: серверный процесс выполняет fast-path функцию. disabled: Это состояние отображается для серверных процессов, у которых

Тип столбца	Описание
	параметр track_activities отключён.
backend_xid xid	Идентификатор верхнего уровня транзакции этого серверного процесса или любой другой.
backend_xmin xid	текущая граница xmin для серверного процесса.
query text	Текст последнего запроса этого серверного процесса. Если state имеет зна чение active, то в этом поле отображается запрос, который выполняется в настоящий момент. Если процесс находится в любом другом состоянии, то в этом поле отображается последний выполненный запрос. По умолчанию текст запроса обрезается до 1024 байт; это число определяется параметром track_activity_query_size.
backend_type text	Тип текущего серверного процесса. Возможные варианты: autovacuum launcher, autovacuum worker, logical replication launcher, logical replication worker, parallel worker, background writer, client backend, checkpointer, startup, walreceiver, walsender и walwriter. Кроме того, фоновые рабочие процессы, регистрируемые расширениями, могут иметь дополнительные типы.

Таблица А.1.1. Типы событий ожидания

Тип события ожидания	Описание
Activity	Серверный процесс простаивает. Это состояние показывает, что процесс ожидает активности в основном цикле обработки. В wait_event обозначается конкретное место ожида ния.
BufferPin	Серверный процесс ожидает исключительного доступа к буферу данных. Ожидание закрепления буфера может растягиваться, если другой процесс удерживает откры тый курсор, который до этого читал данные из целевого буфера.
Client	Серверный процесс ожидает в сокете некоторую активность пользовательского приложения. То есть сервер ждёт, что произойдёт какое-то событие, не зависящее от его внутренних процессов. В wait_event обозначается конкретное место ожидания.
Extension	Серверный процесс ожидает условия, возникающего в модуле расширения.
IO	Серверный процесс ожидает завершения операции ввода/вывода. В wait_event обозначается конкретное место ожидания.
IPC	Серверный процесс ожидает взаимодействия с другим процессом. В wait_event обозначается конкретное место ожидания.
Lock	Серверный процесс ожидает тяжёлую блокировку. Тяжёлые блокировки, также на зываемые блокировками менеджера блокировок или просто блокировками, в основ ном защищают объекты уровня SQL, такие как таблицы. Однако они также приме няются для взаимоисключающего выполнения некоторых внутренних операций, например, расширения отношений. Тип ожидаемой блокировки показывается в wait_ event.
LWLock	Серверный процесс ожидает лёгкую блокировку. В большинстве своём такие бло кировки защищают определённые структуры данных в общей памяти. В wait_ event будет содержаться имя, отражающее цель получения лёгкой блокировки.
Timeout	Серверный процесс ожидает истечения определённого времени. В wait_event обозначается конкретное место ожидания.

Таблица А.1.2. События ожидания, относящиеся к типу Activity

Событие ожидания Activity	**Описание**
ArchiverMain	Ожидание в основном цикле процесса архиватора.
AutoVacuumMain	Ожидание в основном цикле процесса запуска автоочистки.
BgWriterHibernate	Ожидание в фоновом процессе записи, переход в режим «заморозки».

Событие ожидания Activity	**Описание**
BgWriterMain	Ожидание в основном цикле процесса фоновой записи.
CheckpointerMain	Ожидание в основном цикле процесса контрольной точки.
LogicalApplyMain	Ожидание в основном цикле процесса применения логической ре пликации.
LogicalLauncherMain	Ожидание в основном цикле процесса запуска обработчиков логиче ской репликации.
PgStatMain	Ожидание в основном цикле процесса сборщика статистики.
RecoveryWalStream	Ожидание поступления записей WAL в основном цикле стартового процесса во время восстановления.
SysLoggerMain	Ожидание в основном цикле процесса системного журнала (syslogger).
WalReceiverMain	Ожидание в основном цикле процесса-приёмника WAL.
WalSenderMain	Ожидание в основном цикле процесса-передатчика WAL.
WalWriterMain	Ожидание в основном цикле процесса, пишущего WAL.

Таблица A.1.3. События ожидания, относящиеся к типу BufferPin

Событие ожидания BufferPin	**Описание**
BufferPin	Ожидание получения исключительного закрепления буфера.

Таблица A.1.4. События ожидания, относящиеся к типу Client

События ожидания Client	**Описание**
ClientRead	Ожидание при чтении данных, получаемых от клиента.
ClientWrite	Ожидание при записи данных, передаваемых клиенту.
LibPQWalReceiverConnect	Ожидание в приёмнике WAL установления подключения к удалён ному серверу.
LibPQWalReceiverReceive	Ожидание в приёмнике WAL поступления данных от удалённого сервера.
SSLOpenServer	Ожидание SSL при попытке установления соединения.
WalReceiverWaitStart	Ожидание от стартового процесса передачи начальных данных для потоковой репликации.
WalSenderWaitForWAL	Ожидание сброса WAL в процессе-передатчике WAL.
WalSenderWriteData	Ожидание какой-либо активности при обработке ответов от WAL-приёмника в процессе-передатчике WAL.

Таблица A.1.5. События ожидания, относящиеся к типу Extension

Событие ожидания Extension	**Описание**
Extension	Ожидание в расширении.

Таблица A.1.6. События ожидания, относящиеся к типу IO

Событие ожидания IO	**Описание**
BufFileRead	Ожидание чтения из буферизованного файла.

Событие ожидания IO	**Описание**
BufFileWrite	Ожидание записи в буферизованный файл.
ControlFileRead	Ожидание чтения из файла pg_control .
ControlFileSync	Ожидание помещения файла pg_control в надёжное хранилище.
ControlFileSyncUpdate	Ожидание переноса изменений файла pg_ control в надёжное храни лище.
ControlFileWrite	Ожидание записи в файл pg_control .
ControlFileWriteUpdate	Ожидание записи для изменения файла pg_ control .
CopyFileRead	Ожидание чтения во время операции копирования файла.
CopyFileWrite	Ожидание записи во время операции копирования файла.
DSMFillZeroWrite	Ожидание заполнения нулями файла, применяемого для поддержки динамической общей памяти.
DataFileExtend	Ожидание расширения файла данных отношения.
DataFileFlush	Ожидание помещения файла данных отношения в надёжное храни лище.
DataFileImmediateSync	Ожидание немедленной синхронизации файла данных отношения с надёжным хранилищем.
DataFilePrefetch	Ожидание асинхронной предвыборки из файла данных отношения.
DataFileRead	Ожидание чтения из файла данных отношения.
DataFileSync	Ожидание переноса изменений в файле данных отношения в надёж ное хранилище.
DataFileTruncate	Ожидание усечения файла данных отношения.
DataFileWrite	Ожидание записи в файл данных отношения.
LockFileAddToDataDirRead	Ожидание чтения при добавлении строки в файл блокировки катало га данных.
LockFileAddToDataDirSync	Ожидание помещения данных в надёжное хранилище при добавлении строки в файл блокировки каталога данных.
LockFileAddToDataDirWrite	Ожидание записи при добавлении строки в файл блокировки катало га данных.
LockFileCreateRead	Ожидание чтения при создании файла блокировки каталога данных.
LockFileCreateSync	Ожидание помещения данных в надёжное хранилище при создании файла блокировки каталога данных.
LockFileCreateWrite	Ожидание записи при создании файла блокировки каталога данных.
LockFileReCheckDataDirRead	Ожидание чтения во время перепроверки файла блокировки каталога данных.
LogicalRewriteCheckpointSyn c	Ожидание помещения отображений логической перезаписи в надёж ное хранилище во время контрольной точки.
LogicalRewriteMappingSync	Ожидание помещения данных отображений в надёжное хранилище в процессе логической перезаписи.
LogicalRewriteMappingWrite	Ожидание записи данных отображений в процессе логической переза писи.

Событие ожидания IO	**Описание**
LogicalRewriteSync	Ожидание помещения отображений логической перезаписи в надёж ное хранилище.
LogicalRewriteTruncate	Ожидание усечения файла отображений в процессе логической пе резаписи.
LogicalRewriteWrite	Ожидание сохранения отображений логической перезаписи.
RelationMapRead	Ожидание чтения файла отображений отношений.
RelationMapSync	Ожидание помещения файла отображений отношений в надёжное хранилище.
RelationMapWrite	Ожидание записи в файл отображений отношений.
ReorderBufferRead	Ожидание чтения при работе с буфером переупорядочивания.
ReorderBufferWrite	Ожидание записи при работе с буфером переупорядочивания.
ReorderLogicalMappingRead	Ожидание чтения логического отображения при работе с буфером переупорядочивания.
ReplicationSlotRead	Ожидание чтения из управляющего файла слота репликации.
ReplicationSlotRestoreSync	Ожидание помещения в надёжное хранилище управляющего файла слота репликации при восстановлении его в памяти.
ReplicationSlotSync	Ожидание помещения в надёжное хранилище управляющего файла слота репликации.
ReplicationSlotWrite	Ожидание записи в управляющий файл слота репликации.
SLRUFlushSync	Ожидание помещения данных SLRU в надёжное хранилище во вре мя контрольной точки или отключения базы данных.
SLRURead	Ожидание чтения страницы SLRU.
SLRUSync	Ожидание помещения данных SLRU в надёжное хранилище после записи страницы.
SLRUWrite	Ожидание записи страницы SLRU.
SnapbuildRead	Ожидание чтения сериализованного исторического снимка каталога БД.
SnapbuildSync	Ожидание помещения сериализованного исторического снимка ката лога БД в надёжное хранилище.
SnapbuildWrite	Ожидание записи сериализованного исторического снимка каталога БД.
TimelineHistoryFileSync	Ожидание помещения в надёжное хранилище файла истории линии времени, полученного через потоковую репликацию.
TimelineHistoryFileWrite	Ожидание записи файла истории линии времени, полученного через потоковую репликацию.
TimelineHistoryRead	Ожидание чтения файла истории линии времени.
TimelineHistorySync	Ожидание помещения в надёжное хранилище только что созданного файла истории линии времени.
TimelineHistoryWrite	Ожидание записи только что созданного файла истории линии време ни.
TwophaseFileRead	Ожидание чтения файла двухфазного состояния.
TwophaseFileSync	Ожидание помещения файла двухфазного состояния в надёжное хра нилище.

Событие ожидания IO	**Описание**
TwophaseFileWrite	Ожидание записи файла двухфазного состояния.
WALBootstrapSync	Ожидание помещения WAL в надёжное хранилище в процессе на чальной загрузки.
WALBootstrapWrite	Ожидание записи страницы WAL в процессе начальной загрузки.
WALCopyRead	Ожидание чтения при создании нового сегмента WAL путём копиро вания существующего.
WALCopySync	Ожидание помещения в надёжное хранилище нового сегмента WAL, созданного путём копирования существующего.
WALCopyWrite	Ожидание записи при создании нового сегмента WAL путём копиро вания существующего.
WALInitSync	Ожидание помещения в надёжное хранилище нового инициализиро ванного файла WAL.
WALInitWrite	Ожидание записи при инициализации нового файла WAL.
WALRead	Ожидание чтения из файла WAL.
WALSenderTimelineHistoryR ead	Ожидание чтения из файла истории линии времени при обработки процессом walsender команды timeline.
WALSync	Ожидание помещения файла WAL в надёжное хранилище.
WALSyncMethodAssign	Ожидание помещения данных в надёжное хранилище при смене ме тода синхронизации WAL.
WALWrite	Ожидание записи в файл WAL.

Таблица A.1.7. События ожидания, относящиеся к типу IPC

События ожидания IPC	**Описание**
BackupWaitWalArchive	Ожидание файлов WAL, необходимых для успешного завершения архивации.
BgWorkerShutdown	Ожидание завершения фонового рабочего процесса.
BgWorkerStartup	Ожидание запуска фонового рабочего процесса.
BtreePage	Ожидание доступности номера страницы, необходимого для про должения параллельного сканирования B-дерева.
CheckpointDone	Ожидание завершения контрольной точки.
CheckpointStart	Ожидание начала контрольной точки.
ExecuteGather	Ожидание активности дочернего процесса при выполнении узла плана Gather.
HashBatchAllocate	Ожидание выделения хеш-таблицы выбранным участником парал лельного хеширования.
HashBatchElect	Ожидание выделения хеш-таблицы выбранным участником парал лельного хеширования.
HashBatchLoad	Ожидание завершения загрузки хеш-таблицы другими участниками параллельного хеширования.
HashBuildAllocate	Ожидание выделения начальной хеш-таблицы выбранным участни ком параллельного хеширования.
HashBuildElect	Ожидание в процессе выбора участника параллельного хеширова ния для выделения начальной хеш-таблицы.

События ожидания IPC	**Описание**
HashBuildHashInner	Ожидание завершения хеширования внутреннего отношения другими участниками параллельного хеширования.
HashBuildHashOuter	Ожидание завершения хеширования внешнего отношения другими участниками параллельного хеширования.
HashGrowBatchesAllocate	Ожидание выделения дополнительных пакетов выбранным участником параллельного хеширования.
HashGrowBatchesDecide	Ожидание в процессе выбора участника параллельного хеширования для принятия решения о предстоящем добавлении пакетов.
HashGrowBatchesElect	Ожидание в процессе выбора участника параллельного хеширования для выделения дополнительных пакетов.
HashGrowBatchesFinish	Ожидание решения о предстоящем добавлении пакетов участником параллельного хеширования.
HashGrowBatchesRepartition	Ожидание завершения переразбиения другими участниками параллельного хеширования.
HashGrowBucketsAllocate	Ожидание завершения выделения дополнительных групп выбранным участником параллельного хеширования.
HashGrowBucketsElect	Ожидание в процессе выбора участника параллельного хеширования для выделения дополнительных групп.
HashGrowBucketsReinsert	Ожидание завершения добавления кортежей в новые группы другими участниками параллельного хеширования.
LogicalSyncData	Ожидание от удалённого сервера, осуществляющего логическую репликацию, передачи данных для начальной синхронизации таблиц.
LogicalSyncStateChange	Ожидание изменения состояния удалённого сервера, осуществляющего логическую репликацию.
MessageQueueInternal	Ожидание подключения другого процесса к общей очереди сообщений.
MessageQueuePutMessage	Ожидание записи сообщения протокола в общую очередь сообщений.
MessageQueueReceive	Ожидание получения байтов из общей очереди сообщений.
MessageQueueSend	Ожидание передачи байтов в общую очередь сообщений.
ParallelBitmapScan	Ожидание инициализации параллельного сканирования по битовой карте.
ParallelCreateIndexScan	Ожидание завершения сканирования кучи параллельными исполнителями CREATE INDEX.
ParallelFinish	Ожидание завершения вычислений параллельными рабочими процессами.
ProcArrayGroupUpdate	Ожидание обнуления ID транзакции, которое должен произвести ведущий процесс группы по окончании параллельной операции.
ProcSignalBarrier	Ожидание обработки события барьера всеми обслуживающими процессами.
Promote	Ожидание повышения ведомого.
RecoveryConflictSnapshot	Ожидание разрешения конфликтов восстановления для осуществления очистки.

События ожидания IPC	**Описание**
RecoveryConflictTablespace	Ожидание разрешения конфликтов восстановления для удаления та бличного пространства.
RecoveryPause	Ожидание возобновления восстановления.
ReplicationOriginDrop	Ожидание перехода источника репликации в неактивное состояние, что позволит затем удалить его.
ReplicationSlotDrop	Ожидание перехода слота репликации в неактивное состояние, что позволит затем удалить его.
SafeSnapshot	Ожидание получения безопасного снимка для транзакции READ ONLY DEFERRABLE.
SyncRep	Ожидание подтверждения от удалённого сервера при синхронной репликации.
XactGroupUpdate	Ожидание изменения состояния транзакции, которое должен произ вести ведущий процесс группы по окончании параллельной опера ции.

Таблица A.1.8. События ожидания, относящиеся к типу Lock

Событие ожидания Lock	**Описание**
advisory	Ожидание при запросе рекомендательной пользовательской блоки ровки.
extend	Ожидание при расширении отношения.
frozenid	Ожидание изменения pg_ database .datfrozenxid и pg_ database .datminmxid.
object	Ожидание при запросе блокировки для нереляционного объекта БД.
page	Ожидание при запросе блокировки для страницы отношения.
relation	Ожидание при запросе блокировки для отношения.
spectoken	Ожидание при запросе блокировки спекулятивного добавления.
transactionid	Ожидание завершения транзакции.
tuple	Ожидание при запросе блокировки для кортежа.
userlock	Ожидание при запросе пользовательской блокировки.
virtualxid	Ожидание при запросе блокировки виртуального ID транзакции.

Таблица A.1.9. События ожидания, относящиеся к типу LWLock

Событие ожидания LWLock	**Описание**
AddinShmemInit	Ожидание при распределении блоков общей памяти для расширений.
AutoFile	Ожидание при изменении файла postgresql.auto.conf.
Autovacuum	Ожидание при чтении или изменении текущего состояния рабочих процес сов автоочистки.
AutovacuumSchedule	Ожидание при подтверждении, что таблица, выбранная для автоочистки, всё ещё нуждается в очистке.

Событие ожидания LWLock	**Описание**
BackgroundWorker	Ожидание при чтении или изменении состояния фонового рабочего про цесса.
BtreeVacuum	Ожидание при чтении или изменении информации, связанной с очисткой, для индекса-B-дерева.
BufferContent	Ожидание при обращении к странице данных в памяти.
BufferIO	Ожидание при вводе/выводе, связанном со страницей данных.
BufferMapping	Ожидание при связывании блока данных с буфером в пуле буферов.
Checkpoint	Ожидание начала контрольной точки.
CheckpointerComm	Ожидание при управлении запросами fsync.
CommitTs	Ожидание при чтении или изменении последнего значения, заданного в ка честве времени фиксирования транзакции.
CommitTsBuffer	Ожидание ввода/вывода с SLRU-буфером данных о времени фиксирования транзакций.
CommitTsSLRU	Ожидание при обращении к SLRU-кешу данных о времени фиксирования транзакций.
ControlFile	Ожидание при чтении или изменении файла pg_control либо при создании нового файла WAL.
DynamicSharedMemoryCo ntrol	Ожидание при чтении или изменении информации о выделении динамиче ской общей памяти.
LockFastPath	Ожидание при чтении или изменении информации процесса о блокировках по быстрому пути.
LockManager	Ожидание при чтении или изменении информации о «тяжёлых» блокиров ках.
LogicalRepWorker	Ожидание при чтении или изменении состояния рабочих процессов логи ческой репликации.
MultiXactGen	Ожидание при чтении или изменении общего состояния мультитранзак ций.
MultiXactMemberBuffer	Ожидание ввода/вывода с SLRU-буфером данных о членах мультитранзак ций.
MultiXactMemberSLRU	Ожидание при обращении к SLRU-кешу данных о членах мультитранзак ций.
MultiXactOffsetBuffer	Ожидание ввода/вывода с SLRU-буфером данных о смещениях мультит ранзакций.
MultiXactOffsetSLRU	Ожидание при обращении к SLRU-кешу данных о смещениях мультитран закций.
MultiXactTruncation	Ожидание при чтении или очистке информации мультитранзакций.
NotifyBuffer	Ожидание ввода/вывода с SLRU-буфером сообщений NOTIFY.
NotifyQueue	Ожидание при чтении или изменении сообщений NOTIFY.
NotifyQueueTail	Ожидание при изменении границы массива сообщений NOTIFY.
NotifySLRU	Ожидание при обращении к SLRU-кешу сообщений NOTIFY.
OidGen	Ожидание при выделении нового OID.

Событие ожидания LWLock	**Описание**
OldSnapshotTimeMap	Ожидание при чтении или изменении информации о старом снимке.
ParallelAppend	Ожидание выбора следующего подплана в процессе выполнения узла па раллельного добавления (Parallel Append).
ParallelHashJoin	Ожидание синхронизации рабочих процессов в процессе выполнения узла плана Parallel Hash Join.
ParallelQueryDSA	Ожидание выделения динамической общей памяти для параллельного запроса.
PerSessionDSA	Ожидание выделения динамической общей памяти для параллельного запроса.
PerSessionRecordType	Ожидание при обращении к информации параллельного запроса о состав ных типах.
PerSessionRecordTypmod	Ожидание при обращении к информации параллельного запроса о модифи каторах типов, относящихся к типам анонимных записей.
PerXactPredicateList	Ожидание при обращении к списку предикатных блокировок, удерживае мых текущей сериализуемой транзакцией, во время параллельного запроса.
PredicateLockManager	Ожидание при обращении к информации о предикатных блокировках, ис пользуемой сериализуемыми транзакциями.
ProcArray	Ожидание при обращении к общим структурам данных в рамках процесса
RelationMapping	Ожидание при чтении или изменении файла pg_filenode.map , в котором отслеживаются назначения файловых узлов для некоторых системных ка талогов.
RelCacheInit	Ожидание при чтении или изменении файла инициализации кеша отноше ния (pg_ internal.init).
ReplicationOrigin	Ожидание при создании, удалении или использовании источника реплика ции.
ReplicationOriginState	Ожидание при чтении или изменении состояния одного источника репли кации.
ReplicationSlotAllocation	Ожидание при выделении или освобождении слота репликации.
ReplicationSlotControl	Ожидание при чтении или изменении состояния слота репликации.
ReplicationSlotIO	Ожидание при вводе/выводе со слотом репликации.
SerialBuffer	Ожидание ввода/вывода с SLRU-буфером данных о конфликтах сериализу емых транзакций.
SerializableFinishedList	Ожидание при обращении к списку завершённых сериализуемых транзак ций.
SerializablePredicateList	Ожидание при обращении к списку предикатных блокировок, удерживае мых сериализуемыми транзакциями.
SerializableXactHash	Ожидание при чтении или изменении информации о сериализуемых тран закциях.
SerialSLRU	Ожидание при обращении к SLRU-кешу данных о конфликтах сериализуе мых транзакций.

Событие ожидания LWLock	**Описание**
SharedTidBitmap	Ожидание при обращении к разделяемой битовой карте TID в процессе па раллельного сканирования индекса по битовой карте.
SharedTupleStore	Ожидание при обращении к разделяемому хранилищу кортежей во время параллельного запроса.
ShmemIndex	Ожидание при поиске или выделении области в разделяемой памяти.
SInvalRead	Ожидание при получении сообщений из общей очереди сообщений анну лирования.
SInvalWrite	Ожидание при добавлении в общую очередь сообщения аннулирования ка талога.
SubtransBuffer	Ожидание ввода/вывода с SLRU-буфером данных о подтранзакциях.
SubtransSLRU	Ожидание при обращении к SLRU-кешу данных подтранзакций.
SyncRep	Ожидание при чтении или изменении информации о состоянии синхрон ной репликации.
SyncScan	Ожидание при выборе начального положения для синхронизированного сканирования таблицы.
TablespaceCreate	Ожидание при создании или удалении табличного пространства.
TwoPhaseState	Ожидание при чтении или изменении состояния подготовленных транзак ций.
WALBufMapping	Ожидание при замене страницы в буферах WAL.
WALInsert	Ожидание при добавлении записей WAL в буфер в памяти.
WALWrite	Ожидание при записи буферов WAL на диск.
WrapLimitsVacuum	Ожидание при изменении лимитов идентификаторов транзакций и мультитранзакций.
XactBuffer	Ожидание ввода/вывода с SLRU-буфером данных о состоянии транзакций.
XactSLRU	Ожидание при обращении к SLRU-кешу данных о состоянии транзакций.
XactTruncation	Ожидание выполнения функции pg_xact_ status или обновления самого старого видимого в ней идентификатора транзакции.
XidGen	Ожидание при выделении нового идентификатора транзакции.

Таблица A.1.10. События ожидания, относящиеся к типу Timeout

Событие ожидания Timeout	Описание
BaseBackupThrottle	Ожидание в процессе базового резервного копирования из-за ограни чения активности.
PgSleep	Ожидание в результате вызова pg_sleep или родственной функции.
RecoveryApplyDelay	Ожидание применения WAL при восстановлении, вызванное уста новленной задержкой.

RecoveryRetrieveRetryInterval	Ожидание в процессе восстановления, когда нужные сегменты WAL нельзя получить из какого-либо источника (каталога pg_wal , архива или потока).
RegisterSyncRequest	Ожидание при передаче запросов синхронизации процессу контроль ной точки из-за переполнения очереди запросов.
VacuumDelay	Ожидание, вызванное задержкой очистки по критерию стоимости.

A.2. pg_stat_replication

Представление pg_stat_replication для каждого процесса-передатчика WAL будет содержать по одной строке со статистикой о репликации на ведомый сервер, к которому подключён этот процесс. В представлении перечисляются только ведомые серверы, подключённые напрямую.

Таблица A.2. Представление pg_stat_replication

Тип столбца	**Описание**
pid integer	Идентификатор процесса-передатчика WAL
usesysid oid	OID пользователя, подключённого к этому процессу-передатчику WAL
usename name	Имя пользователя, подключённого к этому процессу-передатчику WAL
application_name text	Имя приложения, которое подключено к этому процессу-передатчику WAL
client_addr inet	IP-адрес клиента, подключённого к этому процессу-передатчику WAL. Значение null в этом поле говорит о том, что клиент подсо единён через сокет Unix на сервере.
client_hostname text	Имя компьютера для подключённого клиента, получаемое в результа те обратного поиска в DNS по client_addr . Это поле будет отлично от null только в случае соединений по IP и только при включённом режи ме log_hostname.
client_port integer	Номер TCP-порта, который используется клиентом для взаимодей ствия с процессом-передатчиком WAL, или -1, если используется со кет Unix
backend_start timestamp with time zone	Время запуска процесса, т. е. время подключения клиента к этому процессу- передатчику WAL
backend_xmin xid	Значение xmin, полученное от ведомого сервера при включённом hot_standby_feedback.
state text	Текущее состояние процесса-передатчика WAL. Возможные значе ния: startup: Передатчик WAL запускается. catchup: Подключённый к этому передатчику WAL ведомый сервер догоняет ведущий. streaming: Передатчик WAL транслирует изменения после того, как подключённый к нему ведомый сервер нагнал ведущий. backup: Передатчик WAL передаёт резервную копию. stopping: Передатчик WAL останавливается.
sent_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, переданная через это соединение
write_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, записанная на диск этим ведомым сервером
flush_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, сброшенная на диск этим ведомым сервером

Тип столбца	**Описание**
replay_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, воспроизведён ная в базе данных этим ведомым сервером
write_lag interval	Время, прошедшее с момента локального сброса последних данных WAL до получения уведомления о том, что этот ведомый сервер запи сал их (но ещё не сбросил на диск и не применил). Это позволяет оце нить задержку, возникающую при фиксации транзакции, когда в synchronous_commit выбран уровень remote_write , если данный сер вер будет настроен как синхронный ведомый.
flush_lag interval	Время, прошедшее с момента локального сброса последних данных WAL до получения уведомления о том, что этот ведомый сервер запи сал и сбросил их на диск (но ещё не применил). Это позволяет оце нить задержку, возникающую при фиксации транзакции, когда в synchronous_commit выбран уровень on, если данный сервер будет на строен как синхронный ведомый.
replay_lag interval	Время, прошедшее с момента локального сброса последних данных WAL до получения уведомления о том, что этот ведомый сервер запи сал, сбросил на диск и применил их. Это позволяет оценить задержку, возникающую при фиксации транзакции, когда в synchronous_commit выбран уровень remote_apply , если данный сервер будет настроен как синхронный ве домый.
sync_priority integer	Приоритет этого ведомого сервера для выбора в качестве синхронного ведомого при синхронной репликации с учётом приоритетов. При синхронной репликации с учётом кворума не имеет значения.
sync_state text	Состояние синхронизации этого ведомого сервера. Возможные значе ния: async: Этот ведомый сервер является асинхронным. potential: Этот ведомый сервер сейчас является асинхронным, но мо жет стать синхронным в случае отказа одного из текущих синхронных серверов. sync: Этот ведомый сервер является синхронным. quorum: Этот ведомый сервер считается кандитатом на участие в кво руме.
reply_time timestamp with time zone	Время отправки последнего ответного сообщения, полученного от ве домого сервера

Длительность задержек, показываемая в представлении pg_stat_replication, включает время, которое потребовалось для того, чтобы записать, сбросить на диск и воспроизвести последние записи WAL и для того, чтобы передатчик WAL узнал об этом. Для асинхронного ведомого в столбце replay_lag показывается примерная задержка перед тем, как последние транзакции становятся видны для запросов. Если ведомый сервер нагоняет передающий и в WAL отсутствует активность, последние длительности задержек будут отображаться ещё некоторое время, а затем сменятся на NULL.

Длительность задержек автоматически определяется при физической репликации. Модули логического декодирования могут не выдавать необходимые контрольные сообщения; в их отсутствие механизм отслеживания просто выводит задержку NULL.

A.3. pg_stat_wal_receiver

Представление pg_stat_wal_receiver будет иметь только одну строку со статистикой приёмника WAL от сервера, на котором работает приёмник.

Таблица А.3. Представление pg_stat_wal_receiver

Тип столбца	**Описание**
pid integer	Идентификатор процесса WAL-приёмника
status text	Состояние активности процесса WAL-приёмника
receive_start_lsn pg_lsn	Первая позиция в журнале упреждающей записи в момент запуска приёмни ка WAL
receive_start_tli integer	Первый номер линии времени в момент запуска приёмника WAL
written_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, уже полученная и пере данная для записи на диск, но ещё не сброшенная. Эту позицию не следует ис пользовать при проверках целостности данных.
flushed_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, уже полученная и сброшенная на диск; начальным значением этого поля будет первая позиция в журнале в момент запуска приёмника WAL
received_tli integer	Номер линии времени последней позиции в журнале упреждающей записи, уже полученной и сброшенной на диск; начальным значением этого поля будет номер линии времени первой позиции в момент запуска приёмника WAL
last_msg_send_time timestamp with time zone	Время отправки последнего сообщения, полученного от изначального пере датчика WAL
last_msg_receipt_time timestamp with time zone	Время поступления последнего сообщения, полученного от изначального передатчика WAL
latest_end_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, сообщённая изначаль ному передатчику WAL
latest_end_time timestamp with time zone	Время последней позиции в журнале упреждающей записи, сообщённой из начальному передатчику WAL
slot_name text	Имя слота репликации, используемого этим приёмником WAL
sender_host text	Узел, где работает сервер PG360, обслуживающий данный приёмник WAL. Может задаваться именем или IP-адресом компьютера либо путём ка талога (если подключение установлено через сокет Unix). (Подключение к сокету легко распознать, потому что путь всегда абсолютный и начинается с /.)
sender_port integer	Номер порта сервера PG360, к которому подключён этот приёмник WAL.
conninfo text	Строка подключения, используемая этим приёмником WAL (секретные поля в ней скрыты).

A.4. pg_stat_subscription

Представление pg_stat_subscription содержит по одной строке для подписки для основного рабочего процесса (с NULL в PID, если процесс не работает) и дополнительные строки для рабочих процессов, осуществляющих копирование начальных данных для таблиц в подписке.

Таблица А.4. Представление pg_stat_subscription

Тип столбца	**Описание**
subid oid	OID подписки

Тип столбца	Описание
subname name	Имя подписки
pid integer	Идентификатор рабочего процесса, обслуживающего подписку
relid oid	OID отношения, которое синхронизирует рабочий процесс сейчас; null для основного процесса применения изменений
received_lsn pg_lsn	Последняя позиция в журнале упреждающей записи (начальное значение этого поля 0)
last_msg_send_time timestamp with time zone	Время отправки последнего сообщения, полученного от изначального передатчика WAL
last_msg_receipt_time timestamp with time zone	Время поступления последнего сообщения, полученного от изначального передатчика WAL
latest_end_lsn pg_lsn	Последняя позиция в журнале упреждающей записи, сообщённая изна чальному передатчику WAL
latest_end_time timestamp with time zone	Время последней позиции в журнале упреждающей записи, сообщённой изначальному передатчику WAL

A.5. pg_stat_ssl

Представление pg_stat_ssl содержит по одной строке для каждого обслуживающего процесса и процесса, передающего WAL, и показывает статистику использования SSL для подключений. Его можно соединить с pg_stat_activity или pg_stat_replication по столбцу pid и получить дополнительные сведения о подключении.

Таблица A.5. Представление pg_stat_ssl

Тип столбца	Описание
pid integer	Идентификатор обслуживающего процесса или процесса, передающего WAL
ssl boolean	True, если для этого подключения используется SSL
version text	Версия SSL либо NULL, если SSL для этого подключения не используется
cipher text	Имя применяемого шифра SSL либо NULL, если SSL для этого подключения не используется
bits integer	Число бит в применяемом алгоритме шифрования либо NULL, если SSL для это го подключения не используется
compression boolean	True, если применяется сжатие SSL, false в противном случае, либо NULL, если SSL для этого подключения не используется
client_dn text	Поле DN (Distinguished Name, Уникальное имя) из используемого клиентского сертификата либо NULL, если клиент не передал сертификат или SSL для этого подключения не используется. Это значение усекается, если оказывается длиннее NAMEDATALEN символов (64 символа в стандартной сборке).

Тип столбца	Описание
client_serial numeric	Серийный номер клиентского сертификата либо NULL, если клиент не передал сертификат или SSL для этого подключения не используется. В сочетании с именем удостоверяющего центра, выдавшего сертификат, серийный номер одно значно идентифицирует сертификат (если только УЦ не выдаёт по ошибке оди наковые серийные номера).
issuer_dn text	Уникальное имя (Distinguished Name, DN) удостоверяющего центра, выдавшего клиентский сертификат, либо NULL, если клиент не передал сертификат или SSL для этого подключения не используется. Это значение усекается подобно client_dn .

A.6. pg_stat_archiver

В представлении pg_stat_archiver всегда будет одна строка, содержащая данные о текущем состоянии процесса архивации в кластере.

Таблица A.6. Представление pg_stat_archiver

Тип столбца	Описание
archived_count bigint	Число файлов WAL, которые были успешно архивированы
last_archived_wal text	Имя последнего файла WAL успешно архивированного
last_archived_time timestamp with time zone	Время последней успешной архивации
failed_count bigint	Число неудачных попыток архивации файлов WAL
last_failed_wal text	Имя файла WAL последней неудавшейся архивации
last_failed_time timestamp with time zone	Время последней неудавшейся архивации
stats_reset timestamp with time zone	Последнее время сброса этих статистических данных

A.7. pg_stat_bgwriter

В представлении pg_stat_bgwriter всегда будет только одна строка, в которой будут представлены общие данные по всему кластеру.

Таблица A.7. Представление pg_stat_bgwriter

Тип столбца	Описание
checkpoints_timed bigint	Количество запланированных контрольных точек, которые уже были выполнены
checkpoints_req bigint	Количество запрошенных контрольных точек, которые уже были выполнены
checkpoint_write_time double precision	Общее время, которое было затрачено на этап обработки контрольной точки, в котором файлы записываются на диск, в миллисекундах

Тип столбца	Описание
checkpoint_sync_time double precision	Общее время, которое было затрачено на этап обработки контрольной точки, в котором файлы синхронизируются с дис ком, в миллисекундах
buffers_checkpoint bigint	Количество буферов, записанных при выполнении контрольных точек
buffers_clean bigint	Количество буферов, записанных фоновым процессом записи
maxwritten_clean bigint	Сколько раз фоновый процесс записи останавливал сброс грязных страниц на диск из-за того, что записал слишком много буферов
buffers_backend bigint	Количество буферов, записанных самим серверным процессом
buffers_backend_fsync bigint	Сколько раз серверному процессу пришлось выполнить fsync самостоятельно (обычно фоновый процесс записи сам обрабаты вает эти вызовы, даже когда серверный процесс выполняет за пись самостоятельно)
buffers_alloc bigint	Количество выделенных буферов
stats_reset timestamp with time zone	Последнее время сброса этих статистических данных

A.8. pg_stat_database

Представление pg_stat_database содержит по одной строке со статистикой уровня базы данных для каждой базы кластера, и ещё одну для общих объектов.

Таблица A.8. Представление pg_stat_database

Тип столбца	Описание
datid oid	OID базы данных либо 0 для объектов, относящихся к общим отношениям
datname name	Имя базы данных либо NULL для общих объектов.
numbackends integer	Количество обслуживающих процессов, в настоящее время подключённых к этой базе данных, либо NULL для общих объектов. Это единственный столбец в представлении, значение в котором отражает текущее состояние; все другие столбцы возвращают суммарные значения со времени последне го сброса статистики.
xact_commit bigint	Количество зафиксированных транзакций в этой базе данных
xact_rollback bigint	Количество транзакций в этой базе данных, для которых был выполнен от кат транзакции
blks_read bigint	Количество прочитанных дисковых блоков в этой базе данных
blks_hit bigint	Сколько раз дисковые блоки обнаруживались в буферном кеше, так что чте ние с диска не потребовалось (в значение входят только случаи обнаруже ния в буферном кеше PG360, а не в кеше файловой системы ОС)
tup_returned bigint	Количество строк, возвращённое запросами в этой базе данных
tup_fetched bigint	Количество строк, извлечённое запросами в этой базе данных
tup_inserted bigint	Количество строк, вставленное запросами в этой базе данных
tup_updated bigint	Количество строк, изменённое запросами в этой базе данных

Тип столбца	Описание
tup_deleted bigint	Количество строк, удалённое запросами в этой базе данных
conflicts bigint	Количество запросов, отменённых из-за конфликта с восстановлением в этой базе данных. (Конфликты происходят только на ведомых серверах)
temp_files bigint	Количество временных файлов, созданных запросами в этой базе данных. Подсчитываются все временные файлы независимо от причины их создания (например, для сортировки или для хеширования) и независимо от установ ленного значения log_ temp_files
temp_bytes bigint	Общий объём данных, записанных во временные файлы запросами в этой базе данных. Учитываются все временные файлы, вне зависимости от того, по какой причине они созданы и вне зависимости от значения log_temp_files.
deadlocks bigint	Количество взаимных блокировок, зафиксированное в этой базе данных
checksum_failures bigint	Количество ошибок контрольных сумм в страницах данных этой базы (или общего объекта) либо NULL, если контрольные суммы не проверяются.
checksum_last_failure timestamp with time zone	Время выявления последней ошибки контрольной суммы в страницах дан ных этой базы (или общего объекта) либо NULL, если контрольные суммы не проверяются.
blk_read_time double precision	Время, которое затратили обслуживающие процессы в этой базе на чтение блоков из файлов данных, в миллисекундах (если включён параметр track_io_timing; в противном случае 0)
blk_write_time double precision	Время, которое затратили обслуживающие процессы в этой базе на запись блоков в файлы данных, в миллисекундах (если включён параметр track_io_timing; в противном случае 0)
stats_reset timestamp with time zone	Последнее время сброса этих статистических данных

A.9. pg_stat_database_conflicts

Представление pg_stat_database_conflicts для каждой базы данных будет содержать по одной строке со статистикой на уровне базы по отменам запросов, произошедшим вследствие конфликтов с процессами восстановления на ведомых серверах. В этом представлении будет содержаться только информация по ведомым серверам, поскольку на главных серверах конфликты не возникают.

Таблица A.9. Представление pg_stat_database_conflicts

Тип столбца	Описание
datid oid	OID базы данных
datname name	Имя базы данных
confl_tablespace bigint	Количество запросов в этой базе данных, отменённых из-за того, что таблич ные пространства были удалены
confl_lock bigint	Количество запросов в этой базе данных, отменённых по истечении времени ожидания блокировки
confl_snapshot bigint	Количество запросов в этой базе данных, отменённых из-за устаревших сним ков данных
confl_bufferpin bigint	Количество запросов в этой базе данных, отменённых из-за прикреплённых страниц буфера

Тип столбца	Описание
confl_deadlock bigint	Количество запросов в этой базе данных, отменённых из-за взаимных блоки ровок

A.10. pg_stat_all_tables

Представление pg_stat_all_tables будет содержать по одной строке на каждую таблицу в текущей базе данных (включая таблицы TOAST) со статистикой по обращениям к этой таблице. Представления pg_stat_user_tables и pg_stat_sys_tables содержат ту же самую информацию, но отфильтрованную так, чтобы показывать только пользовательские и системные таблицы соответственно.

Таблица A.10. Представление pg_stat_all_tables

Тип столбца	Описание
relid oid	OID таблицы
schemaname name	Имя схемы, в которой расположена эта таблица
relname name	Имя таблицы
seq_scan bigint	Количество последовательных чтений, произведённых в этой таблице
seq_tup_read bigint	Количество «живых» строк, прочитанных при последовательных чтени ях
idx_scan bigint	Количество сканирований по индексу, произведённых в этой таблице
idx_tup_fetch bigint	Количество «живых» строк, отобранных при сканированиях по индексу
n_tup_ins bigint	Количество вставленных строк
n_tup_upd bigint	Количество изменённых строк (включая изменения по схеме HOT)
n_tup_del bigint	Количество удалённых строк
n_tup_hot_upd bigint	Количество строк, обновлённых в режиме HOT (т. е. без отдельного из менения индекса)
n_live_tup bigint	Оценочное количество «живых» строк
n_dead_tup bigint	Оценочное количество «мёртвых» строк
n_mod_since_analyze bigint	Оценочное число строк, изменённых в этой таблице с момента послед него сбора статистики
n_ins_since_vacuum bigint	Примерное число строк, вставленных в эту таблицу с момента последне го сбора статистики
last_vacuum timestamp with time zone	Время последней очистки этой таблицы вручную (VACUUM FULL не учитывается)
last_autovacuum timestamp with time zone	Время последней очистки таблицы фоновым процессом автоочистки
last_analyze timestamp with time zone	Время последнего выполнения сбора статистики для этой таблицы вруч ную
last_autoanalyze timestamp with time zone	Время последнего выполнения сбора статистики для этой таблицы фо новым процессом автоочистки
vacuum_count bigint	Сколько раз очистка этой таблицы была выполнена вручную (VACUUM FULL не учитывается)
autovacuum_count bigint	Сколько раз очистка этой таблицы была выполнена фоновым процессом автоочистки
analyze_count bigint	Сколько раз сбор статистики для этой таблицы был выполнен вручную
autoanalyze_count bigint	Сколько раз сбор статистики для этой таблицы был выполнен фоновым процессом автоочистки

A.11. pg_stat_all_indexes

Представление pg_stat_all_indexes для каждого индекса в текущей базе данных будет содержать по одной строке со статистикой по обращениям к этому индексу. Представления pg_stat_user_indexes и pg_stat_sys_indexes содержат ту же самую информацию, но отфильтрованную так, чтобы показывать только пользовательские и системные индексы соответственно.

Таблица A.11. Представление pg_stat_all_indexes

Тип столбца	**Описание**
relid oid	OID таблицы для индекса
indexrelid oid	OID индекса
schemaname name	Имя схемы, в которой расположен индекс
relname name	Имя таблицы для индекса
indexrelname name	Имя индекса
idx_scan bigint	Количество произведённых сканирований по этому индексу
idx_tup_read bigint	Количество элементов индекса, возвращённых при сканированиях по этому индексу
idx_tup_fetch bigint	Количество живых строк таблицы, отобранных при простых сканированиях по этому индексу

Индексы могут использоваться при сканировании по индексу, при сканировании «битовой карты» индекса и в работе оптимизатора. Результаты сканирования битовых карт разных индексов могут объединяться логическим умножением или сложением, поэтому когда применяются битовые карты, сложно связать выборки отдельных строк с определёнными индексами. Поэтому при сканировании битовых карт увеличиваются счётчики pg_stat_all_indexes.idx_tup_read для задействованных индексов и счётчик pg_stat_all_tables.idx_tup_fetch для каждой таблицы, а pg_stat_all_indexes.idx_tup_fetch не меняется. Оптимизатор тоже обращается к индексам для проверки переданных констант, значения которых оказываются вне диапазона, записанного в статистике оптимизатора, так как эта статистика может быть неактуальной.

A.12. pg_statio_all_tables

Представление pg_statio_all_tables для каждой таблицы (включая таблицы TOAST) в текущей базе данных будет содержать по одной строке со статистикой по операциям ввода/вывода для этой таблицы. Представления pg_statio_user_tables и pg_statio_sys_tables содержат ту же самую информацию, но отфильтрованную так, чтобы показывать только пользовательские или системные таблицы соответственно.

Таблица A.12. Представление pg_statio_all_tables

Тип столбца	**Описание**
relid oid	OID таблицы
schemaname name	Имя схемы, в которой расположена эта таблица
relname name	Имя таблицы
heap_blks_read bigint	Количество дисковых блоков, прочитанных из этой таблицы

Тип столбца	Описание
heap_blks_hit bigint	Число попаданий в буфер для этой таблицы
idx_blks_read bigint	Количество дисковых блоков, прочитанных из всех индексов этой таблицы
idx_blks_hit bigint	Число попаданий в буфер для всех индексов по этой таблице
toast_blks_read bigint	Количество прочитанных дисковых блоков TOAST (если есть) для этой таблицы
toast_blks_hit bigint	Число попаданий в буфер в таблице TOAST для этой таблицы (если такие есть)
tidx_blks_read bigint	Количество прочитанных дисковых блоков из индекса по TOAST (если есть) для этой таблицы
tidx_blks_hit bigint	Число попаданий в буфер для индекса по TOAST (если есть) для этой таблицы

A.13. pg_statio_all_indexes

Представление pg_statio_all_indexes для каждого индекса в текущей базе данных будет содержать по одной строке со статистикой по операциям ввода/вывода для этого индекса. Представления pg_statio_user_indexes и pg_statio_sys_indexes содержат ту же самую информацию, но отфильтрованную так, чтобы показывать только пользовательские или системные индексы соответственно.

Таблица A.13. Представление pg_statio_all_indexes

Тип столбца	Описание
relid oid	OID таблицы для индекса
indexrelid oid	OID индекса
schemaname name	Имя схемы, в которой расположен индекс
relname name	Имя таблицы для индекса
indexrelname name	Имя индекса
idx_blks_read bigint	Количество дисковых блоков, прочитанных из этого индекса
idx_blks_hit bigint	Число попаданий в буфер для этого индекса

A.14. pg_statio_all_sequences

Представление pg_statio_all_sequences для каждой последовательности в текущей базе данных будет содержать по одной строке со статистикой по операциям ввода/вывода для этой последовательности.

Таблица A.14. Представление pg_statio_all_sequences

Тип столбца	Описание
relid oid	OID последовательности
schemaname name	Имя схемы, в которой расположена эта последовательность
relname name	Имя последовательности
blks_read bigint	Количество дисковых блоков, прочитанных из этой последовательности
blks_hit bigint	Число попаданий в буфер в этой последовательности

A.15. pg_stat_user_functions

Представление pg_stat_user_functions для каждой отслеживаемой функции будет содержать по одной строке со статистикой по выполнениям этой функции. Отслеживаемые функции определяются параметром track_functions.

Таблица A.15. Представление pg_stat_user_functions

Тип столбца	**Описание**
funcid oid	OID функции
schemaname name	Имя схемы, в которой расположена функция
funcname name	Имя функции
calls bigint	Сколько раз вызывалась функция
total_time double precision	Общее время, потраченное на выполнение этой функции и всех других функций, вызванных ею, в миллисекундах
self_time double precision	Общее время, потраченное на выполнение самой функции, без учёта дру гих функций, которые были ею вызваны, в миллисекундах

A.16. pg_stat_slru

PG360 обращается к некоторой информации на диске через кеши, работающие по принципу *SLRU* (simple least-recently-used, простое вытеснение давно не используемых). Представление pg_stat_slru содержит по одной строке для каждого SLRU-кеша и даёт статистику по обращениям к его страницам.

Таблица A.16. Представление pg_stat_slru

Тип столбца	**Описание**
name text	Имя SLRU-кеша
blks_zeroed bigint	Количество блоков, обнулённых при инициализации
blks_hit bigint	Количество случаев, когда дисковые блоки обнаруживались в SLRU-кеше, так что чтение с диска не потребовалось (здесь учитываются только случаи обнаружения в этом кеше, а не в файловом кеше ОС)
blks_read bigint	Количество дисковых блоков, прочитанных через этот SLRU-кеш
blks_written bigint	Количество дисковых блоков, записанных из этого SLRU-кеша
blks_exists bigint	Количество блоков, проверенных на предмет наличия в этом SLRU-ке ше
flushes bigint	Количество операций сброса «грязных» данных для этого SLRU-кеша
truncates bigint	Количество операций усечения для этого SLRU-кеша

Приложение Б

Б.1. Отслеживание выполнения ANALYZE

Во время выполнения ANALYZE представление pg_stat_progress_analyze будет содержать по одной строке для каждого обслуживающего процесса, выполняющего эту команду. Таблицы ниже показывают, какая информация будет отслеживаться, и поясняют, как её интерпретировать.

Таблица Б.1. Представление pg_stat_progress_analyze

Тип столбца	**Описание**
pid integer	Идентификатор (PID) обслуживающего процесса
datid oid	OID базы данных, к которой подключён этот обслуживающий процесс.
datname name	Имя базы данных, к которой подключён этот обслуживающий процесс.
relid oid	OID анализируемой таблицы.
phase text	Текущая фаза обработки (Значения при введены в Таблице Б.1.1.)
sample_blks_total bigint	Общее количество блоков кучи, которые попадут в выборку.
sample_blks_scanned bigint	Количество просканированных блоков кучи.
ext_stats_total bigint	Количество объектов расширенной статистики.
ext_stats_computed bigint	Количество вычисленных объектов расширенной статистики. Этот счёт чик увеличивается только в фазе computing extended statistics (вычисление расширенной статистики).
child_tables_total bigint	Количество дочерних таблиц.
child_tables_done bigint	Количество просканированных дочерних таблиц. Этот счётчик увеличива ется только в фазе acquiring inherited sample rows (извлечение строк выбор ки через наследование).
current_child_table_relid oid	OID дочерней таблицы, сканируемой в данный момент. Это поле содержит актуальное значение только в фазе acquiring inherited sample rows (извлече ние строк выборки через наследование).

Таблица Б.1.1. 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 вы борки.
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.

Б.2. 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. Таблицы ниже показывают, какая информация будет отслеживаться, и поясняют, как её интерпретировать.

Таблица Б.2. Представление pg_stat_progress_create_index

Тип столбца	**Описание**
pid integer	Идентификатор (PID) обслуживающего процесса
datid oid	OID базы данных, к которой подключён этот обслуживающий процесс.
datname name	Имя базы данных, к которой подключён этот обслуживающий процесс.
relid oid	OID таблицы, в которой создаётся индекс.
index_relid oid	OID создаваемого или перестраиваемого индекса. При выполнении CREATE INDEX в неблокирующем режиме содержит 0.
command text	Выполняемая команда: CREATE INDEX, CREATE INDEX CONCURRENTLY, REINDEX или REINDEX CONCURRENTLY.
phase text	Текущая фаза создания индекса (Значения при введены в Таблице Б.2.1)
lockers_total bigint	Общее число процессов, потребовавших ожидания, если таковые имеются.
lockers_done bigint	Число процессов, ожидание которых уже завершено.
current_locker_pid bigint	Идентификатор процесса, удерживающего конфликтующую блокировку в данный момент.
blocks_total bigint	Общее число блоков, которые должны быть обработаны в текущей фазе.
blocks_done bigint	Число блоков, уже обработанных в текущей фазе.
tuples_total bigint	Общее число кортежей, которые должны быть обработаны в текущей фазе.
tuples_done bigint	Число кортежей, уже обработанных в текущей фазе.
partitions_total bigint	При создании индекса в секционированной таблице этот столбец содержит общее число секций, в которых создаётся индекс.
partitions_done bigint	При создании индекса в секционированной таблице этот столбец содержит число секций, в которых индекс уже построен.

Таблица Б.2.1. Фазы CREATE INDEX

Фаза	**Описание**
initializing	Инициализация — процедура CREATE INDEX или REINDEX подготавлива ется 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 .

Фаза	**Описание**
building index	Построение индекса — код, реализующий метод доступа, строит индекс. В этой фазе методы доступа, поддерживающие отслеживание процесса, переда ют 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 опера ции в неблокирующем режиме. Выполнение процедуры в этой фазе отражает ся 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 (пока зывающем общий размер индекса) и blocks_done .
index validation: sorting tuples	Проверка индекса: сортировка кортежей — процедура CREATE 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, со бранные в предыдущих двух фазах. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выполнение процедуры в этой фазе от ражается 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 содержи мое таблицы. Эта фаза пропускается при выполнении операции в неблокиру ющем 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 — проце дура REINDEX CONCURRENTLY ожидает завершения транзакций, которые удерживают блокировки чтения, прежде чем пометить старый индекс как не рабочий. 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 удер живают блокировки чтения, прежде чем удалить старый индекс. Эта фаза пропускается при выполнении операции в неблокирующем режиме. Выпол нение процедуры в этой фазе отражается в столбцах lockers_total , lockers_done и current_locker_ pid .

Б.3. 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 Б.3. Представление pg_stat_progress_vacuum

Тип столбца	**Описание**
pid integer	Идентификатор (PID) обслуживающего процесса
datid oid	OID базы данных, к которой подключён этот обслуживающий процесс.
datname name	Имя базы данных, к которой подключён этот обслуживающий процесс.
relid oid	OID очищаемой таблицы.
phase text	Текущая фаза очистки ((Значения при введены в Таблице Б.3.1)
heap_blks_total bigint	Общее число блоков кучи в таблице. Это число отражает состояние в на чале сканирования; блоки, добавленные позже, не будут (и не должны) обрабатываться текущей командой VACUUM.
heap_blks_scanned bigint	Число просканированных блоков кучи. Так как для оптимизации сканиро вания применяется карта видимости, некоторые блоки могут пропускать ся без осмотра; пропущенные блоки входят в это общее число, так что по завершении очистки это число станет равно heap_blks_total . Этот счётчик увеличивается только в фазе scanning heap.
heap_blks_vacuumed bigint	Число очищенных блоков кучи. Если в таблице нет индексов, этот счёт чик увеличивается только в фазе vacuuming heap (очистка кучи). Блоки, не содержащие «мёртвых» кортежей, при этом пропускаются, так что этот счётчик иногда может увеличиваться резкими рывками.
index_vacuum_count bigint	Количество завершённых циклов очистки индекса.
max_dead_tuples bigint	Число «мёртвых» кортежей, которое мы можем сохранить, прежде чем потребуется выполнить цикл очистки индекса, в зависимости от maintenance_work_mem.
num_dead_tuples bigint	Число «мёртвых» кортежей, собранных со времени последнего цикла очистки индекса.

Таблица Б.3.1. 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 страни ца, 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 ин дексах. Это происходит после завершения полного сканирования кучи и очистки индексов и кучи.
truncating heap	Усечение кучи — VACUUM в настоящее время усекает кучу, чтобы воз вратить ОС объём пустых страниц в конце отношения. Это происходит по сле уборки индексов.
performing final cleanup	Выполнение окончательной очистки — VACUUM выполняет окончатель ную очистку. На этой стадии VACUUM очищает карту свободного про странства, обновляет статистику в pg_class и передаёт статистику сборщи ку статистики, После этой фазы VACUUM завершит свою работу.

Б.4. 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 Б.4. Представление pg_stat_progress_cluster

Тип столбца	**Описание**
pid integer	Идентификатор (PID) обслуживающего процесса
datid oid	OID базы данных, к которой подключён этот обслуживающий процесс.
datname name	Имя базы данных, к которой подключён этот обслуживающий процесс.
relid oid	OID обрабатываемой таблицы.
command text	Выполняемая команда: CLUSTER или VACUUM FULL.
phase text	Текущая фаза обработки (Значения при введены в Таблице Б.4.1.)
cluster_index_relid oid	Если таблица сканируется по индексу, это поле содержит OID данного индекса, а иначе — 0.
heap_tuples_scanned bigint	Число просканированных кортежей кучи. Этот счётчик увеличивается только в фазе seq scanning heap, index scanning heap или writing new heap.
heap_tuples_written bigint	Число записанных кортежей кучи. Этот счётчик увеличивается только в фазе seq scanning heap, index scanning heap или writing new heap.
heap_blks_total bigint	Общее число блоков кучи в таблице. Это число отражает состояние в начале фазы seq scanning heap.
heap_blks_scanned bigint	Число просканированных блоков кучи. Этот счётчик увеличивается только в фазе seq scanning heap.
index_rebuild_count bigint	Число перестроенных индексов. Это счётчик увеличивается только в фазе rebuilding index.

Таблица Б.4.1. 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 об разом.
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.

Б.5. Отслеживание выполнение 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 Б.5. Представление pg_stat_progress_basebackup

Тип столбца	**Описание**
pid integer	Идентификатор процесса-передатчика WAL.
phase text	Текущая фаза обработки (Значения при введены в Таблице Б.5.1)
backup_total bigint	Общий объём данных, который будет передан. Этот объём пример но оценивается и выдаётся в начале фазы streaming database files (передача файлов данных). Это лишь приблизительная оценка, так как база данных может измениться в фазе streaming database files и в резервную копию позже может быть добавлен WAL. Это значение устанавливается равным backup_streamed , когда фактически пере данный объём начинает превышать рассчитанный предварительно. Когда расчёт оценки в pg_basebackup отключён (то есть передан па раметр --no-estimate-size), это поле содержит NULL.
backup_streamed bigint	Объём переданных данных. Этот показатель увеличивается только в фазе streaming database files (передача файлов данных) или transferring wal files (передача файлов WAL).
tablespaces_total bigint	Общее число табличных пространств, которые будут переданы.
tablespaces_streamed bigint	Число переданных табличных пространств. Этот счётчик увеличи вается только в фазе streaming database files (передача файлов дан ных).

Таблица Б.5.1. 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 ко личество файлов данных, которые будут передаваться при создании базовой копии.
streaming database files	Процесс-передатчик WAL в настоящий момент передаёт файлы дан ных 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 за канчивается сразу после данной фазы.

Фаза	**Описание**
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.

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Входящий № сопроводительного докум. и дата	Подп.	Дата
	измененных	замененных	новых	аннулированных					