Федеральное государственное унитарное предприятие
Российский федеральный ядерный центр
Всероссийский научно-исследовательский институт экспериментальной физики

УТВЕРЖДЕН
07623615.00082-02 32 01-ЛУ

КОМПЛЕКС ПРОГРАММ В ЗАЩИЩЕННОМ ИСПОЛНЕНИИ
«СИСТЕМА ПОЛНОГО ЖИЗНЕННОГО ЦИКЛА ИЗДЕЛИЙ
«ЦИФРОВОЕ ПРЕДПРИЯТИЕ»

Программный модуль «Система управления базами данных «Синергия-БД»

Руководство системного программиста

Часть 1

07623615.00082-02 32 01-1

Листов 172

2021

СОДЕРЖАНИЕ

07623615.00082-02 32 01-2 Руководство системного программиста. Часть 2

1. АДМИНИСТРИРОВАНИЕ СУБД «СИНЕРГИЯ-БД»

1.1. Установка СУБД «Синергия-БД»

1.1.1. Astra Linux Special Edition

Операционные системы Astra Linux Special Edition построены на базе Debian Linux. Способы установки пакетов для обеих операционных систем одинаковы. Установка производится с помощью пакетного менеджера `apt`. В дальнейшем все указанные ниже команды выполняются от имени суперпользователя.

Синергия-БД поддерживает следующие режимы установки:

1. Быстрая установка и настройка.

Пакет `postgresql-sdb-13` устанавливает и настраивает все компоненты, необходимые для получения готовой к использованию конфигурации, как клиентских, так и серверных компонентов. Выберите этот вариант, если вы планируете установить только один экземпляр Синергия-БД и вас не беспокоят возможные конфликты с другими продуктами на базе PostgreSQL.

2. Расширенная установка.

Вы можете выбрать любые пакеты, требующиеся для ваших целей, включая пакеты для разработчиков. Этот вариант требует ручной настройки, поэтому для него необходимо хорошо разбираться в Linux и понимать архитектуру PostgreSQL. Это единственно возможный вариант, если вы планируете реализовать один из следующих сценариев использования Синергия-БД:

3. Установка одновременно нескольких версий Синергия-БД либо установка вместе с другими продуктами на базе PostgreSQL.

4. Обновление с другой версии или миграция с другого продукта на базе PostgreSQL.

5. Управление сервером Синергия-БД с использованием программных средств обеспечения высокой степени доступности, таких как pacemaker, вместо стандартных системных механизмов управления службами.

1.1.2. Быстрая установка и настройка

Если вам нужно установить только один экземпляр Синергия-БД и вы не собираетесь использовать никакие другие продукты на базе PostgreSQL в вашей системе, вы можете использовать режим быстрой установки. Типичная процедура установки в этом случае выглядит так:

- установите пакет `postgresql-sdb-13`. При этом по зависимостям установятся все требуемые компоненты, будет создана база данных по умолчанию, запущен сервер баз данных и настроен автозапуск сервера при загрузке системы, а все предоставляемые программы станут доступными в пути `PATH`. В режиме быстрой установки кластер баз данных инициализируется с включёнными контрольными суммами.

После завершения установки вы можете запустить `psql` от имени пользователя `postgres` и подключиться к только что созданной базе данных, находящийся в каталоге данных `/var/lib/sdb/sdb-11/data`.

Так как база данных по умолчанию создаётся скриптом pg-setup, путь к каталогу данных сохраняется в файле `/etc/default/postgresql-sdb-11`. Все последующие команды `pg-setup`, а также любые команды, управляющие службой Синергия-БД, будут нацелены именно на эту базу данных.

1.1.3. Расширенная установка

Разделение дистрибутива на несколько пакетов позволяет произвести инсталляцию по-разному для различных применений: для серверов баз данных, клиентских систем или рабочих станций разработчиков. Такие инсталляции необходимо настраивать вручную, но это даёт дополнительную гибкость при использовании продукта. Вы можете установить несколько версий Синергия-БД параллельно, а также вместе с другими продуктами PostgreSQL. В частности, это может потребоваться при осуществлении обновления или при миграции с другого продукта на базе PostgreSQL.

Для осуществления расширенной установки выполните следующие действия:

1. Выберите пакеты Синергия-БД, требующиеся для ваших целей, и

установите их с помощью стандартных для вашего дистрибутива Linux команд. Доступные пакеты перечислены в таблице 1.1.

В результате все файлы будут установлены в каталог /opt/sdb/sdb-13.

Запустите от имени root утилиту pg-wrapper, чтобы добавить и клиентские, и серверные установленные программы в путь поиска PATH, а также включить страницы man по SQL в файл конфигурации страниц man. Эта утилита входит в состав пакета postgresql-sdb-13-client.

```
/opt/sdb/sdb-11/bin/pg-wrapper links update
```

Подробнее о разрешении возможных конфликтов рассказывается в pg-wrapper.

2. Если вы устанавливаете пакет postgresql-sdb-13-server, то создайте начальную базу данных, запустив вспомогательный скрипт pg-setup от имени root с ключом initdb:

```
/opt/sdb/sdb-11/bin/pg-setup initdb [параметры_initdb]
```

Здесь initdb_options — обычные параметры программы initdb.

Скрипт pg-setup выполняет административные операции с базами данных от имени пользователя postgres. Если вы не укажете никакие параметры initdb, база по умолчанию создаётся в каталоге /var/lib/sdb/sdb-11/data с параметрами локализации, определяемыми переменной окружения LANG текущего сеанса. Переменные окружения LC_* при этом игнорируются.

Так как база данных по умолчанию создаётся скриптом pg-setup, путь к каталогу данных сохраняется в файле /etc/default/postgresql-sdb-13. Все последующие команды pg-setup, а также любые команды, управляющие службой Синергия-БД, будут нацелены именно на эту базу данных.

Запустите сервер с помощью pg-setup от имени root следующим образом:

```
/opt/sdb/sdb-11/bin/pg-setup service start
```

Как и сервер PostgreSQL, сервер Синергия-БД работает под именем пользователя postgres.

1.1.3.1. Выбор устанавливаемых пакетов

В таблице 1.1 перечислены все имеющиеся пакеты СУБД «Синергия-БД».

Таблица 1.1 – Пакеты СУБД «Синергия-БД»

Пакет	Описание
`postgresql-sdb-13`	Пакет верхнего уровня, устанавливающий и настраивающий СУБД «Синергия-БД» для серверных и клиентских систем. Не используйте этот пакет при обновлении или миграции.
`postgresql-sdb-13-client`	Стандартные клиентские приложения, такие как `psql` и `pg_dump`.
`postgresql-sdb-13-libs`	Общие библиотеки, требующиеся для развёртывания клиентских приложений, включая `libpq`; библиотеки времени выполнения для обработчика `ECPG`.
`postgresql-sdb-13-server`	Сервер СУБД «Синергия-БД» и серверный язык программирования PL/pgSQL.
`postgresql-sdb-13-contrib`	Дополнительные расширения и программы, разворачиваемые на серверах баз данных.
`postgresql-sdb-13-dev`	Заголовочные файлы и библиотеки для разработки клиентских приложений и серверных расширений.
`postgresql-sdb-13-plperl`	Реализация языка Perl для программирования на стороне сервера.
`postgresql-sdb-13-plpython`	Реализация языка Python для программирования на стороне сервера.
`postgresql-sdb-13-plpython3`	Реализация языка Python 3 для программирования на стороне сервера.
`postgresql-sdb-13-pltcl`	Реализация языка Tcl для программирования на стороне сервера.

Окончание таблицы 1.1

Пакет	Описание
postgresql-sdb-13-docs	Документация на английском языке.
postgresql-sdb-13-docs-ru	Документация на русском языке.
postgresql-sdb-13-dbg	Отладочная информация.

Для серверных инсталляций требуются как минимум следующие пакеты:

- postgresql-sdb-13-server;
- postgresql-sdb-13-client;
- postgresql-sdb-13-libs.

Для использования дополнительных расширений СУБД «Синергия-БД» вы должны также установить пакет postgresql-sdb-13-contrib. Пакет postgresql-sdb-13-server зависит от пакета postgresql-sdb-13-contrib, поэтому последний должен всегда устанавливаться вместе с сервером.

Для клиентских инсталляций обычно достаточно установить пакеты postgresql-sdb-13-client и postgresql-sdb-13-libs. Если вы используете независимые приложения и вам не нужны стандартные клиентские утилиты, такие как psql, вы можете установить только пакет postgresql-sdb-13-libs.

Для рабочих станций разработчиков необходимый минимум составляют следующие пакеты:

- postgresql-sdb-13-libs;
- postgresql-sdb-13-devel/ postgresql-sdb-13-dev.

Также вы можете установить и настроить сервер с тестовой базой данных.

1.1.3.2. Настройка автоматического запуска сервера

Если вы производите расширенную установку, автоматический запуск сервера по умолчанию отключается. После создания базы данных по умолчанию вы можете настроить автоматический запуск сервера при загрузке системы, используя имеющиеся в вашей операционной системе механизмы управления службами или сторонние средства обеспечения высокой степени доступности. Для облегчения этой задачи в пакете postgresql-sdb-13-server поставляется скрипт pg-setup,

помещаемый в каталог /opt/sdb/sdb-13/bin.

Для настройки автозапуска сервера запустите скрипт pg-setup со следующими параметрами:

```
pg-setup service enable
```

Если требуется, вы можете отключить автозапуск сервера, используя тот же скрипт:

```
pg-setup service disable
```

Вы также можете использовать системные средства управления службами, непосредственно задействуя скрипты для систем инициализации SysV init.d и systemd, предоставляемые в пакете postgresql-sdb-13-server. СУБД «Синергия-БД» поддерживает следующие механизмы управления службами: файл службы systemd, и скрипт init.d в стиле SysV.

Для включения автоматического запуска сервера средствами systemd выполните следующую команду:

```
systemctl enable postgresql-sdb-13.service
```

Для использования скрипта init.d воспользуйтесь скриптом update-rc.d. За подробностями обратитесь к соответствующей странице man.

1.1.4. Установка на рабочих станциях разработчиков

Хотя может быть достаточно установить пакеты postgresql-sdb-13-libs и postgresql-sdb-13-devel/postgresql-sdb-13-dev, обычно на машинах разработчиков удобно иметь и установленный сервер. Для быстрой установки вы можете воспользоваться пакетом postgresql-sdb-13, который автоматически настроит предоставляемые клиентские и серверные программы, а также создаст начальную базу данных. Однако если вы планируете использовать одновременно несколько продуктов на базе PostgreSQL, следуйте инструкциям по расширенной установке.

Чтобы скомпилировать программы с библиотеками СУБД «Синергия-БД», используя утилиту pg_config, поставляемую с СУБД «Синергия-БД», необходимо, чтобы её путь был указан в переменной PATH до путей каких-либо

других версий pg_config. Если в вашей системе нет других версий pg_config, вы можете воспользоваться утилитой pg-wrapper, поставляемой в пакете postgresql-sdb-13-client, для создания символической ссылки на pg_config в стандартном каталоге исполняемых файлов.

Для компиляции программ с использованием pkg-config добавьте путь /opt/sdb/sdb-13/lib/pkgconfig/ в переменную среды PKG_CONFIG_PATH.

Если вы хотите компилировать расширения СУБД «Синергия-БД» с поддержкой встраивания JIT, выполните дополнительные требования:

- установите пакет LLVM для разработки и компилятор Clang. Вы должны выбрать пакеты той же версии, какая использовалась для сборки пакета postgresql-sdb-13-jit, устанавливаемом на сервере. Определить версию, требующуюся для текущего выпуска СУБД «Синергия-БД», вы можете по значению CLANG в файле /opt/sdb/sdb-13/lib/pgxs/src/Makefile.global;
- выполняя команды make и make install, добавьте параметр with-llvm=yes для компиляции и установки файлов с битовым кодом для вашего расширения. По умолчанию компиляции битового кода отключена, так как она зависит от доступности компилятора Clang.

1.1.4.1. Использование сторонних программ с СУБД «Синергия-БД»

Чтобы использовать сервер СУБД «Синергия-БД» с клиентской программой из стороннего продукта, вы можете установить версию библиотек PostgreSQL, с которыми компилировалась эта программа. Например, если программа поставляется в составе ванильного PostgreSQL, вам может потребоваться установить пакеты libpq или postgresql-libs, предоставляемые для вашего дистрибутива Linux. В этом случае программа, возможно, не будет использовать некоторые новые возможности сервера СУБД «Синергия-БД», но скорее всего она и не поддерживает их.

Если вы предпочитаете использовать библиотеки СУБД «Синергия-БД» со

сторонней программой или хотели бы задействовать новую возможность, которая не требует изменения самого клиентского приложения, например, проверку подлинности SCRAM, вы можете перекомпилировать вашу программу с библиотеками СУБД «Синергия-БД».

Если вы создаёте пакеты `.rpm` или `.deb` для вашей программы, рекомендуется сделать следующее:

- добавьте путь `/opt/sdb/sdb-13/bin` в переменную `PATH` в сборочных скриптах (в файле `.spec` или `debian/rules`);
- добавьте `postgresql-sdb-13-dev` в теги `BuildDepends` или `BuildRequires` описания пакета вашей программы.

Тем самым вы обеспечите вызов в процессе создания пакетов нужной версии `pg_config` при каждой пересборке пакета исходного кода.

1.1.5. Настройка нескольких экземпляров СУБД «Синергия-БД»

Чтобы настроить в ОС несколько экземпляров сервера СУБД «Синергия-БД» с разными каталогами данных, нужно проделать следующее:

1) установите и настройте СУБД «Синергия-БД»;

2) после создания начальной базы данных запустите `initdb` и укажите путь к другому каталогу данных и любые другие параметры, требующиеся для инициализации другого экземпляра сервера;

3) задайте другие порты для экземпляров серверов в соответствующих файлах `postgresql.conf`, чтобы не допустить конфликтов;

4) если требуется, настройте автоматический запуск сервера следующим образом:

- создайте копию файла `/etc/init.d/postgresql-sdb-13` или `/lib/systemd/system/postgresql-sdb-13.service` с другим именем и поменяйте в ней путь к каталогу данных;
- включите автоматический запуск сервера, используя скрипты автозапуска, предоставленные для вашего системного механизма управления службами,

вместо `pg-setup`. Убедитесь в том, что вы используете переименованные копии скриптов, созданные на предыдущем шаге.

1.1.6. Установка дополнительно поставляемых модулей

Синергия-БД поставляется с набором дополнительных серверных расширений, или модулей. В Linux они распространяются в отдельном пакете `postgrespro-contrib`.

Установив двоичные файлы, вы должны развернуть в базе данных дополнительные нужные вам расширения. В большинстве случае для этого достаточно выполнить команду `CREATE EXTENSION`. Однако для некоторых расширений также требуется, чтобы при запуске сервера загружались определённые разделяемые библиотеки. Если вы хотите использовать такие расширения, вам нужно настроить параметр:

```
shared_preload_libraries = 'lib1, lib2, lib3'
```

в файле `postgresql.conf` вашего экземпляра СУБД «Синергия-БД» и перезапустить сервер, а затем выполнить команду `CREATE EXTENSION`.

Более конкретные инструкции по установке и настройке определённого расширения вы можете найти в его документации.

Чтобы получить список расширений, доступных в вашей инсталляции СУБД «Синергия-БД», просмотрите системный каталог `pg_available_extensions`.

1.1.7. Миграция на СУБД «Синергия-БД»

Разные основные версии СУБД «Синергия-БД», а также разные продукты на базе PostgreSQL, имеющие одинаковую основную версию, могут иметь несовместимые на двоичном уровне базы данных, так что просто заменить программу сервера и продолжить использовать его в общем случае нельзя. Чтобы преобразовать базы данных с более старой основной версии, вам потребуется выполнить выгрузку/загрузку данных с помощью `pg_dumpall` или использовать `pg_upgrade`.

1.2. Подготовка к работе и сопровождение сервера

1.2.1. Учётная запись пользователя СУБД «Синергия-БД»

Как и любую другую службу, доступную для внешнего мира, Синергия-БД рекомендуется запускать под именем отдельного пользователя. Эта учётная запись должна владеть только данными, которыми управляет сервер, и разделять её с другими службами не следует. Например, не стоит использовать для этого пользователя `nobody`. Также рекомендуется не устанавливать под именем этого пользователя исполняемые файлы, чтобы их нельзя было подменить в случае компрометации системы.

Для создания пользователя в Unix-подобной системе следует искать команду `useradd` или `adduser`. В качестве имени пользователя часто используется `postgres`, и именно это имя предполагается в данной документации, но вы можете выбрать и другое, если захотите.

1.2.2. Создание кластера баз данных

Прежде чем вы сможете работать с базами данных, вы должны проинициализировать область хранения баз данных на диске. Мы называем это хранилище *кластером баз данных* (в SQL применяется термин «кластер каталога»). Кластер баз данных представляет собой набор баз, управляемых одним экземпляром работающего сервера. После инициализации кластер будет содержать базу данных с именем postgres, предназначенную для использования по умолчанию утилитами, пользователями и сторонними приложениями. Сам сервер баз данных не требует наличия базы postgres, но многие внешние вспомогательные программы рассчитывают на её существование. При инициализации в каждом кластере создаётся ещё одна база, с именем `template1`. Как можно понять из имени, она применяется впоследствии в качестве шаблона создаваемых баз данных; использовать её в качестве рабочей не следует.

С точки зрения файловой системы, кластер баз данных представляет собой один каталог, в котором будут храниться все данные. Мы называем его *каталогом данных* или *областью данных*. Где именно хранить данные, вы абсолютно свободно

можете выбирать сами. Какого-либо стандартного пути не существует, но часто данные размещаются в /var/lib/sdb/sdb-13/data или в /var/lib/sdb/sdb-13/data. Для инициализации кластера баз данных применяется команда initdb, которая устанавливается в составе Синергия-БД. Расположение кластера базы данных в файловой системе задаётся параметром -D, например:

```
$initdb -D /var/lib/sdb/sdb-13/data
```

Заметьте, что эту команду нужно выполнять от имени учётной записи СУБД «Синергия-БД», о которой говорится в предыдущем разделе.

Также можно запустить команду initdb, воспользовавшись программой pg_ctl, примерно так:

```
$pg_ctl -D /var/lib/sdb/sdb-13/data initdb
```

Этот вариант может быть удобнее, если вы используете pg_ctl для запуска и остановки сервера, так как pg_ctl будет единственной командой, с помощью которой вы будете управлять экземпляром сервера баз данных.

Команда initdb попытается создать указанный вами каталог, если он не существует. Конечно, она не сможет это сделать, если initdb не будет разрешено записывать в родительский каталог. Вообще рекомендуется, чтобы пользователь Синергия-БД был владельцем не только каталога данных, но и родительского каталога, так что такой проблемы быть не должно. Если же и нужный родительский каталог не существует, вам нужно будет сначала создать его, используя права root, если вышестоящий каталог защищён от записи. Таким образом, процедура может быть такой:

```
root# mkdir /var/lib/sdb/sdb-13
root# chown postgres /var/lib/sdb/sdb-13
root# su postgres
postgres$ initdb -D /var/lib/sdb/sdb-13/data
```

Команда initdb не будет работать, если указанный каталог данных уже существует и содержит файлы; это мера предохранения от случайной перезаписи существующей инсталляции.

Так как каталог данных содержит все данные базы, очень важно защитить его

от неавторизованного доступа. Для этого initdb лишает прав доступа к нему всех пользователей, кроме пользователя Синергия-БД и, возможно, его группы. Если группе разрешается доступ, то только для чтения. Это позволяет непривилегированному пользователю, входящему в одну группу с владельцем кластера, делать резервные копии данных кластера или выполнять другие операции, для которых достаточно доступа только для чтения.

Заметьте, чтобы корректно разрешить или запретить доступ группы к данным существующего кластера, необходимо выключить кластер и установить соответствующий режим для всех каталогов и файлов до запуска Синергия-БД. В противном случае в каталоге данных возможно смешение режимов. Для кластеров, к которым имеет доступ только владелец, требуется установить режим 0700 для каталогов и 0600 для файлов, а для кластеров, в которых также разрешается чтение группой, режим 0750 для каталогов и 0640 для файлов.

Однако, даже когда содержимое каталога защищено, если проверка подлинности клиентов настроена по умолчанию, любой локальный пользователь может подключиться к базе данных и даже стать суперпользователем. Если вы не доверяете другим локальным пользователям, мы рекомендуем использовать один из параметров команды initdb: -W, --pwprompt или --pwfile и назначить пароль суперпользователя баз данных. Кроме того, воспользуйтесь параметром -A md5 или -A password и отключите разрешённый по умолчанию режим аутентификации trust; либо измените сгенерированный файл pg_hba.conf после выполнения initdb, но перед тем, как запустить сервер в первый раз. Возможны и другие подходы — применить режим проверки подлинности peer или ограничить подключения на уровне файловой системы.

Команда initdb также устанавливает для кластера баз данных локаль по умолчанию. Обычно она просто берёт параметры локали из текущего окружения и применяет их к инициализируемой базе данных. Однако можно выбрать и другую локаль для базы данных. Команда initdb задаёт порядок сортировки по умолчанию для применения в определённом кластере баз данных, и, хотя новые базы данных могут создаваться с иным порядком сортировки, порядок в базах-

шаблонах, создаваемых `initdb`, можно изменить, только если удалить и пересоздать их. Также учтите, что при использовании локалей, отличных от C и POSIX, возможно снижение производительности. Поэтому важно правильно выбрать локаль с самого начала.

Команда `initdb` также задаёт кодировку символов по умолчанию для кластера баз данных. Обычно она должна соответствовать кодировке локали.

Для локалей, отличных от C и POSIX, порядок сортировки символов зависит от системной библиотеки локализации, а он, в свою очередь, влияет на порядок ключей в индексах. Поэтому кластер нельзя перевести на несовместимую версию библиотеки ни путём восстановления снимка, ни через двоичную репликацию, ни перейдя на другую операционную систему или обновив её версию.

1.2.3. Использование дополнительных файловых систем

Во многих инсталляциях кластеры баз данных создаются не в «корневом» томе, а в отдельных файловых системах (томах). Если вы решите сделать так же, то не следует выбирать в качестве каталога данных самый верхний каталог дополнительного тома (точку монтирования). Лучше всего создать внутри каталога точки монтирования каталог, принадлежащий пользователю Синергия-БД, а затем создать внутри него каталог данных. Это исключит проблемы с разрешениями, особенно для таких операций, как `pg_upgrade`, и при этом гарантирует чистое поведение в случае, если дополнительный том окажется отключён.

1.2.4. Использование сетевых файловых систем

Во многих инсталляциях кластеры баз данных создаются в сетевых файловых ресурсах. Иногда это реализуется с применением сетевой файловой системы (NFS, Network File System) или сетевых хранилищ (NAS, Network Attached Storage), использующих NFS внутри. Синергия-БД не делает ничего специфического с файловыми системами NFS, то есть он предполагает, что NFS работает точно так же, как и локально подключённые диски. Но если реализация клиента или сервера NFS не обеспечивает стандартное поведение файловой системы, это чревато

нестабильной работой. В частности, возможно разрушение данных при отложенной (асинхронной) записи на сервер NFS. Поэтому, по возможности, во избежание таких проблем монтируйте файловые системы NFS в синхронном режиме (без кеширования). Кроме того, не рекомендуется применять мягкое монтирование файловой системы NFS.

В сетях хранения данных (SAN, Storage Area Networks) обычно используются собственные протоколы, не NFS, и они могут быть не подвержены (а могут быть и подвержены) этим рискам. По вопросам гарантии согласованности данных обратитесь к документации производителя. Синергия-БД не может быть надёжнее файловой системы, которую он использует.

1.3. Запуск и останов СУБД «Синергия-БД»

1.3.1. Запуск сервера

Чтобы кто-либо смог обратиться к базе данных, необходимо сначала запустить сервер баз данных. Программа сервера называется `postgres`. Для работы программа `postgres` должна знать, где найти данные, которые она будет использовать. Указать это местоположение позволяет параметр `-D`. Таким образом, проще всего запустить сервер, выполнив команду:

```
$ postgres -D /var/lib/sdb/sdb-13/data
```

в результате которой сервер продолжит работу в качестве процесса переднего плана. Запускать эту команду следует под именем учётной записи Синергия-БД. Без параметра `-D` сервер попытается использовать каталог данных, указанный в переменной окружения `PGDATA`. Если и эта переменная не определена, сервер не будет запущен.

Однако обычно лучше запускать `postgres` в фоновом режиме. Для этого можно применить обычный синтаксис, принятый в оболочке Unix:

```
$ postgres -D /var/lib/sdb/sdb-13/data >logfile 2>&1 &
```

Важно где-либо сохранять информацию, которую выводит сервер в каналы `stdout` и `stderr`, как показано выше. Это полезно и для целей аудита, и для диагностики проблем.

Программа postgres также принимает несколько других параметров командной строки. За дополнительными сведениями обратитесь к справочной странице postgres.

Такой вариант запуска довольно быстро может оказаться неудобным. Поэтому для упрощения подобных задач предлагается вспомогательная программа pg_ctl, например:

```
pg_ctl start -l logfile
```

запустит сервер в фоновом режиме и направит выводимые сообщения сервера в указанный файл журнала. Параметр -D для неё имеет то же значение, что и для программы postgres. С помощью pg_ctl также можно остановить сервер.

Обычно возникает желание, чтобы сервер баз данных сам запускался при загрузке операционной системы. Скрипты автозапуска для разных систем разные, но в составе Синергия-БД предлагается несколько типовых скриптов в каталоге contrib/start-scripts. Для установки такого скрипта в систему требуются права root.

В различных системах приняты разные соглашения о порядке запуска служб в процессе загрузки. Во многих системах для этого используется файл /etc/rc.local или /etc/rc.d/rc.local. В других применяются каталоги init.d или rc.d. Однако при любом варианте запускаться сервер должен от имени пользователя Синергия-БД, но не root или какого-либо другого пользователя. Поэтому команду запуска обычно следует записывать в форме su postgres -c '...', например:

```
su postgres -c 'pg_ctl start -D /var/lib/sdb/sdb-11/data
-l serverlog'
```

Ниже приведены более конкретные предложения для нескольких основных ОС. Вместо указанных шаблонных значений необходимо подставить правильный путь к каталогу данных и фактическое имя пользователя.

В системах Linux вы можете либо добавить

```
/opt/sdb/sdb-13/bin/pg_ctl start -l logfile -D
/var/lib/sdb/sdb-13/data
```

в /etc/rc.d/rc.local или в /etc/rc.local, либо воспользоваться файлом contrib/start-scripts/linux в дереве исходного кода Синергия-БД.

Используя systemd, вы можете применить следующий файл описания службы (например, /etc/systemd/system/postgresql.service):

```
[Unit]
Description=PostgreSQL database server
Documentation=man:postgres(1)

[Service]
Type=notify
User=postgres
ExecStart=/opt/sdb/sdb-13/bin/postgres -D
/var/lib/sdb/sdb-13/data
ExecReload=/bin/kill -HUP $MAINPID
KillMode=mixed
KillSignal=SIGINT
TimeoutSec=0

[Install]
WantedBy=multi-user.target
```

Для использования Type=notify требуется, чтобы сервер был скомпилирован с указанием configure --with-systemd.

Особого внимания заслуживает значение тайм-аута. На момент написания этой документации по умолчанию в systemd принят тайм-аут 90 секунд, так что процесс, не сообщивший о своей готовности за это время, будет уничтожен. Но серверу Синергия-БД при запуске может потребоваться выполнить восстановление после сбоя, так что переход в состояние готовности может занять гораздо больше времени. Предлагаемое значение 0 отключает логику тайм-аута.

Когда сервер работает, идентификатор его процесса (PID) сохраняется в файле postmaster.pid в каталоге данных. Это позволяет исключить запуск нескольких экземпляров сервера с одним каталогом данных, а также может быть полезно для

выключения сервера.

1.3.2. Сбои при запуске сервера

Есть несколько распространённых причин, по которым сервер может не запуститься. Чтобы понять, чем вызван сбой, просмотрите файл журнала сервера или запустите сервер вручную (не перенаправляя его потоки стандартного вывода и ошибок) и проанализируйте выводимые сообщения. Ниже мы рассмотрим некоторые из наиболее частых сообщений об ошибках более подробно.

```
LOG:  could not bind IPv4 address "127.0.0.1": Address
already in use
HINT:  Is another postmaster already running on port
5432? If not, wait a few seconds and retry.
FATAL:  could not create any TCP/IP sockets
```

Это обычно означает именно то, что написано: вы пытаетесь запустить сервер на том же порту, на котором уже работает другой. Однако, если сообщение ядра не `Address already in use` или подобное, возможна и другая проблема. Например, при попытке запустить сервер с номером зарезервированного порта будут выданы такие сообщения:

```
$ postgres -p 666
LOG:  could not bind IPv4 address "127.0.0.1":
Permission denied
HINT:  Is another postmaster already running on port
666? If not, wait a few seconds and retry.
FATAL:  could not create any TCP/IP sockets
```

Следующее сообщение:

```
FATAL:  could not create shared memory segment: Invalid
argument
DETAIL:  Failed system call was shmget(key=5440001,
size=4011376640, 03600).
```

может означать, что установленный для вашего ядра предельный размер разделяемой памяти слишком мал для рабочей области, которую пытается создать Синергия-БД (в данном примере 4011376640 байт). Возможно также, что в вашем ядре вообще отсутствует поддержка разделяемой памяти в стиле System-V. В

качестве временного решения можно попытаться запустить сервер с меньшим числом буферов (shared_buffers), но в итоге вам, скорее всего, придётся переконфигурировать ядро и увеличить допустимый размер разделяемой памяти. Вы также можете увидеть это сообщение при попытке запустить несколько серверов на одном компьютере, если запрошенный ими объём разделяемой памяти в сумме превышает этот предел.

Сообщение:

```
FATAL:  could not create semaphores: No space left on
device
DETAIL:  Failed system call was semget(5440126, 17,
03600).
```

не означает, что у вас закончилось место на диске. Это значит, что установленное в вашем ядре предельное число семафоров System V меньше, чем количество семафоров, которое пытается создать Синергия-БД. Как и в предыдущем случае можно попытаться обойти эту проблему, запустив сервер с меньшим числом допустимых подключений (max_connections), но в конце концов вам придётся увеличить этот предел в ядре.

Если вы получаете ошибку «`illegal system call`» (неверный системный вызов), то, вероятнее всего, ваше ядро вовсе не поддерживает разделяемую память или семафоры. В этом случае вам остаётся только переконфигурировать ядро и включить их поддержку.

1.3.3. Проблемы с подключениями клиентов

Хотя ошибки подключений, возможные на стороне клиента, довольно разнообразны и зависят от приложений, всё же несколько проблем могут быть связаны непосредственно с тем, как был запущен сервер. Описание ошибок, отличных от описанных ниже, следует искать в документации соответствующего клиентского приложения.

```
psql: could not connect to server: Connection refused
        Is the server running on host "server.joe.com"
and accepting
        TCP/IP connections on port 5432?
```

Это общая проблема «я не могу найти сервер и начать взаимодействие с ним». Показанное выше сообщение говорит о попытке установить подключение по TCP/IP. Очень часто объясняется это тем, что сервер просто забыли настроить для работы по протоколу TCP/IP.

Кроме того, при попытке установить подключение к локальному серверу через Unix-сокет можно получить такое сообщение:

```
psql: could not connect to server: No such file or
directory
        Is the server running locally and accepting
        connections on Unix domain socket
"/tmp/.s.PGSQL.5432"?
```

Путь в последней строке помогает понять, к правильному ли адресу пытается подключиться клиент. Если сервер на самом деле не принимает подключения по этому адресу, обычно выдаётся сообщение ядра `Connection refused` (В соединении отказано) или `No such file or directory` (Нет такого файла или каталога), приведённое выше. Важно понимать, что `Connection refused` в данном контексте не означает, что сервер получил запрос на подключение и отверг его. В этом случае были бы выданы другие сообщения. Другие сообщения об ошибках, например, `Connection timed out` (Тайм-аут соединения) могут сигнализировать о более фундаментальных проблемах, например, о нарушениях сетевых соединений.

1.3.4. Останов сервера

Сервер баз данных можно отключить несколькими способами (вы выбираете тот или иной вариант отключения, посылая разные сигналы главному процессу postgres):

- `SIGTERM`

Запускает так называемое умное выключение. Получив `SIGTERM`, сервер перестает принимать новые подключения, но позволяет всем существующим сеансам закончить работу в штатном режиме. Сервер будет отключен только после завершения всех сеансов. Если сервер находится в режиме архивации,

сервер дополнительно ожидает выхода из этого режима. При этом в данном случае сервер позволяет устанавливать новые подключения, но только для суперпользователей (это исключение позволяет суперпользователю подключиться и прервать архивацию). Если, получая этот сигнал, сервер находится в процессе восстановления, восстановление и потоковая репликация будут прерваны только после завершения всех обычных сеансов.

- SIGINT

Запускает быстрое выключение. Сервер запрещает новые подключения и посылает всем работающим серверным процессам сигнал SIGTERM, в результате чего их транзакции прерываются и сами процессы завершаются. Управляющий процесс ждет, пока будут завершены все эти процессы и затем завершается сам. Если сервер находится в режиме архивации, архивация прерывается, так что архив оказывается неполным.

- SIGQUIT

Запускает немедленное выключение. Сервер посылает всем работающим серверным процессам сигнал SIGQUIT и ждёт их завершения. Если они не завершились в течение 5 секунд, сервер прерывает их сигналом SIGKILL. Основной серверный процесс завершается, как только завершаются все дочерние процессы, без выполнения обычного процесса завершения. Это приведёт к тому, что при следующем старте серверу придётся выполнить восстановление путём накатывания журналов упреждающей записи. Это режим стоит использовать только в крайних случаях.

Удобную возможность отправлять эти сигналы, отключающие сервер, предоставляет программа pg_ctl. Кроме того, соответствующий сигнал можно отправить с помощью команды kill. PID основного процесса postgres можно узнать, воспользовавшись программой ps, либо прочитав файл postmaster.pid в каталоге данных. Например, можно выполнить быстрое выключение так:

```
$ kill -INT `head -1 /var/lib/sdb/sdb-
13/data/postmaster.pid`
```

Для выключения сервера не следует использовать сигнал SIGKILL. При таком

выключении сервер не сможет освободить разделяемую память и семафоры, и, возможно, это придется делать вручную, чтобы сервер мог запуститься снова. Кроме того, при уничтожении главного процесса postgres сигналом `SIGKILL`, он не успеет передать этот сигнал своим дочерним процессам, так что может потребоваться завершать их также вручную.

Чтобы завершить отдельный сеанс, не прерывая работу других сеансов, воспользуйтесь функцией `pg_terminate_backend()` или отправьте сигнал `SIGTERM` дочернему процессу, обслуживающему этот сеанс.

2. ПОДГОТОВКА К РАБОТЕ И СОПРОВОЖДЕНИЕ СЕРВЕРА

В этой главе рассказывается, как организовать работу сервера баз данных и его взаимодействие с операционной системой, а также о некоторых задачах сопровождения.

2.1. Управление ресурсами ядра

Синергия-БД иногда может исчерпывать некоторые ресурсы операционной системы до предела, особенно при запуске нескольких копий сервера в одной системе или при работе с очень большими базами. В этом разделе описываются ресурсы ядра, которые использует Синергия-БД, и подходы к решению проблем, связанных с ограниченностью этих ресурсов.

2.1.1. Разделяемая память и семафоры

Разделяемая память и семафоры в совокупности называются средствами межпроцессного взаимодействия (IPC) в стиле System V (к этим средствам также относятся очереди сообщений, но они не имеют отношения к Синергия-БД). Эти средства необходимы для работы Синергия-БД.

Если эти механизмы полностью отсутствуют в системе, при запуске сервера обычно выдаётся ошибка `Illegal system call` (Неверный системный вызов). В этом случае единственный способ решить проблему - переконфигурировать ядро системы. Без них Синергия-БД просто не будет работать. Это довольно редкая ситуация, особенно с современными операционными системами.

Когда Синергия-БД превышает один из различных жёстких пределов IPC, сервер отказывается запускаться, но выдаёт полезное сообщение, говорящее об ошибке и о том, что с ней делать. Соответствующие параметры ядра в разных системах называются аналогично (они перечислены в таблице 2.1), но устанавливаются они по-разному. Ниже предлагаются способы их изменения.

Таблица 2.1 – Параметры IPC

Имя	Описание	Значения, необходимые для запуска одного экземпляра СУБД «Синергия-БД»
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
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 системы	как SHMSEG плюс потребность других приложений
SEMMNI	Максимальное число идентификаторов семафоров (т. е., их наборов)	как минимум ceil((max_connections + autovacuum_max_workers + max_worker_processes + 5) / 16) плюс потребность других приложений
SEMMNS	Максимальное число семафоров для всей системы	ceil((max_connections + autovacuum_max_workers + 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
SEMVMX	Максимальное значение семафора	не меньше 1000 (по умолчанию оно обычно равно 32767; без необходимости менять его не следует)

Синергия-БД запрашивает небольшой блок разделяемой памяти System V (обычно 48 байт на 64-битной платформе) для каждой копии сервера. В большинстве современных операционных систем такой объём выделяется без проблем. Однако, если запускать много копий сервера, или разделяемую память System V занимают и другие приложения, может понадобиться увеличить значение SHMMAX, максимальный размер сегмента разделяемой памяти (в байтах), либо SHMALL, общий объём разделяемой памяти System V, доступный для всей системы. Заметьте, что SHMALL во многих системах задаётся в страницах, а не в байтах.

Менее вероятны проблемы с минимальным размером сегментов разделяемой памяти (SHMMIN), который для Синергия-БД не должен превышать примерно 32 байт (обычно это всего 1 байт). Максимальное число сегментов для всей системы (SHMMNI) или для одного процесса (SHMSEG) тоже обычно не влияет на

работоспособность сервера, если только это число не равно нулю.

Синергия-БД использует по одному семафору на одно разрешённое подключение, на разрешённый рабочий процесс автоочистки и фоновый процесс, 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 и max_worker_processes, плюс один дополнительный на каждые 16 семафоров подключений и рабочих процессов (см. формулу в Таблице 2-1). Параметр SEMMNI определяет максимальное число наборов семафоров, которые могут существовать в системе в один момент времени. Таким образом, этот параметр должен быть не меньше ceil((max_connections + autovacuum_max_workers + max_worker_processes + 5) / 16). В качестве временного решения проблем, которые вызваны этими ограничениями, но обычно сопровождаются некорректными сообщениями, например, "No space left on device" (На устройстве не осталось места) от функции semget, можно уменьшить число разрешённых соединений.

В некоторых случаях может потребоваться увеличить SEMMAP как минимум до уровня SEMMNS. Этот параметр 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 времени привести к уменьшению числа доступных семафоров.

Параметр SEMMSL, определяющий, сколько семафоров может быть в одном наборе, для Синергия-БД должен равняться как минимум 17.

Другие параметры, связанные с «аннулированием операций» с семафорами, например, SEMMNU и SEMUME, на работу Синергия-БД не влияют.

По умолчанию максимальный размер сегмента равен 32 МБ, а максимальный общий размер составляет 2097152 страниц. Страница почти всегда содержит 4096 байт, за исключением нестандартных конфигураций ядра с поддержкой «огромных страниц». Точно узнать размер страницы можно, выполнив `getconf PAGE_SIZE`.

Параметры размера разделяемой памяти можно изменить, воспользовавшись командой `sysctl`. Например, так можно выделить 16 ГБ для разделяемой памяти:

```
$ sysctl -w kernel.shmmax=17179869184
$ sysctl -w kernel.shmall=4194304
```

Чтобы сохранить эти изменения после перезагрузки, их также можно записать в файл `/etc/sysctl.conf` (это настоятельно рекомендуется).

В некоторых старых дистрибутивах может не оказаться программы `sysctl`, но те же изменения можно произвести, обратившись к файловой системе `/proc`:

```
$ echo 17179869184 >/proc/sys/kernel/shmmax
$ echo 4194304 >/proc/sys/kernel/shmall
```

Остальные параметры имеют вполне подходящие значения, так что их обычно менять не нужно.

2.1.2. RemoveIPC в systemd

Если используется `systemd`, необходимо позаботиться о том, чтобы ресурсы IPC (общая память и семафоры) не освобождались преждевременно операционной системой. Это особенно актуально при сборке и установке Синергия-БД из исходного кода. Пользователей дистрибутивных пакетов Синергия-БД 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. Этот параметр по умолчанию включён в стандартной сборке `systemd`, но в некоторых дистрибутивах операционных систем он по умолчанию отключён.

Обычно нежелательный эффект этого включённого параметра проявляется в

том, что объекты семафоров, используемые сервером Синергия-БД, удаляются без видимых причин, что приводит к отказу сервера с сообщениями вида:

```
LOG: semctl(1234567890, 0, IPC_RMID, ...) failed:
Invalid argument
```

Различные типы объектов IPC (разделяемая память/семафоры, System V/POSIX) обрабатываются в `systemd` несколько по-разному, поэтому могут наблюдаться ситуации, когда некоторые ресурсы IPC не удаляются так, как другие. Однако полагаться на эти тонкие различия не рекомендуется.

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.

Какой пользователь является «системным», определяется во время компиляции `systemd`, исходя из значения `SYS_UID_MAX` в `/etc/login.defs`.

Скрипт упаковывания и развёртывания сервера должен предусмотрительно создавать пользователя `postgres` как системного пользователя, используя команды `useradd -r`, `adduser --system` или равнозначные.

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.

2.1.3. Ограничения ресурсов

В Unix-подобных операционных системах существуют различные типы ограничений ресурсов, которые могут влиять на работу сервера Синергия-БД. Особенно важны ограничения на число процессов для пользователя, число открытых файлов и 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). За подробностями обратитесь к документации операционной системы. Для Синергия-БД интерес представляют параметры `maxproc`, `openfiles` и `datasize`. Они могут задаваться, например, так:

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

здесь `-cur` обозначает мягкий предел. Чтобы задать жёсткий предел, нужно заменить это окончание на `-max`.

Ядро также может устанавливать общесистемные ограничения на использование некоторых ресурсов.

В Linux максимальное число открытых файлов, которое поддерживает ядро, определяется в спецфайле `/proc/sys/fs/file-max`. Изменить этот предел можно, записав другое число в этот файл, либо добавив присваивание в файл `/etc/sysctl.conf`. Максимальное число файлов для одного процесса задаётся при компиляции ядра; за дополнительными сведениями обратитесь к `/usr/src/linux/Documentation/proc.txt`.

Сервер Синергия-БД использует для обслуживания каждого подключения отдельный процесс, так что возможное число процессов должно быть не меньше числа разрешённых соединений плюс число процессов, требуемых для остальной системы. Это обычно не проблема, но, когда в одной системе работает множество серверов, предел может быть достигнут.

В качестве максимального числа открытых файлов по умолчанию обычно выбираются «социально-ориентированные» значения, позволяющие использовать одну систему нескольким пользователям так, чтобы ни один из них не потреблял

слишком много системных ресурсов. Если вы запускаете в системе несколько серверов, это должно вполне устраивать, но на выделенных машинах может возникнуть желание увеличить этот предел.

С другой стороны, некоторые системы позволяют отдельным процессам открывать очень много файлов и, если это делают сразу несколько процессов, они могут легко исчерпать общесистемный предел. Если вы столкнётесь с такой ситуацией, но не захотите менять общесистемное ограничение, вы можете ограничить использование открытых файлов сервером Синергия-БД, установив параметр конфигурации `max_files_per_process`.

2.1.4. Чрезмерное выделение памяти в Linux

В Linux 2.4 и новее механизм виртуальной памяти по умолчанию работает не оптимально для Синергия-БД. Вследствие того, что ядро выделяет память в чрезмерном объёме, оно может уничтожить главный управляющий процесс Синергия-БД (`postmaster`), если при выделении памяти процессу Синергия-БД или другому процессу виртуальная память будет исчерпана.

Когда это происходит, вы можете получить примерно такое сообщение ядра (где именно искать это сообщение, можно узнать в документации вашей системы):

```
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, Синергия-БД придётся перезапустить.

Один из способов обойти эту проблему - запускать Синергия-БД на компьютере, 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 Синергия-БД, эту проблему можно решить, изменив конфигурацию сервера. В некоторых случаях может помочь

уменьшение конфигурационных параметров, связанных с памятью, а именно shared_buffers и work_mem. В других случаях проблема может возникать, потому что разрешено слишком много подключений к самому серверу баз данных. Чаще всего в такой ситуации стоит уменьшить число подключений max_connections и организовать внешний пул соединений.

В Linux 2.6 и новее «чрезмерное выделение» памяти можно предотвратить, изменив поведение ядра. Хотя при этом OOM killer (уничтожение процессов при нехватке памяти) всё равно может вызываться, вероятность такого уничтожения значительно уменьшается, а значит поведение системы становится более стабильным. Для этого нужно включить режим строгого выделения памяти, воспользовавшись sysctl:

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

либо поместив соответствующую запись в /etc/sysctl.conf. Возможно, вы также захотите изменить связанный параметр vm.overcommit_ratio. За подробностями обратитесь к документации ядра Documentation/vm/overcommit-accounting.

Другой подход, который можно применить (возможно, вместе с изменением vm.overcommit_memory), заключается в исключении процесса postmaster из числа возможных жертв при нехватке памяти. Для этого нужно задать для свойства поправка очков OOM этого процесса значение -1000. Проще всего это можно сделать, выполнив:

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

в скрипте запуска управляющего процесса непосредственно перед тем, как запускать postmaster. Заметьте, что делать это надо под именем root, иначе ничего не изменится; поэтому проще всего вставить эту команду в стартовый скрипт, принадлежащий пользователю root. Если вы делаете это, вы также должны установить в данном скрипте эти 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 будут запускаться с обычной, нулевой поправкой очков OOM, так что при необходимости механизм OOM сможет уничтожать их. Вы можете задать и другое значение для PG_OOM_ADJUST_VALUE, если хотите, чтобы дочерние процессы исполнялись с другой поправкой OOM. PG_OOM_ADJUST_VALUE также можно опустить, в этом случае подразумевается нулевое значение. Если вы не установите PG_OOM_ADJUST_FILE, дочерние процессы будут работать с той же поправкой очков OOM, которая задана для главного процесса, что неразумно, так всё это делается как раз для того, чтобы главный процесс оказался на особом положении.

В старых ядрах Linux /proc/self/oom_score_adj отсутствует, но та же функциональность может быть доступна через /proc/self/oom_adj. Эта переменная процесса работает так же, только значение, исключающее уничтожение процесса, равно -17, а не -1000.

2.1.5. Огромные страницы в Linux

Использование огромных страниц снижает накладные расходы при работе с большими непрерывными блоками памяти, что характерно для Синергия-БД, особенно при больших значениях shared_buffers. Чтобы можно было использовать эту возможность в Синергия-БД, ядро должно быть собрано с параметрами CONFIG_HUGETLBFS=y и CONFIG_HUGETLB_PAGE=y. Также вам понадобится настроить параметр ядра vm.nr_hugepages. Чтобы оценить требуемое количество огромных страниц, запустите Синергия-БД без поддержки огромных страниц и посмотрите на показатель VmPeak процесса postmaster, а также узнайте размер огромной страницы, воспользовавшись файловой системой /proc. Например, вы можете получить:

```
$ head -1 $PGDATA/postmaster.pid
4170
$ grep ^VmPeak /proc/4170/status
VmPeak:  6490428 kB
$ grep ^Hugepagesize /proc/meminfo
```

```
Hugepagesize:       2048 kB
```

В данном случае 6490428 / 2048 даёт примерно 3169.154, так что нам потребуется минимум 3170 огромных страниц, и мы можем задать это значение так:

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

Большее значение стоит указать, если огромные страницы будут использоваться и другими программами в этой системе. Не забудьте 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. Перезагрузки должен быть свободен больший объём памяти для преобразования в огромные страницы. Чтобы проверить текущую ситуацию с размещением огромных страниц, выполните:

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

Также может потребоваться дать пользователю операционной системы, 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 Синергия-БД использует огромные страницы, когда считает это возможным, а в противном случае переходит к обычным страницам. Чтобы задействовать огромные страницы принудительно, можно установить для `huge_pages` значение on в `postgresql.conf`. Заметьте, что с таким значением Синергия-БД не сможет запуститься, если не получит достаточного количества огромных страниц.

Более подробно о механизме огромных страниц в Linux можно узнать в документации на ядро Linux.

3. НАСТРОЙКА СЕРВЕРА

3.1. Изменение параметров

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:

1) 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;

2) строка

Обычно строковое значение заключается в апострофы (при этом внутренние апострофы дублируются). Однако, если значение является простым числом или идентификатором, апострофы обычно можно опустить;

3) число (целое или с плавающей точкой)

Десятичная точка как разделитель целой и дробной части допускается только для параметров, принимающих числа с плавающей точкой. Разделители разрядов в записи числа не принимаются. Заключать в апострофы число не требуется;

4) 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 единица по умолчанию для параметра, которая указывается в `pg_settings.unit`. Для удобства параметры также можно задавать, указывая единицу измерения явно, например, задать `'120 ms'` для значения времени, при этом такое значение будет переведено в

основную единицу измерения параметра. Заметьте, что для этого значение должно записываться в виде строки (в апострофах). Имя единицы является регистронезависимым, а между ним и числом допускаются пробельные символы.

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

5) 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.

3.1.2. Определение параметров в файле конфигурации

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

```
# Это комментарий
log_connections = yes
log_destination = 'syslog'
search_path = '"$user", public'
shared_buffers = 128MB
```

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.

Параметры, установленные таким образом, задают значения по умолчанию для данного кластера. Эти значения будут действовать в активных сеансах, если не будут переопределены. В следующих разделах описывается, как их может переопределить администратор или пользователь.

Основной процесс сервера перечитывает файл конфигурации заново, получая сигнал SIGHUP; послать его проще всего можно, запустив pg_ctl reload в командной строке или вызвав SQL-функцию pg_reload_conf(). Основной процесс сервера передаёт этот сигнал всем остальным запущенным серверным процессам, так что существующие сеансы тоже получают новые значения (после того, как завершится выполнение текущей команды клиента). Также возможно послать этот сигнал напрямую одному из серверных процессов. 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 (но об этом сообщается в журнале).

В дополнение к postgresql.conf в каталоге данных Синергия-БД содержится файл postgresql.auto.conf, который имеет тот же формат, что и postgresql.conf, но не должен редактироваться вручную. Этот файл содержит параметры, задаваемые командой ALTER SYSTEM. Он считывается автоматически одновременно с postgresql.conf и заданные в нём параметры действуют таким же образом. Параметры в postgresql.auto.conf переопределяют те, что указаны в postgresql.conf.

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

3.1.3. Управление параметрами через SQL

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

- команда `ALTER DATABASE` позволяет переопределить глобальные параметры на уровне базы данных;
- команда `ALTER ROLE` позволяет переопределить для конкретного пользователя как глобальные, так и локальные для базы данных параметры.

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

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

- команда `SHOW` позволяет узнать текущее значение всех параметров. Соответствующая ей функция - `current_setting`(имя_параметра `text`);
- команда `SET` позволяет изменить текущее значение параметров, которые действуют локально в рамках сеанса; на другие сеансы она не влияет. Соответствующая ей функция - `set_config`(имя_параметра, новое_значение, локально).

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 присвоение значения столбцу, равносильно выполнению команды SET. Например, команде

```
SET configuration_parameter TO DEFAULT;
```

равнозначен запрос:

```
UPDATE pg_settings SET setting = reset_val WHERE name =
'configuration_parameter';
```

3.1.4. Управление параметрами в командной строке

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

1. 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 postgresql.conf или командой ALTER SYSTEM, так что их нельзя изменить глобально без перезапуска сервера.

2. 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, например:

```
env PGOPTIONS="-c geqo=off -c statement_timeout=5min"
psql
```

Другие клиенты и библиотеки могут иметь собственные механизмы управления параметрами, через командную строку или как-то иначе, используя

которые пользователь сможет менять параметры сеанса, не выполняя непосредственно команды SQL.

3.1.5. Упорядочение содержимого файлов конфигурации

Синергия-БД 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 присвоений значений параметров, `postgresql.conf` может содержать директивы включения файлов, которые будут прочитаны и обработаны, как если бы их содержимое было вставлено в данном месте файла конфигурации. Это позволяет разбивать файл конфигурации на физически отдельные части. Директивы включения записываются просто:

```
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 так:

```
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, т. е. цифры идут перед буквами, а буквы в верхнем регистре - перед буквами в нижнем.

Включение файлов или каталогов позволяет разделить конфигурацию базы данных на логические части, а не вести один большой файл `postgresql.conf`. Например, представьте, что в некоторой компании есть два сервера баз данных, с разным объёмом ОЗУ. Скорее всего при этом их конфигурации будут иметь общие элементы, например, параметры ведения журналов. Но параметры, связанные с памятью, у них будут различаться. Кроме того, другие параметры могут быть специфическими для каждого сервера. Один из вариантов эффективного управления такими конфигурациями - разделить изменения стандартной конфигурации на три файла. Чтобы подключить эти файлы, можно добавить в конец файла `postgresql.conf` следующие директивы:

```
include 'shared.conf'
include 'memory.conf'
include 'server.conf'
```

Общие для всех серверов параметры будут помещаться в `shared.conf`. Файл `memory.conf` может иметь два варианта - первый для серверов с 8ГБ ОЗУ, а второй для серверов с 16 ГБ. Наконец, `server.conf` может содержать действительно специфические параметры для каждого отдельного сервера.

Также возможно создать каталог с файлами конфигурации и поместить туда все эти файлы. Например, так можно подключить каталог `conf.d` в конце `postgresql.conf`:

```
include_dir 'conf.d'
```

Затем можно дать файлам в каталоге `conf.d` следующие имена:

```
00shared.conf
01memory.conf
02server.conf
```

Такое именование устанавливает чёткий порядок подключения этих файлов,

что важно, так как если параметр определяется несколько раз в разных файлах конфигурации, действовать будет последнее определение. В рамках данного примера, установленное в `conf.d/02server.conf` значение переопределит значение того же параметра, заданное в `conf.d/01memory.conf`.

Вы можете применить этот подход и с описательными именами файлов:

```
00shared.conf
01memory-8GB.conf
02server-foo.conf
```

При таком упорядочивании каждому варианту файла конфигурации даётся уникальное имя. Это помогает исключить конфликты, если конфигурации разных серверов нужно хранить в одном месте, например, в репозитории системы управления версиями. Кстати, хранение файлов конфигурации в системе управления версиями - это ещё один эффективный приём, который стоит применять.

3.2. Подключения

3.2.1. 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 доменные сокеты Unix. По умолчанию этот параметр содержит `localhost`, что допускает подключение к серверу по TCP/IP только через локальный интерфейс «замыкания». Хотя механизм аутентификации клиентов (см. п. 4) 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. Заметьте, что 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 сервера, работающего в режиме резерва, значение этого параметра должно быть больше или равно значению на главном. В противном случае на резервном сервере не будут разрешены запросы.

`superuser_reserved_connections (integer)`

Определяет количество «слотов» подключений, которые Синергия-БД 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 доменного сокета Unix, 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 доменными сокетами Unix, 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 не удаляйте эти файлы вручную.

unix_socket_group (string)

Задаёт группу-владельца доменных сокетов Unix. Пользователем-владельцем сокетов всегда будет пользователь, запускающий сервер. В сочетании с unix_socket_permissions данный параметр можно использовать как дополнительный механизм управления доступом к доменным 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 доменным сокетам Unix. Для доменных 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, см. также unix_socket_group) и 0700 (только пользователь). Заметьте, что для доменных сокетов требуется только право на запись, так что добавлять или отзывать права на чтение/выполнение не имеет смысла.

Этот механизм управления доступом не зависит от описанного в п. 4.

Этот параметр можно задать только при запуске сервера.

Данный параметр неприменим для некоторых систем, которые полностью игнорируют разрешения для сокетов. В таких системах примерно тот же эффект можно получить, указав в параметре unix_socket_directories каталог, доступ к которому ограничен должным образом.

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 неактивности (в секундах), после которого по TCP клиенту должен отправляться сигнал сохранения соединения. При значении 0 применяется системная величина. Этот параметр поддерживается только в системах, где определён символ TCP_KEEPIDLE или TCP_KEEPALIVE; в других системах он должен быть равен нулю. В сеансах, подключённых через доменные сокеты Unix, он игнорируется и всегда считается равным 0.

tcp_keepalives_interval (integer)

Задаёт интервал (в секундах), по истечении которого следует повторять сигнал сохранения соединения, если ответ от клиента не был получен. При значении 0 применяется системная величина. Этот параметр поддерживается только в системах, где определён символ TCP_KEEPINTVL; в других системах он должен быть равен нулю. В сеансах, подключённых через доменные сокеты Unix, 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 применяется системная величина. Этот параметр поддерживается

только в системах, где определён символ TCP_KEEPCNT; в других системах он должен быть равен нулю. В сеансах, подключённых через доменные сокеты Unix, он игнорируется и всегда считается равным 0.

3.3. Потребление ресурсов

3.3.1. Память

shared_buffers (integer)

Задаёт объём памяти, который будет использовать сервер баз данных для буферов в разделяемой памяти. По умолчанию это обычно 128 мегабайт (128MB), но может быть и меньше, если конфигурация вашего ядра накладывает дополнительные ограничения (это определяется в процессе initdb). Это значение не должно быть меньше 128 килобайт. Этот минимум зависит от величины BLCKSZ. Однако для хорошей производительности обычно требуются гораздо большие значения. Задать этот параметр можно только при запуске сервера.

Если вы используете выделенный сервер 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 Синергия-БД использует и кеш операционной системы, 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 серверах с объёмом ОЗУ меньше 1ГБ следует использовать меньший процент ОЗУ, чтобы оставить достаточно памяти для операционной системы.

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.

В настоящее время это поддерживается только в Linux. Во всех других системах значение try просто игнорируется.

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

Заметьте, что этот параметр влияет только на основную область общей памяти. Огромные страницы (также называемые «суперстраницами» или «большими» страницами) могут также автоматически использоваться при обычном выделении памяти, без явного запроса со стороны Синергия-БД. В Linux это называется «прозрачными огромными страницами» (Transparent Huge Pages, THP). Известно, что это приводит к снижению быстродействия Синергия-БД в некоторых системах Linux у ряда пользователей, поэтому использовать этот механизм в настоящее время не рекомендуется (в отличие от явного использования huge_pages).

temp_buffers (integer)

Задаёт максимальное число временных буферов для каждого сеанса. По умолчанию объём временных буферов составляет восемь мегабайт (1024 буфера). Этот параметр можно изменить в отдельном сеансе, но только до первого обращения к временным таблицам; после этого изменить его значение для текущего сеанса не удастся.

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 занимает около 64 байтов (в количестве temp_buffers). Однако если буфер действительно используется, он будет дополнительно занимать 8192 байта (или в общем случае, BLCKSZ байтов).

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 сервера, работающего в режиме резерва, значение этого параметра должно быть больше или равно значению на главном. В противном случае на резервном сервере не будут разрешены запросы.

work_mem (integer)

Задаёт объём памяти, который будет использоваться для внутренних операций сортировки и хеш-таблиц, прежде чем будут задействованы временные файлы на диске. Значение по умолчанию - четыре мегабайта (4MB). Заметьте, что в сложных запросах одновременно могут выполняться несколько операций сортировки или хеширования, так что этот объём памяти будет доступен для каждой операции. 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.

maintenance_work_mem (integer)

Задаёт максимальный объём памяти для операций обслуживания БД, в частности VACUUM, CREATE INDEX и ALTER TABLE ADD FOREIGN KEY. По умолчанию его значение - 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.

Учтите, 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. По умолчанию равен -1, что означает, что этот объём определяется значением maintenance_work_mem. Этот параметр не влияет на поведение команды VACUUM, выполняемой в других контекстах.

max_stack_depth (integer)

Задаёт максимальную безопасную глубину стека для исполнителя. В идеале это значение должно равняться предельному размеру стека, ограниченному ядром (как устанавливает команда ulimit -s или равнозначные ей), 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. Значение по умолчанию - два мегабайта (2MB), выбрано с большим запасом, так что риск переполнения стека минимален. Однако, с другой стороны, его может быть недостаточно для выполнения сложных функций. Изменить этот параметр могут только суперпользователи.

Если max_stack_depth будет превышать фактический предел ядра, то функция с неограниченной рекурсией сможет вызвать крах отдельного процесса сервера. В системах, где Синергия-БД может определить предел, установленный ядром, он не позволит установить для этого параметра небезопасное значение. Однако эту информацию выдают не все системы, поэтому выбирать это значение следует с осторожностью.

dynamic_shared_memory_type (enum)

Выбирает механизм динамической разделяемой памяти, который будет использовать сервер. Допустимые варианты: posix (для выделения разделяемой памяти POSIX функцией shm_open), sysv (для выделения разделяемой памяти System V функцией shmget), mmap (для эмуляции разделяемой памяти через

отображение в память файлов, хранящихся в каталоге данных) и `none` (для отключения этой функциональности). Не все варианты поддерживаются на разных платформах; первый из поддерживаемых данной платформой вариантов становится для неё вариантом по умолчанию. Применять `mmap`, который нигде не выбирается по умолчанию, вообще не рекомендуется, так как операционная система может периодически записывать на диск изменённые страницы, что создаст дополнительную нагрузку; однако, это может быть полезно для отладки, когда каталог `pg_dynshmem` находится в RAM-диске или, когда другие механизмы разделяемой памяти недоступны.

3.3.2. Диск

`temp_file_limit (integer)`

Задаёт максимальный объём дискового пространства, который сможет использовать один сеанс для временных файлов, например, при сортировке и хешировании, или для сохранения удерживаемого курсора. Транзакция, которая попытается превысить этот предел, будет отменена. Этот параметр задаётся в килобайтах, а 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 сеансе Синергия-БД. Следует отметить, что при этом учитывается только место, занимаемое явно создаваемыми временными таблицами; на временные файлы, которые создаются неявно при выполнении запроса, это ограничение не распространяется.

3.3.3. Использование ресурсов ядра

`max_files_per_process (integer)`

Задаёт максимальное число файлов, которые могут быть одновременно открыты каждым серверным подпроцессом. Значение по умолчанию - 1000 файлов. Если ядро реализует безопасное ограничение по процессам, об этом параметре можно не беспокоиться. Но на некоторых платформах ядро позволяет отдельному

процессу открыть больше файлов, чем могут открыть несколько процессов одновременно. Если вы столкнётесь с ошибками "Too many open files" (Слишком много открытых файлов), попробуйте уменьшить это число. Задать этот параметр можно только при запуске сервера.

3.3.4. Задержка 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 базой данных. 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 (integer)

Продолжительность времени, в миллисекундах, в течение которого будет простаивать процесс, превысивший предел стоимости. По умолчанию его значение равно нулю, то есть задержка очистки отсутствует. При положительных значениях интенсивность очистки будет зависеть от стоимости. Заметьте, что во многих системах разрешение таймера составляет 10 мс, поэтому если задать в vacuum_cost_delay значение, не кратное 10, фактически будет получен тот же результат, что и со следующим за ним кратным 10.

При настройке интенсивности очистки для `vacuum_cost_delay` обычно выбираются довольно небольшие значения, например, 10 или 20 миллисекунд. Чтобы точнее ограничить потребление ресурсов при очистке, лучше всего изменять другие параметры стоимости очистки.

`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.

3.3.5. Фоновая запись

В числе специальных процессов сервера есть процесс фоновой записи, задача которого - осуществлять запись «грязных» (новых или изменённых) общих буферов на диск. Он старается записывать данные из буферов так, чтобы обычным серверным процессам, обрабатывающим запросы, не приходилось ждать записи или это ожидание было минимальным. Однако процесс фоновой записи увеличивает общую нагрузку на подсистему ввода/вывода, так как он может записывать неоднократно изменяемую страницу при каждом изменении, тогда как она может

быть записана всего раз в контрольной точке. Параметры, рассматриваемые в данном подразделе, 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 количество загрязнённых буферов (это настраивается следующими параметрами). Затем он засыпает на время `bgwriter_delay` (задаваемое в миллисекундах), и всё повторяется снова. Однако если в пуле не остаётся загрязнённых буферов, он может быть неактивен более длительное время. По умолчанию этот параметр равен 200 миллисекундам (200ms). Заметьте, что во многих системах разрешение таймера составляет 10 мс, поэтому если задать в `bgwriter_delay` значение, не кратное 10, фактически будет 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.

`bgwriter_lru_maxpages (integer)`

Задаёт максимальное число буферов, которое сможет записать процесс фоновой записи за раунд активности. При нулевом значении фоновая запись отключается. Учтите, что на контрольные точки, которые управляются отдельным вспомогательным процессом, это не влияет. 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 чем bgwriter_flush_after байт, 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. Он может принимать значение от 0 (при этом управление отложенной записью отключается) до 2 Мбайт (2MB). Значение по умолчанию - 512kB в Linux и 0 в других ОС. Если BLCKSZ отличен от 8 Кбайт, 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.

3.3.6. Асинхронное поведение

effective_io_concurrency (integer)

Задаёт допустимое число параллельных операций ввода/вывода, которое говорит Синергия-БД о том, сколько операций ввода/вывода могут быть выполнены одновременно. Чем больше это число, тем больше операций ввода/вывода будет пытаться выполнить параллельно Синергия-БД 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.

Хорошим начальным значением этого параметра будет число отдельных дисков, составляющих массив RAID 0 или RAID 1, если база данных размещена в нём. Для RAID 5 следует исключить один диск (как диск с чётностью). Однако, если база данных часто обрабатывает множество запросов в различных сеансах, и при небольших значениях дисковый массив может быть полностью загружен. Если продолжать увеличивать это значение при полной загрузке дисков, это приведёт только к увеличению нагрузки на процессор.

Асинхронный ввод/вывод зависит от эффективности функции `posix_fadvise`, которая отсутствует в некоторых операционных системах. В случае её отсутствия попытка задать для этого параметра любое ненулевое значение приведёт к ошибке. В некоторых системах, эта функция присутствует, но на самом деле ничего не делает.

Значение по умолчанию равно 1 в системах, где это поддерживается, и 0 в остальных. Это значение можно переопределить для таблиц в определённом табличном пространстве, установив одноимённый параметр табличного пространства.

`max_worker_processes (integer)`

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

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

Одновременно с изменением этого значения также может быть полезно изменить max_parallel_workers, max_parallel_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. Учтите, что запрошенное количество рабочих процессов может быть недоступно во время выполнения. В этом случае план будет выполняться с меньшим числом процессов, что может быть неэффективно. Значение по умолчанию - 2. Значение 0 отключает параллельное выполнение запросов.

Учтите, что параллельные запросы могут потреблять значительно больше ресурсов, чем не параллельные, так как каждый рабочий процесс является отдельным процессом и оказывает на систему примерно такое же влияние, как дополнительный пользовательский сеанс. Это следует учитывать, выбирая значение этого параметра, а также настраивая другие параметры, управляющие использованием ресурсов, например, work_mem. Ограничения ресурсов, такие как work_mem, применяются к каждому рабочему процессу отдельно, что означает, что общая нагрузка для всех процессов может оказаться гораздо больше, чем при обычном использовании одного процесса. Например, параллельный запрос, задействующий 4 рабочих процесса, может использовать в 5 раз больше времени процессора, объёма памяти, ввода/вывода и т. д., по сравнению с запросом, не задействующим рабочие процессы вовсе.

max_parallel_maintenance_workers (integer)

Задаёт максимальное число рабочих процессов, которые могут запускаться одной служебной командой. В настоящее время параллельные процессы может использовать только одна служебная команда, CREATE INDEX, и только при построении индекса-B-дерева. Параллельные рабочие процессы берутся из пула процессов, контролируемого параметром max_worker_processes, в количестве,

ограничиваемом значением max_parallel_workers. Учтите, что запрошенное количество рабочих процессов может быть недоступно во время выполнения. В этом случае служебная операция будет выполняться с меньшим числом процессов, чем ожидалось. Значение по умолчанию - 2. Значение 0 отключает использование параллельных исполнителей служебными командами.

Заметьте, что параллельно выполняемые служебные команды не должны потреблять значительно больше памяти, чем равнозначные непараллельные операции. Это отличает их от параллельных запросов, при выполнении которых ограничения ресурсов действуют на отдельные рабочие процессы. Для параллельных служебных команд ограничение ресурсов maintenance_work_mem считается действующим на команду в целом, вне зависимости от числа параллельных рабочих процессов. Тем не менее, параллельные служебные команды могут гораздо больше нагружать процессор и каналы ввода/вывода.

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. Заметьте, что 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 backend_flush_after байт, 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. Он может принимать значение от 0 (при этом управление отложенной записью отключается) до 2 Мбайт (2MB). По умолчанию он имеет значение 0, то есть это поведение отключено. Если BLCKSZ отличен от 8 Кбайт, максимальное значение корректируется пропорционально.

old_snapshot_threshold (integer)

Задаёт минимальное время, которое можно пользоваться снимком без риска получить ошибку снимок слишком стар. Этот параметр можно задать только при запуске сервера.

По истечении этого времени старые данные могут 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 снимке, клиенту будет выдана ошибка, если возраст снимка превысит заданный предел и из этого снимка будет запрошена страница, изменённая со времени его создания.

Значение -1 (по умолчанию) отключает это поведение. Полезные значения для производственной среды могут лежать в интервале от нескольких часов до нескольких дней. Заданное значение округляется до минут, а минимальные значения (как например, 0 или 1min) допускаются только потому, что они могут быть полезны при тестировании. Хотя допустимым будет и значение 60d (60 дней), учтите, что при многих видах нагрузки критичное замусоривание базы или зацикливание идентификаторов транзакций может происходить в намного меньших временных отрезках.

Когда это ограничение действует, освобождённое пространство в конце отношения не может быть отдано операционной системе, так как при этом будет удалена информация, необходимая для выявления условия снимок слишком стар. Всё пространство, выделенное отношению, останется связанным с ним до тех пор, пока не будет освобождено явно (например, с помощью команды VACUUM FULL).

Установка этого параметра не гарантирует, что обозначенная ошибка будет

выдаваться при всех возможных обстоятельствах. На самом деле, если можно получить корректные результаты, например, из курсора, материализовавшего результирующий набор, ошибка не будет выдана, даже если нижележащие строки в целевой таблице были ликвидированы при очистке. Некоторые таблицы, например, системные каталоги, не могут быть безопасно очищены в сжатые сроки, так что на них этот параметр не распространяется. Для таких таблиц этот параметр не сокращает раздувание, но и не чреват ошибкой снимок слишком стар при сканировании.

3.4. Планирование запросов

3.4.1. Конфигурация методов планировщика

Эти параметры конфигурации дают возможность грубо 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. Улучшить качество планов, выбираемых планировщиком, можно и более подходящими способами, в частности, скорректировать константы стоимости (см. п. 3.7.2), выполнить `ANALYZE` вручную, изменить значение параметра конфигурации `default_statistics_target` и увеличить объём статистики, собираемой для отдельных столбцов, воспользовавшись командой `ALTER TABLE SET STATISTICS`.

`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_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 наборы дочерних секций должны быть одинаковыми. 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.).

3.4.2. Константы стоимости для планировщика

Переменные стоимости, описанные в данном разделе, 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. Но при желании можно использовать и другую шкалу, например, выразить в миллисекундах фактическое время выполнения запросов на конкретной машине.

`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.

`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.

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.

Произвольный доступ к механическому дисковому хранилищу обычно гораздо дороже последовательного доступа, более чем в четыре раза. Однако по умолчанию выбран небольшой коэффициент (4.0), в предположении, что большой объём данных при произвольном доступе, например, при чтении индекса, окажется в кеше. Таким образом, можно считать, что значение по умолчанию моделирует ситуацию, когда произвольный доступ в 40 раз медленнее последовательного, но 90% операций произвольного чтения удовлетворяются из кеша.

Если вы считаете, что для вашей рабочей нагрузки процент попаданий не достигает 90%, вы можете увеличить параметр random_page_cost, чтобы он больше соответствовал реальной стоимости произвольного чтения. И напротив, если ваши данные могут полностью поместиться в кеше, например, когда размер базы меньше общего объёма памяти сервера, может иметь смысл уменьшить random_page_cost. С хранилищем, у которого стоимость произвольного чтения ненамного выше последовательного, как например, у твердотельных накопителей, так же лучше выбрать меньшее значение random_page_cost.

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. 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. Заметьте, что при параллельном сканировании по индексу обычно не затрагивается весь индекс; здесь учитывается число страниц, которое по мнению планировщика будет затронуто при сканировании. 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. При установке этого параметра следует учитывать и объём разделяемых буферов Синергия-БД, и процент дискового кеша ядра, который будут занимать файлы данных Синергия-БД. Кроме того, следует принять во внимание ожидаемое число параллельных запросов к разным таблицам, так как общий размер будет разделяться между ними. Этот параметр не влияет на размер разделяемой памяти, выделяемой Синергия-БД, или размер резервируемого кеша ядра; он используется только для целей оценки стоимости. При этом система

не учитывает, что данные могут оставаться в дисковом кеше от запроса к запросу. Значение этого параметра по умолчанию - 4 гигабайта (4GB).

`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 дорогостоящие оптимизации. Значение по умолчанию - 500000.

3.4.3. 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.

`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. Заметьте, что конструкция FULL OUTER JOIN считается одним элементом списка FROM. 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.

Параметр geqo_effort сам по себе ничего не делает, он используется только для вычисления значений по умолчанию для других переменных, влияющих на поведение GEQO (они описаны ниже). При желании эти переменные можно просто установить вручную.

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.

3.4.4. Другие параметры планировщика

default_statistics_target (integer)

Устанавливает целевое ограничение 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 ограничений таблиц 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 секционированные таблицы и наследование.

Когда данный параметр разрешает это для таблицы, планировщик сравнивает

условия запроса с ограничениями CHECK данной таблицы и не сканирует её, если они оказываются несовместимыми, например:

```
CREATE TABLE parent(key integer, ...);
CREATE TABLE child1000(check (key between 1000 and
1999)) INHERITS(parent);
CREATE TABLE child2000(check (key between 2000 and
2999)) INHERITS(parent);
...
SELECT * FROM parent WHERE key = 2400;
```

Если включено исключение по ограничению, команда SELECT не будет сканировать таблицу child1000, в результате чего запрос выполнится быстрее.

В настоящее время исключение по ограничению разрешено по умолчанию только в условиях, возникающих при реализации секционированных таблиц. Включение этой возможности для всех таблиц влечёт дополнительные издержки на планирование, довольно заметные для простых запросов, но никакого выигрыша это не приносит. Если вы не применяете секционированные таблицы, лучше всего полностью отключить эту возможность.

```
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 восьми.

Если это значение сделать равным geqo_threshold или больше, при таком объединении запросов может включиться планировщик GEQO и в результате будет получен неоптимальный план.

jit (boolean)

Определяет, может ли Синергия-БД использовать компиляцию JIT, если она поддерживается. По умолчанию он выключен (off).

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.

Если это значение сделать равным geqo_threshold или больше, при таком объединении запросов может включиться планировщик GEQO и в результате будет получен неоптимальный план.

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).

Говоря точнее, 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`).

3.5. Регистрация ошибок и протоколирование работы сервера

3.5.1. Куда протоколировать

`log_destination (string)`

Синергия-БД 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. Например, он может иметь такое содержание:

```
stderr log/postgresql.log
csvlog log/postgresql.csv
```

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 коллектор сообщений (logging collector). Это фоновый процесс, который собирает отправленные в stderr сообщения и перенаправляет их в журнальные файлы. Такой подход зачастую более полезен чем запись в syslog, поскольку некоторые сообщения в syslog могут не попасть. 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 журнальные файлы. Можно задавать как абсолютный путь, так и относительный от каталога данных кластера. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера. Значение по умолчанию pg_log.

log_filename (string)

При включённом logging_collector задаёт имена журнальных файлов. Значение трактуется как строка формата в функции strftime, поэтому в ней можно использовать спецификаторы % для включения в имена файлов информации о дате и времени. При наличии зависящих от часового пояса спецификаторов % будет использован пояс, заданный в log_timezone. Поддерживаемые спецификаторы % похожи на те, что перечислены в описании strftime спецификации Open Group. Обратите внимание, что системная функция strftime напрямую не используется. Поэтому нестандартные, специфичные для платформы особенности не будут работать. Значение по умолчанию postgresql-%Y-%m-%d_%H%M%S.log.

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

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

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

log_rotation_age (integer)

Определяет максимальное время жизни отдельного журнального файла, при включённом logging_collector. После того как прошло заданное количество минут, создаётся новый журнальный файл. Для запрета создания нового файла по прошествии определённого времени, нужно установить значение 0. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске

сервера.

log_rotation_size (integer)

Определяет максимальный размер отдельного журнального файла, при включённом logging_collector. После того как заданное количество килобайт записано в текущий файл, создаётся новый журнальный файл. Для запрета создания нового файла при превышении определённого размера, нужно установить значение 0. 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, Синергия-БД будет перезаписывать существующие журнальные файлы, а не дописывать в них. Однако, перезапись при переключении на новый файл возможна только в результате ротации по времени, но не при старте сервера или ротации по размеру файла. При выключенном параметре всегда продолжается запись в существующий файл. Например, включение этого параметра в комбинации с log_filename равным postgresql-%H.log, приведёт к генерации 24-х часовых журнальных файлов, которые циклически перезаписываются. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера.

3.5.2. Когда протоколировать

log_min_messages (enum)

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

log_min_error_statement (enum)

Управляет тем, какие SQL-операторы, завершившиеся ошибкой, записываются в журнал сервера. SQL-оператор будет записан в журнал, если он завершится ошибкой с указанным уровнем важности или выше. Допустимые значения: DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL и PANIC. По умолчанию используется ERROR. Это означает, что в журнал сервера будут записаны все операторы, завершившиеся сообщением с уровнем важности ERROR, LOG, FATAL и PANIC. Чтобы фактически отключить запись операторов с ошибками, установите для этого параметра значение PANIC. Изменить этот параметр могут только суперпользователи.

log_min_duration_statement (integer)

Записывает в журнал продолжительность выполнения всех команд, время работы которых равно или превышает указанное количество миллисекунд. Значение 0 (ноль) заставляет записывать продолжительность работы всех команд. Значение -1 (по умолчанию) запрещает регистрировать продолжительность выполнения операторов. Например, при значении 250ms, все команды, которые выполняются за 250 миллисекунд и дольше будут записаны в журнал сервера. Включение параметра полезно для выявления плохо оптимизированных запросов в приложении. Только суперпользователи могут изменить этот параметр.

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

В таблице 3.1 поясняются уровни важности сообщений в СУБД «Синергия-БД».

Таблица 3.1 – Уровни важности сообщений

Уровень	Использование
DEBUG1.. DEBUG5	Более детальная информация для разработчиков. Чем больше номер, тем детальнее.
INFO	Неявно запрошенная пользователем информация, например вывод команды VACUUM VERBOSE.
NOTICE	Информация, которая может быть полезной пользователям. Например, уведомления об усечении длинных идентификаторов.
WARNING	Предупреждения о возможных проблемах. Например, COMMIT вне транзакционного блока.
ERROR	Сообщает об ошибке, из-за которой прервана текущая команда.

Уровень	Использование
LOG	Информация, полезная для администраторов. Например, выполнение контрольных точек.
FATAL	Сообщает об ошибке, из-за которой прервана текущая сессия.
PANIC	Сообщает об ошибке, из-за которой прерваны все сессии.

3.5.3. Что протоколировать

application_name (string)

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

debug_print_parse (boolean)

debug_print_rewritten (boolean)

debug_print_plan (boolean)

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

debug_pretty_print (boolean)

Включает выравнивание сообщений, выводимых debug_print_parse, debug_print_rewritten или debug_print_plan. В результате сообщения легче читать, но они значительно длиннее, чем в формате "compact", который используется при выключенном значении. По умолчанию включён.

log_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` (Приложение 1), а также имя файла с исходным кодом, имя функции и номер строки, сгенерировавшей ошибку. Только суперпользователи могут изменить этот параметр.

`log_hostname (boolean)`

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

`log_line_prefix (string)`

Строка, в стиле функции `printf`, которая выводится в начале каждой строки журнала сообщений. С символов `%` начинаются управляющие последовательности (таблица 3.2), которые заменяются статусной информацией, описанной ниже. Неизвестные управляющие последовательности игнорируются. Все остальные символы напрямую копируются в журнальную строку. Некоторые управляющие последовательности используются только для пользовательских процессов и будут игнорироваться фоновыми процессами, например, основным процессом сервера. Подставляемая информация может быть выровнена по ширине влево или вправо указанием числа после `%` и перед кодом последовательности. Отрицательное число дополняет значение пробелами справа, а положительное число дополняет пробелами слева. Выравнивание можно использовать для улучшения читаемости. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера. По умолчанию используется пустая строка.

Таблица 3.2 – Управлящие последовательности

Спецсимвол	**Назначение**	**Только для пользовательского процесса**
%a	Имя приложения (application_name)	да
%u	Имя пользователя	да
%d	Имя базы данных	да
%r	Имя удалённого узла или IP-адрес, а также номер порта	да
%h	Имя удалённого узла или IP-адрес	да
%p	Идентификатор процесса	нет
%t	Штамп времени, без миллисекунд	нет
%m	Штамп времени, с миллисекундами	нет
%n	Штамп времени, с миллисекундами (в виде	нет

Спецсимвол	Назначение	Только для пользовательского процесса
	времени Unix)	
%i	Тег команды: тип текущей команды в сессии	да
%e	Код ошибки SQLSTATE	нет
%c	Идентификатор сессии. Подробности ниже	нет
%l	Номер строки журнала для каждой сессии или процесса. Начинается с 1	нет
%s	Штамп времени начала процесса	нет
%v	Идентификатор виртуальной транзакции (backendID/localXID)	нет
%x	Идентификатор транзакции (0 если не присвоен)	нет
%q	Ничего не выводит. Непользовательские процессы останавливаются в этой точке. Игнорируется пользовательскими процессами	нет
%%	Выводит %	нет

`%c` выводит псевдоуникальный номер сессии, состоящий из двух 4-х битных шестнадцатеричных чисел (без лидирующих нулей), разделённых точкой. Эти числа представляют собой время старта процесса и идентификатор процесса, поэтому `%c` можно использовать для экономии места при записи этих значений. Например, для получения идентификатора сессии из `pg_stat_activity`, используйте запрос:

```
SELECT to_hex(trunc(EXTRACT(EPOCH FROM
backend_start))::integer) || '.' ||
       to_hex(pid)
FROM pg_stat_activity;
```

`log_lock_waits (boolean)`

Нужно ли фиксировать в журнале события, когда сессия ожидает получения блокировки больше чем указано в `deadlock_timeout`. По умолчанию выключено.

`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 удваиваются).

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 включением в журнал информации об именах и размерах временных файлов. Временные файлы могут использоваться для сортировок, хеширования и временного хранения результатов запросов. На каждый временный файл, при его удалении, в журнал записывается отдельное сообщение. Значение 0 говорит о том, что нужно записывать информацию о всех временных файлах. Положительное значение задаёт размер временных файлов в килобайтах, при достижении или превышении которого, информация о временном файле будет записана. Значение по умолчанию -1, что отключает запись информации о временных файлах. Только суперпользователи могут изменить этот параметр.

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.

3.6. Статистика времени выполнения

3.6.1. Сбор статистики по запросам и индексам

Эти параметры управляет функциями сбора статистики на уровне сервера. Когда ведётся сбор статистики, собираемые данные можно просмотреть в семействе системных представлений `pg_stat` и `pg_statio`. За дополнительными сведениями обратитесь к п. 4 (Часть 2. Руководство системного программиста).

`track_activities (boolean)`

Включает сбор сведений о текущих командах, выполняющихся во всех сеансах (в частности, отслеживается время запуска команды). По умолчанию этот параметр включён. Заметьте, что даже когда сбор ведётся, собранная информация доступна не для всех пользователей, а только для суперпользователей и пользователя-владельца сеанса, в котором выполняется текущая команда. Поэтому это не должно повлечь риски, связанные с безопасностью. Изменить этот параметр могут только суперпользователи.

`track_activity_query_size (integer)`

Задаёт число байт, которое будет зарезервировано для отслеживания выполняемой в данной момент команды в каждом активном сеансе, для поля `pg_stat_activity.query`. Значение по умолчанию - 1024. Задать этот параметр можно только при запуске сервера.

`track_counts (boolean)`

Включает сбор статистики активности в базе данных. 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 операционной системы, что может значительно замедлить работу на некоторых платформах. Для оценивания издержек замера времени на вашей платформе можно воспользоваться утилитой `pg_test_timing`. Статистику

ввода/вывода можно получить через представление pg_stat_database, в выводе EXPLAIN (когда используется параметр TIMING) и через представление pg_stat_statements. 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, а 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. Если разместить целевой каталог в файловой системе в ОЗУ, это снизит нагрузку на физическое дисковое хранилище и может увеличить быстродействие. Задать этот параметр можно только в postgresql.conf или в командной строке при запуске сервера.

3.6.2. Мониторинг статистики

log_statement_stats (boolean)
log_parser_stats (boolean)
log_planner_stats (boolean)
log_executor_stats (boolean)

Эти параметры включают вывод статистики по производительности соответствующего модуля в протокол работы сервера. Это грубый инструмент профилирования, похожий на функцию getrusage() в операционной системе. 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.

3.7. Автоматическая очистка

Эти параметры управляют поведением механизма автоочистки. За дополнительными сведениями обратитесь к п. 8.1.6. Заметьте, что многие из этих параметров могут быть переопределены на уровне таблиц.

`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.

Заметьте, что даже если этот параметр отключён, система будет запускать процессы автоочистки, когда это необходимо для предотвращения наложений идентификаторов транзакций. За дополнительными сведениями обратитесь к п. 8.1.5.

`log_autovacuum_min_duration (integer)`

Задаёт время (в миллисекундах) выполнения действия автоочистки, при превышении которого информация об этом действии записывается в протокол. При нулевом значении в протоколе фиксируются все действия автоочистки. Значение -1 (по умолчанию) отключает протоколирование действий автоочистки. Например, если задать значение `250ms`, в протоколе будут фиксироваться все операции автоматической очистки и анализа, выполняемые дольше 250 мс. Кроме того, когда этот параметр имеет любое значение, отличное от -1, в протокол будет записываться сообщение в случае пропуска действия автоочистки из-за конфликтующей блокировки. 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 базы данных. Демон автоочистки проверяет базу данных через заданный интервал времени и выдаёт команды VACUUM и ANALYZE, когда это требуется для таблиц этой базы. Задержка задаётся в секундах и по умолчанию 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_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_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 наложений идентификаторов транзакций в этой таблице. Заметьте, что система запустит процессы автоочистки для предотвращения наложений, даже если для всех других целей автоочистка отключена.

При очистке могут также удаляться старые файлы из подкаталога pg_clog, 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. За дополнительными сведениями обратитесь к п. 8.1.5.

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 наложений идентификаторов мультитранзакций в этой таблице. Заметьте, что система запустит процессы автоочистки для предотвращения наложений, 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. За дополнительными сведениями обратитесь к п. 8.1.5.1.

`autovacuum_vacuum_cost_delay (integer)`

Задаёт задержку при превышении предела стоимости, которая будет применяться при автоматических операциях `VACUUM`. При значении -1 применяется обычная задержка `vacuum_cost_delay`. Значение по умолчанию - 20 миллисекунд. 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`. Заметьте, что 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.

`special_blocking_policy (boolean)`

Включает/отключает автоматическую блокировку учётных записей пользователей при превышении пределов, задаваемых параметрами `special_blocking_counts_wrong` и `special_blocking_idle_interval`.

По умолчанию: `false`.

`special_blocking_counts_wrong (interval)`

Задаёт максимальное число неудачных попыток входа для отдельного пользователя базы данных во временном интервале, определённом параметром `special_blocking_interval`. Если пользователь базы данных превышает это ограничение, он автоматически блокируется и может быть разблокирован только администратором базы данных.

Этот параметр действует, только если включён special_blocking_policy.

По умолчанию: 1.

special_blocking_interval (interval)

Задаёт временной интервал, по истечении которого число неудачных попыток подключения к серверу сбрасывается. Этот интервал отсчитывается от первой неудачной попытки. Число неудачных попыток, допускаемых для каждого отдельного пользователя базы данных, задаётся параметром special_blocking_counts_wrong.

Этот параметр действует, только если включён special_blocking_policy.

По умолчанию: 10min.

special_blocking_idle_interval (interval)

Задаёт максимальный временной интервал после последнего успешного входа. Если пользователь базы данных не подключается к серверу в течение этого времени, данный пользователь автоматически блокируется и может быть разблокирован только администратором базы данных.

Этот параметр действует, только если включён special_blocking_policy.

По умолчанию: 30d (30 дней).

3.8. Параметры клиентских сеансов по умолчанию

3.8.1. Поведение команд

client_min_messages (enum)

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

позиция LOG здесь отличается от принятой в log_min_messages.

Сообщения уровня INFO передаются клиенту всегда.

search_path (string)

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

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

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

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

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

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

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

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_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 базе данных. Если табличное пространство с заданным именем не существует, Синергия-БД 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; для них задействуется temp_tablespaces.

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

За дополнительными сведениями о табличных пространствах обратитесь к п. 6.6.

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 этот список содержит больше одного имени, Синергия-БД выбирает из этого списка случайный элемент при создании каждого временного объекта; однако при создании последующих объектов внутри транзакции табличные пространства перебираются последовательно. Если в этом списке оказывается пустая строка, Синергия-БД будет автоматически использовать вместо этого элемента табличное пространство по умолчанию для текущей базы данных.

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 базы данных.

`check_function_bodies (boolean)`

Этот параметр обычно включён. Выключение этого параметра (присвоение 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.).

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.

Предполагается, что системы логической репликации будут устанавливать для него значение replica, применяя реплицированные изменения. В результате триггеры и правила (с конфигурацией по умолчанию) не будут срабатывать в репликах. Подробнее об этом говорится в описании предложений ENABLE TRIGGER и ENABLE RULE команды ALTER TABLE.

Внутри Синергия-БД значения origin и local воспринимаются как равнозначные. Сторонние системы репликации могут различать их для своих внутренних целей, например, отмечать значением local сеанс, изменения в котором не должны реплицироваться.

Так как внешние ключи реализованы посредством триггеров, присвоение этому параметру значения replica влечёт отключение всех проверок внешних ключей, что может привести к нарушению согласованности данных при некорректном использовании.

statement_timeout (integer)

Задаёт максимальное время выполнения оператора (в миллисекундах), начиная с момента получения сервером команды от клиента, по истечении которого оператор прерывается. Если log_min_error_statement имеет значение ERROR или ниже, оператор, прерванный по тайм-ауту, будет также записан в журнал. 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 (в миллисекундах) любым оператором получения блокировки таблицы, индекса, строки или другого объекта базы данных. 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 log_min_error_statement имеет значение ERROR или ниже, оператор, прерванный по тайм-ауту, будет также записан в журнал. При значении, равном нулю (по умолчанию), этот контроль длительности отключается.

В отличие от statement_timeout, этот тайм-аут может произойти только при ожидании блокировки. Заметьте, что при ненулевом statement_timeout бессмысленно задавать в lock_timeout такое же или большее значение, так как тайм-аут оператора всегда будет происходить раньше.

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

idle_session_timeout (integer)

Завершать любые сеансы, которые простаивают дольше заданного (в миллисекундах) времени.

Значение по умолчанию (0) отключает это поведение.

idle_in_transaction_session_timeout (integer)

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

Значение по умолчанию (0) отключает это поведение.

vacuum_freeze_table_age (integer)

Задаёт максимальный возраст для поля pg_class.relfrozenxid таблицы, при достижении которого команда VACUUM будет сканировать всю таблицу.

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 в целях предотвращения наложений. За дополнительными сведениями обратитесь к п. 8.1.5.

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. За дополнительными сведениями обратитесь к п. 8.1.5.

vacuum_multixact_freeze_table_age (integer)

Задаёт максимальный возраст для поля pg_class.relminmxid таблицы, при достижении которого команда VACUUM будет выполнять полное сканирование таблицы. 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 в целях предотвращения наложений. За дополнительными сведениями обратитесь к п. 8.1.5.1.

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. Хотя пользователи могут задать любое значение от нуля до одного миллиарда, в VACUUM введён внутренний предел для действующего значения, равный половине autovacuum_multixact_freeze_max_age, чтобы принудительная автоочистка не выполнялась слишком часто.

vacuum_cleanup_index_scale_factor (floating point)

Определяет процент от общего числа кортежей кучи, подсчитанных при последнем сборе статистики, который может быть вставлен без необходимости сканирования индекса на стадии очистки при выполнении VACUUM. В настоящее время этот параметр применяется только к индексам-B-деревьям.

Если из кучи не удалялись кортежи, индексы-B-деревья будут в любом случае сканироваться на стадии очистки VACUUM, когда выполняется хотя бы одно из следующих условий: статистика индексов устарела; индекс содержит удалённые страницы, которые могут быть переработаны при очистке. Статистика индекса считается устаревшей, если число недавно вставленных кортежей превышает процент vacuum_cleanup_index_scale_factor от общего числа кортежей в куче, полученного при предыдущем сборе статистики. Общее число кортежей в куче хранится в метастранице индекса. Заметьте, что эти данные появятся в ней только после того, как процедура VACUUM не обнаружит ни одного «мёртвого» кортежа, поэтому сканирование индекса-B-дерева на стадии очистки может быть пропущено, только если «мёртвые» кортежи не будут найдены на втором и последующих циклах VACUUM.

Параметр может принимать значения от 0 до 10000000000. Когда vacuum_cleanup_index_scale_factor равен 0, сканирование индекса на этапе очистки VACUUM не пропускается никогда. Значение по умолчанию - 0.1.

bytea_output (enum)

Задаёт выходной формат для значения типа bytea. Это может быть формат hex (по умолчанию) или escape (традиционный формат Синергия-БД). Входные значения bytea воспринимаются в обоих форматах, независимо от данного

параметра.

xmlbinary (enum)

Задаёт способ кодирования двоичных данных в XML. Это кодирование применяется, например, когда значения bytea преобразуются в XML функциями xmlelement или xmlforest. Допустимые варианты, определённые в стандарте XML-схем: base64 и hex. Значение по умолчанию - base64.

Конечный выбор в основном дело вкуса, ограничения могут накладываться только клиентскими приложениями. Оба метода поддерживают все возможные значения, хотя результат кодирования в base64 немного компактнее шестнадцатеричного вида (hex).

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).

Согласно стандарту SQL, этот параметр должен задаваться командой:

```
SET XML OPTION { DOCUMENT | CONTENT };
```

Этот синтаксис тоже поддерживается в Синергия-БД.

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 GIN и очередь очищается. Размер по умолчанию - четыре мегабайта (4MB). Этот предел можно переопределить для отдельных индексов GIN, изменив их параметры хранения.

3.9. Управление блокировками

deadlock_timeout (integer)

Время ожидания блокировки (в миллисекундах), по истечении которого будет выполняться проверка состояния взаимоблокировки. Эта проверка довольно дорогостоящая, поэтому сервер не выполняет её при всяком ожидании блокировки.

Мы оптимистично полагаем, что взаимоблокировки редки в производственных приложениях и поэтому просто ждём некоторое время, прежде чем пытаться выявить взаимоблокировку. При увеличении значения этого параметра сокращается время, уходящее на ненужные проверки взаимоблокировки, но замедляется реакция на реальные взаимоблокировки. 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. Если вы пытаетесь исследовать задержки, вызванные блокировками, имеет смысл уменьшить его по сравнению с обычным значением deadlock_timeout.

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. Заметьте, что это не число строк, которое может быть заблокировано; их количество не ограничено. Значение по умолчанию, 64, как показала практика, вполне приемлемо, но может возникнуть потребность его увеличить, если запросы обращаются ко множеству различных таблиц в одной транзакции, как например, запрос к родительской таблице со многими потомками. Этот параметр можно задать только при запуске сервера.

Для сервера, работающего в режиме резерва, значение этого параметра должно

быть больше или равно значению на главном. В противном случае на резервном 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 одним момент времени может быть заблокировано не больше этого число 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. Заметьте, что это не число строк, которое может быть заблокировано; их количество не ограничено. Значение по умолчанию, 64, как показала практика, вполне приемлемо, но может возникнуть потребность его увеличить, если запросы обращаются ко множеству различных таблиц в одной сериализуемой транзакции, как например, запрос к родительской таблице со многими потомками. 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.

3.10. Обработка 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), Синергия-БД будет автоматически перезагружаться после сбоя серверного процесса. Такой вариант позволяет обеспечить максимальную степень доступности базы данных. Однако в некоторых обстоятельствах, например, когда Синергия-БД управляется кластерным ПО, такую перезагрузку лучше отключить, чтобы кластерное ПО могло вмешаться и выполнить, возможно, более подходящие действия.

`data_sync_retry (boolean)`

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

В некоторых операционных системах 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, Синергия-БД 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.

3.11. Предопределённые параметры

Следующие «параметры» доступны только для чтения, их значения задаются при компиляции или при установке Синергия-БД. По этой причине они исключены из примера файла `postgresql.conf`. Эти параметры сообщают различные аспекты поведения Синергия-БД, которые могут быть интересны для определённых приложений, например, средств администрирования.

`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 (например, `shared_buffers`).

`data_checksums (boolean)`

Сообщает, включён ли в этом кластере контроль целостности данных. За дополнительными сведениями обратитесь к `data checksums`.

`data_directory_mode (integer)`

В Unix-системах этот параметр показывает разрешения для каталога данных (`data_directory`), определённые при запуске. За дополнительными сведениями обратитесь к `group access`.

`debug_assertions (boolean)`

Сообщает, был ли Синергия-БД собран с проверочными утверждениями. Это имеет место, когда при сборке Синергия-БД определён макрос `USE_ASSERT_CHECKING` (для этого нужно передать configure флаг `--enable-cassert`). По умолчанию Синергия-БД собирается без проверочных утверждений.

`integer_datetimes (boolean)`

Сообщает, был ли Синергия-БД собран с поддержкой даты и времени в 64-битных целых. Начиная с СУБД «Синергия-БД» версии 11, он всегда равен `on`.

`lc_collate (string)`

Сообщает локаль, по правилам которой выполняется сортировка текстовых данных. За дополнительными сведениями обратитесь к п. 7.1. Это значение определяется при создании базы данных.

lc_ctype (string)

Сообщает локаль, определяющую классификацию символов. За дополнительными сведениями обратитесь к п. 7.1. Это значение определяется при создании базы данных. Обычно оно не отличается от lc_collate, но для некоторых приложений оно может быть определено по-другому.

max_function_args (integer)

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

max_identifier_length (integer)

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

max_index_keys (integer)

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

segment_size (integer)

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

server_encoding (string)

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

server_version (string)

Сообщает номер версии сервера. Она определяется константой PG_VERSION

при сборке сервера.

`server_version_num (integer)`

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

`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 число блоков (страниц) в файле сегмента WAL. Общий размер файла сегмента WAL равняется произведению `wal_segment_size` и `wal_block_size`; по умолчанию это 16 мегабайт. За дополнительными сведениями обратитесь к п. 6.4 (Часть 2. Руководство системного программиста).

4. АУТЕНТИФИКАЦИЯ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

При подключении к серверу базы данных, клиентское приложение указывает имя пользователя Синергия-БД, так же, как и при обычном входе пользователя на компьютер с ОС Unix. При работе в среде SQL по имени пользователя определяется, какие у него есть права доступа к объектам базы данных (подробнее это описывается в п. 5). Следовательно, важно указать на этом этапе, к каким базам пользователь имеет право подключиться.

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

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

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

4.1. Файл pg_hba.conf

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

каталога с данными. Однако его можно разместить в любом другом месте; см. конфигурационный параметр `hba_file`.

Обычный 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 табуляцией. В полях могут быть использованы пробелы, если они взяты в кавычки. Если в кавычки берётся какое-либо зарезервированное слово в поле базы данных, пользователя или адресации (например, `all` или `replication`), то слово теряет своё особое значение и просто обозначает базу данных, пользователя или сервер с данным именем.

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

Запись может быть сделана в одном из семи форматов:

```
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  [параметры-аутентификации]
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 описаны ниже:

1) local

Управляет подключениями через доменные сокеты Unix. Без подобной записи подключения через доменные сокеты Unix невозможны;

2) host

Управляет подключениями, устанавливаемыми по TCP/IP.

3) hostssl

Управляет подключениями, устанавливаемыми по TCP/IP.

4) hostnossl

Этот тип записей противоположен `hostssl`, ему соответствуют только подключения по TCP/IP;

5) база

Определяет, каким именам баз данных соответствует эта запись. Значение `all` определяет, что подходят все базы данных. Значение `sameuser` определяет, что данная запись соответствует только, если имя запрашиваемой базы данных совпадает с именем запрашиваемого пользователя. Значение `samerole` определяет, что запрашиваемый пользователь должен быть членом роли с таким же именем, как и у запрашиваемой базы данных; `samegroup` - это устаревший, но допустимый вариант значения `samerole`. Суперпользователи не становятся членами роли автоматически из-за `samerole`, а только если они являются явными членами роли, прямо или косвенно, и не только из-за того, что они суперпользователи. Значение `replication` показывает, что запись соответствует, если запрашивается подключение репликации (имейте в виду, что подключения репликации не определяют какую-то конкретную базу данных). В противном случае, это имя определённой базы данных Синергия-БД. Несколько имён баз данных можно указать, разделяя их запятыми. Файл,

содержащий имена баз данных, можно указать, поставив знак `@` в начале его имени;

6) пользователь

Указывает, какому имени (или именам) пользователя базы данных соответствует эта запись. Значение `all` показывает, что это подходит всем пользователям. В противном случае, это либо имя конкретного пользователя базы данных, или имя группы, в начале которого стоит знак +. В Синергия-БД 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 подходящей только для этой конкретной роли. В связи с этим, суперпользователь рассматривается как член роли, только если он явно является членом этой роли, прямо или косвенно, а не только потому, что он является суперпользователем. Несколько имён пользователей можно указать, разделяя их запятыми. Файл, содержащий имена пользователей, можно указать, поставив знак `@` в начале его имени;

7) адрес

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

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

Типичные примеры диапазонов адресов IPv4, указанных таким образом: `172.20.143.89/32` для одного компьютера, `172.20.143.0/24` для небольшой и `10.6.0.0/16` для крупной сети. Диапазон адресов IPv6 может выглядеть как `::1/128` для одного компьютера (это адрес замыкания IPv6)

или как `fe80::7a31:c1ff:0000:0000/96` для небольшой сети. `0.0.0.0/0` представляет все адреса IPv4, а `::0/0` - все адреса IPv6. Чтобы указать один компьютер, используйте длину маски 32 для IPv4 или 128 для IPv6. Опускать замыкающие нули в сетевом адресе нельзя.

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

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

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

Указание имени, начинающееся с точки (`.`), соответствует суффиксу актуального имени узла. Так, `.example.com` будет соответствовать `foo.example.com` (а не только `example.com`).

Когда в `pg_hba.conf` указываются имена узлов, следует добиться, чтобы

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

Это поле применимо только к записям `host`, `hostssl` и `hostnossl`.

Также обратный запрос необходим для того, чтобы реализовать возможность соответствия суффиксов, поскольку для сопоставления с шаблоном требуется знать фактическое имя компьютера клиента.

Обратите внимание, что такое поведение согласуется с другими популярными реализациями контроля доступа на основе имён, такими как Apache HTTP Server и TCP Wrappers;

8) IP-адрес

IP-маска

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

Эти поля применимы только к записям `host`, `hostssl` и `hostnossl`;

9) метод-аутентификации

Указывает метод аутентификации, когда подключение соответствует этой записи. Варианты выбора приводятся ниже:

- trust

Разрешает безусловное подключение. Этот метод позволяет тому, кто может подключиться к серверу с базой данных Синергия-БД, войти под любым желаемым пользователем Синергия-БД без введения пароля и без какой-либо другой аутентификации. За подробностями обратитесь к п. 4.3.1;

- reject

Отклоняет подключение безусловно. Эта возможность полезна для

«фильтрации» некоторых серверов группы, например, строка `reject` может отклонить попытку подключения одного компьютера, при этом следующая строка позволяет подключиться остальным компьютерам в той же сети;

- md5

Требует от клиента предоставить для аутентификации пароль, дважды хешированный алгоритмом MD5;

- password

Требует для аутентификации введения клиентом незашифрованного пароля. Поскольку пароль посылается простым текстом через сеть, такой способ не стоит использовать, если сеть не вызывает доверия;

- gss

Для аутентификации пользователя использует GSSAPI. Этот способ доступен только для подключений по TCP/IP;

- ident

Получает имя пользователя операционной системы клиента, связываясь с сервером Ident, и проверяет, соответствует ли оно имени пользователя базы данных. Аутентификация `ident` может использоваться только для подключений по TCP/IP. Для локальных подключений применяется аутентификация `peer`;

- peer

Получает имя пользователя операционной системы клиента из операционной системы и проверяет, соответствует ли оно имени пользователя запрашиваемой базы данных. Доступно только для локальных подключений;

- 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 операционной системой;

10) параметры-аутентификации

После поля метод-аутентификации может идти поле (поля) вида `имя=значение`, определяющее параметры метода аутентификации. Подробнее о параметрах, доступных для различных методов аутентификации, рассказывается ниже.

Помимо описанных далее параметров, относящихся к различным методам, есть один общий параметр аутентификации `clientcert`, который можно задать в любой записи `hostssl`. Если он равен 1, клиент должен представить подходящий (доверенный) сертификат SSL, в дополнение к другим требованиям метода проверки подлинности.

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

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

Файл pg_hba.conf прочитывается во время запуска и в момент получения основным процессом сервера сигнала SIGHUP. Если вы редактируете файл во время работы системы, необходимо послать сигнал процессу postmaster, используя pg_ctl reload или kill -HUP, чтобы он прочел обновленный файл.

Системное представление pg_hba_file_rules может быть полезно для предварительной проверки изменений в файле pg_hba.conf или для диагностики проблем, когда перезагрузка этого файла не даёт желаемого эффекта. Строки в этом представлении, содержащие в поле error не NULL, указывают на проблемы в соответствующих строках файла.

За более подробной информацией по методам аутентификации обратитесь к п. 4.3. Примеры записей файла pg_hba.conf показаны ниже.

Пример.

```
# Позволяет любому пользователю локальной системы подключаться
# к любой базе данных, используя любое имя пользователя баз данных, через
# Unix-сокеты (по умолчанию для локальных подключений).
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust

# То же, но для локальных замкнутых подключений по TCP/IP.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             127.0.0.1/32            trust

# То же, что и в предыдущей строке, но с указанием
# сетевой маски в отдельном столбце
#
# TYPE  DATABASE        USER         IP-ADDRESS      IP-MASK          METHOD
host    all             all          127.0.0.1    255.255.255.255     trust

# То же для IPv6.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             ::1/128                 trust

# То же самое, но с использованием имени компьютера
# (обычно покрывает и IPv4, и IPv6).
```

```
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             localhost               trust

# Позволяет любому пользователю любого компьютера с IP-адресом
# 192.168.93.x подключаться к базе данных "postgres"
# с именем, которое сообщает для данного подключения ident
# (как правило, имя пользователя операционной системы).
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    postgres        all             192.168.93.0/24         ident

# Позволяет любому пользователю компьютера 192.168.12.10 подключаться
# к базе данных "postgres", если он передаёт правильный пароль.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    postgres        all             192.168.12.10/32        scram-sha-256

# Позволяет любым пользователям с компьютеров в домене example.com
# подключаться к любой базе данных, если передаётся правильный пароль.
#
# Для всех пользователей требуется аутентификация SCRAM, за исключением
# пользователя 'mike', который использует старый клиент, не поддерживающий
# аутентификацию SCRAM.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             mike            .example.com            md5
host    all             all             .example.com            scram-sha-256

# В случае отсутствия предшествующих строчек с "host", следующие две строки
# откажут в подключении с 192.168.54.1 (поскольку данная запись будет
# выбрана первой), но разрешат подключения GSSAPI с любых других
# адресов. С нулевой маской ни один бит из IP-адреса компьютера
# не учитывается, так что этой строке соответствует любой компьютер.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             192.168.54.1/32         reject
host    all             all             0.0.0.0/0               gss

# Позволяет пользователям с любого компьютера 192.168.x.x подключаться
# к любой базе данных, если они проходят проверку ident.  Если же ident
# говорит, например, что это пользователь "bryanh" и он запрашивает
# подключение как пользователь Синергия-БД "guest1", подключение
```

```
# будет разрешено, если в файле pg_ident.conf есть сопоставление
# "omicron", позволяющее пользователю "bryanh" подключаться как "guest1".
#
# TYPE  DATABASE        USER         ADDRESS                 METHOD
host    all             all          192.168.0.0/16          ident map=omicron

# Если для локальных подключений предусмотрены только эти три строки,
# они позволят локальным пользователям подключаться только к своим
# базам данных (базам данных с именами, совпадающими с
# именами пользователей баз данных), кроме администраторов
# или членов роли "support", которые могут подключиться к любой БД.
# Список имён администраторов содержится в файле $PGDATA/admins.
# Пароли запрашиваются в любом случае.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   sameuser        all                                     md5
local   all             @admins                                 md5
local   all             +support                                md5

# Последние две строчки выше могут быть объединены в одну:
local   all             @admins,+support                        md5

# В столбце DATABASE также могут указываться списки и имена файлов:
local   db1,db2,@demodbs  all                                   md5
```

4.2. Файл сопоставления имён пользователей

Когда используется внешняя система аутентификации, такая, как Ident или GSSAPI, случается, что имя пользователя операционной системы, инициировавшей подключение, может не совпадать с именем пользователя базы данных, под которым он хочет подключиться. В этом случае может быть составлен файл сопоставления имён пользователя, чтобы соотнести имя пользователя операционной системы и пользователя базы данных. Чтобы использовать функцию сопоставления имён пользователя, укажите `map=map-name` в поле параметров `pg_hba.conf`. Этот параметр поддерживается для всех методов аутентификации, получающих внешние имена пользователей. Для различных подключений могут использоваться разные файлы сопоставления. Чтобы указать, какой файл сопоставления использовать при каждом конкретном подключении, имя нужного файла сопоставления должно быть

указано в параметре map-name файла pg_hba.conf.

Сопоставления имён пользователя определяются в файле сопоставления ident, который по умолчанию называется pg_ident.conf и хранится в каталоге данных кластера. Файл сопоставления может быть помещён и в другое место, обратитесь к информации о настройке параметра ident_file. Файл сопоставления ident содержит строки общей формы:

```
map-name system-username database-username
```

Комментарии и пробелы применяются так же, как и в файле pg_hba.conf. map-name является произвольным именем, на которое будет ссылаться файл сопоставления файла pg_hba.conf. Два других поля указывают имя пользователя операционной системы и соответствующее имя пользователя базы данных. Имя map-name может быть использовано неоднократно, чтобы указывать множественные сопоставления пользовательских имён в рамках одного файла сопоставления.

Нет никаких ограничений по количеству пользователей баз данных, на которые может ссылаться пользователь операционной системы, и наоборот. Тем не менее, записи в файле скорее подразумевают, что «пользователь этой операционной системы может подключиться как пользователь этой базы данных», нежели показывают, что эти имена пользователей эквивалентны. Подключение разрешается, если существует запись в файле сопоставления, соединяющая имя, полученное от внешней системы аутентификации, с именем пользователя базы данных, под которым пользователь хочет подключиться.

Если поле system-username начинается со знака (/), оставшаяся его часть рассматривается как регулярное выражение. Регулярное выражение может включать в себя одну группу, или заключённое в скобки подвыражение, на которое можно сослаться в поле database-username, написав \1 (с одной обратной косой). Это позволяет сопоставить несколько имён пользователя с одной строкой, что особенно удобно для простых замен. Например, эти строки:

```
mymap    /^(.*)@mydomain\.com$       \1
mymap    /^(.*)@otherdomain\.com$    guest
```

удалят часть домена для имён пользователей, которые заканчиваются на

@mydomain.com, и позволят пользователям, чьё имя пользователя системы заканчивается на @otherdomain.com, подключиться как guest.

Файл pg_ident.conf читается при запуске системы, а также в тот момент, когда основной сервер получает сигнал SIGHUP. Если вы редактируете файл во время работы системы, необходимо послать сигнал процессу postmaster, используя pg_ctl reload или kill -HUP, чтобы он прочел обновленный файл.

Файл pg_ident.conf, который может быть использован в сочетании с файлом pg_hba.conf, показан в примере ниже. В этом примере любым пользователям компьютеров в сети 192.168 с именами, отличными от bryanh, ann или robert, будет отказано в доступе. Пользователь системы robert получит доступ только тогда, когда подключается как пользователь Синергия-БД bob, а не как robert, или какой-либо другой пользователь. Пользователь ann сможет подключиться только как ann. Пользователь bryanh сможет подключиться как bryanh или как guest1.

Пример.

```
# MAPNAME       SYSTEM-USERNAME         PG-USERNAME
omicron         bryanh                  bryanh
omicron         ann                     ann
# на этих машинах bob может подключаться как robert
omicron         robert                  bob
# bryanh также может подключаться как guest1
omicron         bryanh                  guest1
```

5. РОЛИ БАЗЫ ДАННЫХ

Синергия-БД 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 привилегии 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.

В этом разделе описывается, как создавать и управлять ролями.

5.1. Роли базы данных

Роли базы данных концептуально полностью отличаются от пользователей операционной системы. На практике поддержание соответствия между ними может быть удобным, но не является обязательным. Роли базы данных являются глобальными для всей установки кластера базы данных (не для отдельной базы данных). Для создания роли используется команда SQL `CREATE ROLE`:

```
CREATE ROLE имя;
```

Это `имя` соответствует правилам именования идентификаторов SQL: либо обычное, без специальных символов, либо в двойных кавычках. На практике, к команде обычно добавляются другие указания, такие как `LOGIN`. 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, например:

```
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 операционной системы, инициализирующий кластер баз данных. Обычно эта роль называется `postgres`. Для создания других ролей, вначале нужно подключиться с этой ролью.

Каждое подключение к серверу базы данных выполняется под именем конкретной роли, и эта роль определяет начальные привилегии 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. Например, программа `psql` для указания роли использует аргумент командной строки `-U`. Многие приложения предполагают, что по умолчанию нужно использовать имя пользователя операционной системы (включая `createuser` и `psql`). Поэтому часто бывает удобным поддерживать соответствие между именами ролей и именами пользователей операционной системы.

Список доступных для подключения ролей, который могут использовать клиенты, определяется конфигурацией аутентификации, как описывалось в п. 4. Поэтому клиент не ограничен только ролью, соответствующей имени пользователя операционной системы, также, как и имя для входа может не соответствовать реальному имени. Так как роль определяет набор доступных привилегий, очень важно тщательно настраивать привилегии в многопользовательской среде.

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 роли такой роли можно использовать любой из вариантов:

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

Команда CREATE USER эквивалентна CREATE ROLE за исключением того, что CREATE USER по умолчанию предполагает атрибут LOGIN, в то время как CREATE ROLE нет;

- 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 CREATE ROLE имя SUPERUSER. Это 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 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 CREATE ROLE имя CREATEROLE. Роль с привилегией CREATEROLE может также изменять и удалять другие роли, а

также выдавать и отзывать членство в ролях. Однако, для создания, изменения, удаления суперпользовательских ролей, а также изменения в них членства, требуется иметь статус суперпользователя; привилегии 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 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 операционная система используют раздельные пароли. Пароль указывается при создании роли: CREATE ROLE имя PASSWORD 'строка'.

Атрибуты ролей могут быть изменены после создания командой ALTER ROLE. Более детальная информация в справке по командам CREATE ROLE и ALTER ROLE.

Рекомендуется создать роль с привилегиями CREATEDB и CREATEROLE, но не суперпользователя и в последующем использовать её для управления базами данных и ролями. Такой подход позволит избежать опасностей, связанных с использованием полномочий суперпользователя для задач, которые их не требуют.

На уровне ролей можно устанавливать многие конфигурационные параметры времени выполнения, описанные в п. 3. Например, если по некоторым причинам всякий раз при подключении к базе данных требуется отключить использование индексов (подсказка: плохая идея) можно выполнить:

```
ALTER ROLE myname SET enable_indexscan TO off;
```

Установленное значение параметра будет сохранено (но не будет применено сразу). Для последующих подключений с этой ролью это будет выглядеть как

выполнение команды `SET enable_indexscan TO off` перед началом сессии. Но это только значение по умолчанию, в течение сессии эту установку можно изменить. Для удаления установок на уровне ролей для параметров конфигурации используется `ALTER ROLE rolename RESET varname`. Обратите внимание, что установка параметров конфигурации на уровне роли без привилегии `LOGIN` лишено смысла, т. к. они никогда не будут применены.

5.3. Членство в роли

Часто бывает удобным сгруппировать пользователей для упрощения администрирования привилегий: привилегии выдаются или отзываются на всю группу. В Синергия-БД для этого создаётся роль, которая представляет группу, а затем членство (membership) 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`, хотя при желании его можно установить.

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

```
GRANT group_role TO role1, ... ;
REVOKE group_role FROM role1, ... ;
```

Можно выдавать членство в групповой роли другим групповым ролям (потому что в действительности нет никаких различий между групповыми и не групповыми ролями). База данных не позволит замкнуть предоставление членства по кругу. Также, не допускается выдача членства в роли для `PUBLIC`.

Члены групповой роли могут использовать её привилегии двумя способами. Во-первых, каждый член группы может явно выполнить `SET ROLE`, чтобы временно «стать» групповой ролью. В этом состоянии, сессия 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 привилегии всех ролей, членами которых они являются, в том числе и унаследованными этими ролями привилегиями, например:

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
```

После подключения с ролью joe, сессия базы данных будет использовать полномочия, выданные напрямую joe и привилегии, выданные admin, так как joe «наследует» привилегии admin. Однако привилегии, выданные wheel не будут доступны, потому что, хотя joe неявно и является членом wheel, это членство получено через роль admin, которая имеет атрибут NOINHERIT. После выполнения команды:

```
SET ROLE admin;
```

сессия будет использовать только привилегии, выданные admin, привилегии, выданные joe не будут доступны. После выполнения команды:

```
SET ROLE wheel;
```

сессия будет использовать только привилегии, выданные wheel, привилегии joe и admin не будут доступны. Начальный набор привилегий можно вернуть любой из команд:

```
SET ROLE joe;
SET ROLE NONE;
RESET ROLE;
```

Команда SET ROLE в любой момент разрешает выбрать любую роль, прямым или косвенным членом которой является оригинальная роль, под которой был выполнен вход в систему. Поэтому, в примере выше, не обязательно сначала становиться admin, перед тем как стать wheel.

В стандарте SQL есть чёткое различие между пользователями и ролями. При

этом пользователи, в отличие от ролей, не наследуют привилегии автоматически. Такое поведение может быть получено в Синергия-БД, если для ролей, используемых как роли в стандарте SQL, устанавливать атрибут INHERIT, а для ролей, используемых как пользователи в стандарте SQL, устанавливать атрибут NOINHERIT. Однако, в Синергия-БД все роли по умолчанию имеют атрибут INHERIT. Это сделано для обратной совместимости с версиями, предшествующими 8.1, в которых пользователи всегда могли использовать привилегии групп, членами которых они являются.

Атрибуты роли LOGIN, SUPERUSER, CREATEDB и CREATEROLE можно рассматривать как особые привилегии, но они никогда не наследуются, как обычные привилегии на объекты базы данных. Необходимо через SET ROLE установить роль, имеющую один из этих атрибутов, чтобы им воспользоваться. Продолжая предыдущий пример, можно установить атрибуты CREATEDB и CREATEROLE для роли admin. Затем при входе с ролью joe, получить доступ к этим привилегиям будет возможно только после выполнения SET ROLE admin.

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).

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.

Владение объектами можно передавать в индивидуальном порядке, применяя команду ALTER, например:

```
ALTER TABLE bobs_table OWNER TO alice;
```

Кроме того, для переназначения какой-либо другой роли владения сразу всеми объектами, принадлежащих удаляемой роли, можно применить команду REASSIGN

OWNED. Так как REASSIGN OWNED не может обращаться к объектам в других базах данных, её необходимо выполнить в каждой базе, которая содержит объекты, принадлежащие этой роли. Заметьте, что первая такая команда REASSIGN OWNED изменит владельца для всех разделяемых между базами объектов, то есть для баз данных или табличных пространств, принадлежащих удаляемой 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. Также заметьте, что 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.

С учётом этого, общий рецепт удаления роли, которая владела объектами, вкратце таков:

```
REASSIGN OWNED BY doomed_role TO successor_role;
DROP OWNED BY doomed_role;
-- повторить предыдущие команды для каждой базы в
кластере
DROP ROLE doomed_role;
```

Когда не все объекты нужно передать одному новому владельцу, лучше сначала вручную отработать исключения, а в завершение выполнить показанные выше действия.

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

5.5. Предопределённые роли

В Синергия-БД 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.

Имеющиеся предопределённые роли описаны в таблице 5.1. Заметьте, что конкретные разрешения для каждой из предопределённых ролей в будущем могут изменяться по мере добавления дополнительной функциональности. Администраторы должны следить за этими изменениями, просматривая замечания к выпускам.

Таблица 5.1 – Предопределённые роли

Роль	Разрешаемый 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_signal_backend	Передавать сигналы другим обслуживающим процессам (например, отменять запрос, завершать процесс).
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.
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.

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.

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.

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`:

```
GRANT pg_signal_backend TO admin_user;
```

5.6. Безопасность функций

Функции, триггеры и политики защиты на уровне строк позволяют пользователям внедрять код в обслуживающие процессы, который может быть непреднамеренно выполнен другими пользователями. Таким образом эти механизмы позволяют пользователям запускать «троянский код» относительно просто. Лучшая защита от этого - строгое ограничение круга лиц, которые могут создавать объекты. Там, где это невозможно, пишите запросы так, чтобы они ссылались только на объекты с доверенными владельцами. Удалите из `search_path` схему `public` и любые другие схемы, в которых могут создавать объекты недоверенные пользователи.

Функции выполняются внутри серверного процесса с полномочиями пользователя операционной системы, запускающего сервер базы данных. Если используемый для функций язык программирования разрешает неконтролируемый доступ к памяти, то это даёт возможность изменить внутренние структуры данных сервера. Таким образом, помимо всего прочего, такие функции могут обойти ограничения доступа к системе. Языки программирования, допускающие такой доступ, считаются «недоверенными» и создавать функции на этих языках Синергия-БД разрешает только суперпользователям.

6. УПРАВЛЕНИЕ БАЗАМИ ДАННЫХ

Каждый работающий экземпляр сервера Синергия-БД обслуживает одну или несколько баз данных. Поэтому базы данных представляют собой вершину иерархии SQL-объектов («объектов базы данных»). Данный раздел описывает свойства баз данных, процессы создания, управления и удаления.

6.1. Обзор

База данных - именованная коллекция объектов SQL («объектов базы данных»). В целом, каждый объект базы данных (таблицы, функции и т. д.) принадлежит одной и только одной базе данных. Правда есть несколько системных каталогов, например, `pg_database`, которые принадлежат всему кластеру и доступны для каждой базы данных этого кластера. Если точнее, база данных это набор схем, которые включают в себя таблицы, функции и т. д. Таким образом, полная иерархия включает в себя: сервер, базу данных, схему, таблицу (или иные типы объектов, к примеру, функции).

При подключении к серверу базы данных, клиент должен указать в параметрах подключения имя базы данных, с которой нужно соединиться. Одно соединение не может иметь доступ более чем к одной базе данных. Однако приложение не ограничено в количестве соединений к одной и той же или разным базам данных. Базы данных разделены физически и контроль доступа осуществляется на уровне соединения. В случае, когда один экземпляр сервера Синергия-БД обслуживает проекты или пользователей, которых необходимо изолировать друг от друга, рекомендуется размещать их в раздельных базах данных. В случае, когда проекты или пользователи взаимосвязаны и должны иметь возможность использовать общие ресурсы, они должны размещаться в одной базе данных, но, возможно, в раздельных схемах. Схемы - в чистом виде логическая структура, и кто к чему может получить доступ управляется системой привилегий.

Базы данных создаются командой `CREATE DATABASE`, а удаляются командой `DROP DATABASE`. Список существующих баз данных можно посмотреть в системном каталоге `pg_database`, например:

```
SELECT datname FROM pg_database;
```

Метакоманда \l или ключ -l командной строки приложения psql также позволяют вывести список существующих баз данных.

6.2. Создание базы данных

Для создания базы данных сервер Синергия-БД должен быть развёрнут и запущен.

База данных создаётся SQL-командой CREATE DATABASE:

```
CREATE DATABASE имя;
```

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

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. Более подробно в п. 6.3.

Для удобства, есть утилита командной строки для создания баз данных, createdb.

```
createdb dbname
```

Утилита createdb подключается к базе данных postgres и выполняет

ранее описанную SQL-команду `CREATE DATABASE`. Подробнее о её вызове можно узнать в `createdb`. Обратите внимание, что команда `createdb` без параметров создаст базу данных с именем текущего пользователя.

Иногда необходимо создать базу данных для другого пользователя и назначить его владельцем, чтобы он мог конфигурировать и управлять ею. Для этого используйте одну из следующих команд:

```
CREATE DATABASE dbname OWNER rolename;
```

из среды SQL, или:

```
createdb -O rolename dbname
```

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

6.3. Конфигурирование баз данных

Обратившись к п. 2 можно выяснить, что сервер Синергия-БД имеет множество параметров конфигурации времени исполнения. Можно выставить специфичные для базы данных значения по умолчанию.

Например, если по какой-то причине 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, используйте `ALTER DATABASE dbname RESET varname`.

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 утилита командной строки для удаления баз данных `dropdb`:

```
dropdb dbname
```

В отличие от команды `createdb` утилита не использует имя текущего пользователя по умолчанию.

6.5. Табличные пространства

Табличные пространства в Синергия-БД 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 Синергия-БД. Это полезно минимум 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. Например, часто используемый индекс можно разместить

на очень быстром и надёжном, но дорогом SSD-диске. В то же время таблица с архивными данными, которые редко используются и скорость доступа к ним не важна, может быть размещена в более дешёвом и медленном хранилище.

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 Синергия-БД. 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;
```

Как вариант, используйте 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 используйте команду `DROP TABLESPACE`.

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

```
SELECT spcname FROM pg_tablespace;
```

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

Синергия-БД использует 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.

7. ЛОКАЛИЗАЦИЯ

Данный раздел описывает доступные возможности локализации с точки зрения администратора. Синергия-БД поддерживает два средства локализации:

- использование средств локализации операционной системы для обеспечения определяемых локалью порядка правил сортировки (таблица 7.1), форматирования чисел, перевода сообщений и прочих аспектов. Это рассматривается в п. 7.1 и п. 7.2;
- обеспечение возможностей использования различных кодировок для хранения текста на разных языках и перевода кодировок между клиентом и сервером. Это рассматривается в п. 7.3;

7.1. Поддержка языковых стандартов

Поддержка языковых стандартов в приложениях относится к культурным предпочтениям, которые касаются алфавита, порядка сортировки, форматирования чисел и т. п. Синергия-БД использует соответствующие стандартам ISO C и POSIX возможности локали, предоставляемые операционной системой сервера. За дополнительной информацией обращайтесь к документации вашей системы.

7.1.1. Обзор

Поддержка локали автоматически инициализируется, когда кластер базы данных создаётся при помощи `initdb`. `initdb` инициализирует кластер баз данных по умолчанию с локалью из окружения выполнения. Поэтому, если ваша система уже использует локаль, которую вы хотите выбрать для вашего кластера баз данных, вам не требуется дополнительно совершать никаких действий. Если вы хотите использовать другую локаль (или вы точно не знаете, какая локаль используется в вашей системе), вы можете указать для `initdb` какую именно локаль использовать через задание параметра `–locale`, например:

```
initdb --locale=ru_RU
```

Данный пример для Unix-систем задаёт русский язык (`ru`), на котором говорят в России (`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 на вашей системе, зависит от того, что было включено в операционную систему 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:

Таблица 7.1– Порядок сортировки строк

LC_COLLATE	Порядок сортировки строк
LC_CTYPE	Классификация символов (Что представляет собой буква? Каков её эквивалент в верхнем регистре?)
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. Например, чтобы настроить локаль на канадский французский, но при этом использовать американские правила форматирования денежных сумм, используйте initdb --locale=fr_CA --lc-monetary=en_US.

Если вы хотите, чтобы система работала без языковой поддержки, используйте специальное имя локали C либо эквивалентное ему POSIX.

Значения некоторых категорий локали должны быть заданы при создании базы данных. Вы можете использовать различные параметры локали для различных баз данных, но после создания базы вы уже не сможете 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. Однако, можно смягчить эти ограничения через задание правил сравнения, как это описано в п. 7.2. 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 вы можете изменить в любое время, настроив параметры конфигурации сервера, которые имеют такое же имя, как и категории локали (подробнее см. п. 3.11.2). Значения, выбранные через `initdb`, фактически записываются лишь в файл конфигурации `postgresql.conf`, чтобы использоваться по умолчанию при запуске сервера. Если вы удалите эти значения из `postgresql.conf`, сервер получит соответствующие значения из своей среды выполнения.

Обратите 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.

Некоторые библиотеки локализации сообщений также обращаются к переменной среды `LANGUAGE`, которая заменяет все прочие параметры локализации при выборе языка сообщений. В случае затруднений, пожалуйста, воспользуйтесь документацией по вашей операционной системе, в частности, справкой по `gettext`.

Для того чтобы стал возможен перевод сообщений на язык, выбранный пользователем, NLS должен быть выбран на момент сборки (`configure --enable-nls`). В остальном поддержка локализации осуществляется автоматически.

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 Синергия-БД является влияние на производительность. Это замедляет обработку символов и мешает `LIKE` использовать обычные индексы. По этой причине используйте локали только в том случае, если они действительно вам нужны.

В качестве обходного решения, которое позволит Синергия-БД пользоваться индексами с предложениями `LIKE` с использованием локали, отличной от C, существует несколько классов пользовательских операторов. Они позволяют создать индекс, который выполняет строгое посимвольное сравнение, игнорируя правила сравнения, соответствующие локали.

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.

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, в случае, если 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.

В качестве примера рассмотрим данное определение таблицы:

```
CREATE TABLE test1 (
    a text COLLATE "de_DE",
    b text COLLATE "es_ES",
    ...
);
```

Затем в

```
SELECT a < 'foo' FROM test1;
```

выполняется оператор сравнения < согласно правилам de_DE, так как выражение объединяет неявно определяемые правила сортировки с правилами,

заданными по умолчанию. Но в

```
SELECT a < ('foo' COLLATE "fr_FR") FROM test1;
```

сравнение выполняется с помощью правил `fr_FR`, так как явное определение правил сортировки переопределяет неявное. Кроме того, получив

```
SELECT a < b FROM test1;
```

анализатор запросов не может определить, какое правило сортировки использовать, поскольку столбцы `a` и `b` имеют конфликтующие неявные правила сортировки. Так как оператору < требуется знать, какое правило использовать, это приведёт к ошибке. Ошибку можно устранить, применив явное указание правил сортировки к любому из двух входных выражений, например:

```
SELECT a < b COLLATE "de_DE" FROM test1;
```

либо эквивалентное ему

```
SELECT a COLLATE "de_DE" < b FROM test1;
```

С другой стороны, следующее выражение схожей структуры

```
SELECT a || b FROM test1;
```

не приводит к ошибке, поскольку для оператора `||` правила сортировки не имеют значения, так как результат не зависит от сортировки.

Правила сортировки, назначенные функции или комбинации входных выражений оператора, также могут быть применены к функции или результату оператора, если функция или оператор возвращают результат сортируемого типа данных. Так, в

```
SELECT * FROM test1 ORDER BY a || 'foo';
```

упорядочение будет происходить согласно правилам `de_DE`. Но данный запрос:

```
SELECT * FROM test1 ORDER BY a || b;
```

приводит к ошибке, потому что, даже если оператору `||` не нужно знать правила сортировки, предложению `ORDER BY` это требуется. Как было сказано выше, конфликт может быть разрешён при помощи явного указания правил сортировки:

```
SELECT * FROM test1 ORDER BY a || b COLLATE "fr_FR";
```

7.2.2. Управление правилами сортировки

Правила сортировки представляют собой объект схемы SQL, который сопоставляет SQL-имя с локалью, реализуемой библиотекой, установленной в операционной системе. В определении правила сортировки задаётся *провайдер*, то есть библиотека, реализующая правило сортировки. Стандартный провайдер с именем `libc` использует системную библиотеку C и предоставляет её локали. Именно эти локали используются большинством утилит операционной системы. Также есть провайдер `icu`, который использует внешнюю библиотеку ICU. Локали ICU можно использовать, только если поддержка ICU была включена в конфигурации сборки Синергия-БД.

Правило сортировки, предоставляемое провайдером `libc`, сопоставляется с комбинацией параметров `LC_COLLATE` и `LC_CTYPE`, которую может принять системный вызов `setlocale()`. Основная цель правила сортировки - настроить параметр `LC_COLLATE`, который управляет упорядочиванием символов. Однако на практике редко требуется иметь значение `LC_CTYPE`, отличное от `LC_COLLATE`, поэтому удобнее объединить их в одну сущность, и не создавать отдельную инфраструктуру для указания `LC_CTYPE` в выражениях. Правила сортировки `libc` также связаны с кодировкой набора символов. Одно и то же имя правила сортировки может существовать для разных кодировок.

Объект правила сортировки, предоставляемой провайдером `icu`, сопоставляется с именованным сортировщиком, реализуемым библиотекой ICU. ICU не поддерживает различные характеристики «collate» и «ctype», так что они всегда совпадают. Кроме того, правила сортировки ICU не зависят от кодировки, так что в базе данных будет всего одно правило сортировки ICU с определённым именем.

7.3. Поддержка кодировок

Поддержка кодировок в Синергия-БД 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 Синергия-БД 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 есть только одна кодировка, которая будет работать правильно.

7.3.1. Поддерживаемые кодировки

Таблица 7.2 показывает кодировки, доступные в Синергия-БД.

Таблица 7.2 – Кодировки Синергия-БД

Имя	Описание	Язык	Поддержка на сервере	ICU?	Байтов 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_	ISO 8859-7,	Латинский/Греческий	Да	Да	1	

Имя	Описание	Язык	Поддержка на сервере	ICU?	Байтов на символ	Псевдонимы
7	ECMA 118					
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

Продолжение таблицы 7.2

Имя	Описание	Язык	Поддержка на сервере	ICU?	Байтов на символ	Псевдонимы
KOI8U	KOI8-U	Кириллица (Украинский)	Да	Да	1	
LATIN1	ISO 8859-1, ECMA 94	Западноевропейские	Да	Да	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Центрально-европейские	Да	Да	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	Южноевропейские	Да	Да	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	Североевропейские	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	Мультиязычный редактор Emacs	Да	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	Унифицированный код Хангыль	Корейский	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	Центральноевропейские	Да	Да	1	

Окончание таблицы 7.2

Имя	Описание	Язык	Поддержка на сервере	ICU?	Байтов на символ	Псевдонимы
WIN1251	Windows CP1251	Кириллица	Да	Да	1	WIN
WIN1252	Windows CP1252	Западноевропейские	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 Синергия-БД 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 значения от 0 до 127 байт согласно кодировке ASCII, тогда как значения от 128 до 255 воспринимаются как незначимые. Перекодировка не будет выполнена при выборе SQL_ASCII. Таким образом, этот вариант является не столько объявлением того, что используется определённая кодировка, сколько объявлением того, что кодировка игнорируется. В большинстве случаев, если вы работаете с любыми данными, отличными от ASCII, не стоит использовать SQL_ASCII, так как Синергия-БД не сможет преобразовать или проверить символы, отличные от ASCII.

7.3.2. Настройка кодировки

initdb определяет кодировку по умолчанию для кластера Синергия-БД, 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;
```

Заметьте, что 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. Более подробное описание приведено в п. 6.3.

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

```
$ psql -l
                                        List of databases
   Name    |  Owner   | Encoding  |  Collation  |    Ctype    |
Access Privileges
-----------+----------+-----------+-------------+-------------+--
-----------------------------------
 clocaledb | hlinnaka | SQL_ASCII | C           | C           |
 englishdb | hlinnaka | UTF8      | en_GB.UTF8  | en_GB.UTF8  |
 japanese  | hlinnaka | UTF8      | ja_JP.UTF8  | ja_JP.UTF8  |
 korean    | hlinnaka | EUC_KR    | ko_KR.euckr | ko_KR.euckr |
```

```
 postgres  | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  |
 template0 | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  |
{=c/hlinnaka, hlinnaka=CTc/hlinnaka}
 template1 | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  |
{=c/hlinnaka, hlinnaka=CTc/hlinnaka}
(7 rows)
```

На большинстве современных операционных систем Синергия-БД может определить, какая кодировка подразумевается параметром `LC_CTYPE`, что обеспечит использование только соответствующей кодировки базы данных. На более старых системах необходимо самостоятельно следить за тем, чтобы использовалась кодировка, соответствующая выбранной языковой среде. Ошибка в этой области, скорее всего, приведёт к странному поведению зависимых от локали операций, таких как сортировка.

Синергия-БД позволит суперпользователям создавать базы данных с кодировкой `SQL_ASCII`, даже когда значение `LC_CTYPE` не установлено в `C` или `POSIX`. Как было сказано выше, `SQL_ASCII` не гарантирует, что данные, хранящиеся в базе, имеют определённую кодировку, и таким образом, этот выбор чреват сбоями, связанными с локалью. Использование данной комбинации устарело и, возможно, будет полностью запрещено.

8. ПРИРЕГЛАМЕНТНЫЕ ЗАДАЧИ ОБСЛУЖИВАНИЯ БАЗЫ ДАННЫХ

Как и в любой СУБД, в Синергия-БД для достижения оптимальной производительности нужно регулярно выполнять определённые процедуры. Задачи, которые рассматриваются в этом разделе, являются обязательными, но они по природе своей повторяющиеся и легко поддаются автоматизации с использованием стандартных средств, таких как задания cron. Создание соответствующих заданий и контроль над их успешным выполнением входят в обязанности администратора базы данных.

Одной из очевидных задач обслуживания СУБД является регулярное создание резервных копий данных. При отсутствии свежей резервной копии у вас не будет шанса восстановить систему после катастрофы (сбой диска, пожар, удаление важной таблицы по ошибке и т. д.). Механизмы резервного копирования и восстановления в Синергия-БД детально рассматриваются в п. 1 (Часть 2. Руководство системного программиста).

Другое важное направление обслуживания СУБД - периодическая «очистка» базы данных. Эта операция рассматривается в п. 8.1. С ней тесно связано обновление статистики, которая будет использоваться планировщиком запросов.

Ещё одной задачей, требующей периодического выполнения, является управление файлами журнала. Она рассматривается в п. 8.3.

Для контроля состояния базы данных и для отслеживания нестандартных ситуаций можно использовать `check_postgres`. Скрипт `check_postgres` можно интегрировать с Nagios и MRTG, однако он может работать и самостоятельно.

По сравнению с некоторыми другими СУБД Синергия-БД неприхотлив в обслуживании. Тем не менее, должное внимание к вышеперечисленным задачам будет значительно способствовать комфортной и производительной работе с СУБД.

8.1. Регламентная очистка

Базы данных Синергия-БД требуют периодического проведения процедуры обслуживания, которая называется очисткой. Во многих случаях очистку достаточно

выполнять с помощью демона автоочистки, который описан в п. 8.1.6. Возможно, в вашей ситуации для получения оптимальных результатов потребуется настроить описанные там же параметры автоочистки. Некоторые администраторы СУБД могут дополнить или заменить действие этого демона командами VACUUM (обычно они выполняются по расписанию в заданиях cron или Планировщика задач). Чтобы правильно организовать очистку вручную, необходимо понимать темы, которые будут рассмотрены в следующих подразделах. Администраторы, которые полагаются на автоочистку, возможно, всё же захотят просмотреть этот материал, чтобы лучше понимать и настраивать эту процедуру.

8.1.1. Основные принципы очистки

Команды VACUUM в Синергия-БД 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 Синергия-БД;
- 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, как рассматривается в следующих подразделах.

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 исключительной блокировки обрабатываемой

таблицы 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 (см. п. 3.4.4).

8.1.2. Высвобождение дискового пространства

В Синергия-БД команды UPDATE или DELETE не вызывают немедленного удаления старой версии изменяемых строк. Этот подход необходим для реализации эффективного многоверсионного управления конкурентным доступом (MVCC): версия строки не должна удаляться до тех пор, пока она остаётся потенциально видимой для других транзакций. Однако в конце концов устаревшая или удалённая версия строки оказывается не нужна ни одной из транзакций. После этого занимаемое ей место должно быть освобождено и может быть отдано новым строкам, во избежание неограниченного роста потребности в дисковом пространстве. Это происходит при выполнении команды VACUUM.

Обычная форма VACUUM удаляет неиспользуемые версии строк в таблицах и индексах и помечает пространство свободным для дальнейшего использования. Однако это дисковое пространство не возвращается операционной системе, кроме особого случая, когда полностью освобождаются одна или несколько страниц в конце таблицы и можно легко получить исключительную блокировку таблицы. Команда VACUUM FULL, напротив, кардинально сжимает таблицы, записывая абсолютно новую версию файла таблицы без неиспользуемого пространства. Это минимизирует размер таблицы, однако может занять много времени. Кроме того, для этого требуется больше места на диске для записи новой копии таблицы до завершения операции.

Обычно цель регулярной очистки - выполнять простую очистку (VACUUM) достаточно часто, чтобы не возникала необходимость в VACUUM FULL. Демон

автоочистки пытается работать в этом режиме, и на самом деле он сам никогда не выполняет VACUUM FULL. Основная идея такого подхода не в том, чтобы минимизировать размер таблиц, а в том, чтобы поддерживать использование дискового пространства на стабильном уровне: каждая таблица занимает объём, равный её минимальному размеру, плюс объём, который был занят между процедурами очистки. Хотя с помощью VACUUM FULL можно сжать таблицу до минимума и возвратить дисковое пространство операционной системе, большого смысла в этом нет, если в будущем таблица так же вырастет снова. Следовательно, для активно изменяемых таблиц лучше с умеренной частотой выполнять VACUUM, чем очень редко выполнять VACUUM FULL.

Некоторые администраторы предпочитают планировать очистку БД самостоятельно, например, проводя все работы ночью в период низкой загрузки. Однако очистка только по фиксированному расписанию плоха тем, что при резком скачке интенсивности изменений таблица может разрастить настолько, что для высвобождения пространства действительно понадобится выполнить VACUUM FULL. Использование демона автоочистки снимает эту проблему, поскольку он планирует очистку динамически, отслеживая интенсивность изменений. Полностью отключать этот демон может иметь смысл, только если вы имеете дело с предельно предсказуемой загрузкой. Возможен и компромиссный вариант - настроить параметры демона автоочистки так, чтобы он реагировал только на необычайно высокую интенсивность изменений и мог удержать ситуацию под контролем, в то время как команды VACUUM, запускаемые по расписанию, будут выполнять основную работу в периоды нормальной загрузки.

Если же автоочистка не применяется, обычно планируется выполнение VACUUM для всей базы данных раз в сутки в период низкой активности, и в случае необходимости оно дополняется более частой очисткой интенсивно изменяемых таблиц. В некоторых ситуациях, когда изменения производятся крайне интенсивно, самые востребованные таблицы могут очищаться раз в несколько минут. Если в вашем кластере несколько баз данных, не забывайте выполнять VACUUM для каждой из них; при этом может быть полезна программа vacuumdb.

8.1.3. Обновление статистики планировщика

Планировщик запросов в Синергия-БД, выбирая эффективные планы запросов, полагается на статистическую информацию о содержимом таблиц. Эта статистика собирается командой `ANALYZE`, которая может вызываться сама по себе или как дополнительное действие команды `VACUUM`. Статистика должна быть достаточно точной, так как в противном случае неудачно выбранные планы запросов могут снизить производительность базы данных.

Демон автоочистки, если он включён, будет автоматически выполнять `ANALYZE` после существенных изменений содержимого таблицы. Однако администраторы могут предпочесть выполнение `ANALYZE` вручную, в частности, если известно, что производимые в таблице изменения не повлияют на статистику по «интересным» столбцам. Демон же планирует выполнение `ANALYZE` в зависимости только от количества вставленных или изменённых строк; он не знает, приведут ли они к значимым изменениям статистики.

Как и процедура очистки для высвобождения пространства, частое обновление статистики полезнее для интенсивно изменяемых таблиц, нежели для тех таблиц, которые изменяются редко. Однако даже в случае часто изменяемой таблицы обновление статистики может не требоваться, если статистическое распределение данных меняется слабо. Как правило, достаточно оценить, насколько меняются максимальное и минимальное значения в столбцах таблицы. Например, максимальное значение в столбце `timestamp`, хранящем время изменения строки, будет постоянно увеличиваться по мере добавления и изменения строк; для такого столбца может потребоваться более частое обновление статистики, чем, к примеру, для столбца, содержащего адреса страниц (URL), которые запрашивались с сайта. Столбец с URL-адресами может меняться столь же часто, однако статистическое распределение его значений, вероятно, будет изменяться относительно медленно.

Команду `ANALYZE` можно выполнять для отдельных таблиц и даже просто для отдельных столбцов таблицы, поэтому, если того требует приложение, одни статистические данные можно обновлять чаще, чем другие. Однако на практике

обычно лучше просто анализировать всю базу данных, поскольку это быстрая операция, так как `ANALYZE` читает не каждую отдельную строку, а статистически случайную выборку строк таблицы.

8.1.4. Обновление карты видимости

Процедура очистки поддерживает карты видимости для каждой таблицы, позволяющие определить, в каких страницах есть только записи, заведомо видимые для всех активных транзакций (и всех будущих транзакций, пока страница не будет изменена). Это имеет два применения.

Во-первых, сам процесс очистки может пропускать такие страницы при следующем запуске, поскольку на этих страницах вычищать нечего.

Во-вторых, с такими картами Синергия-БД может возвращать результаты некоторых запросов, используя только индекс, не обращаясь к данным таблицы. Так как индексы Синергия-БД не содержат информацию о видимости записей, при обычном сканировании по индексу необходимо извлечь соответствующую запись из таблицы и проверить её видимость для текущей транзакции. Поэтому при сканировании только индекса, наоборот, сначала проверяется карта видимости. Если известно, что все записи на странице видимы, то выборку из таблицы можно пропустить. Это наиболее полезно с большими наборами данных, когда благодаря карте видимости, можно оптимизировать чтение с диска. Карта видимости значительно меньше таблицы, поэтому она легко помещается в кеш, даже когда объём страниц очень велик.

8.1.5. Предотвращение ошибок из-за зацикливания счётчика транзакций

В Синергия-БД семантика транзакций MVCC зависит от возможности сравнения номеров идентификаторов транзакций (XID): версия строки, у которой XID добавившей её транзакции больше, чем XID текущей транзакции, относится "к будущему" и не должна быть видна в текущей транзакции. Однако поскольку идентификаторы транзакций имеют ограниченный размер (32 бита), кластер, работающий долгое время (более 4 миллиардов транзакций) столкнётся с

зацикливанием идентификаторов транзакций: счётчик XID прокрутится до нуля, и внезапно транзакции, которые относились к прошлому, окажутся в будущем - это означает, что их результаты станут невидимыми. Одним словом, это катастрофическая потеря данных. На самом деле данные никуда не пропадают, однако если вы не можете их получить. Для того чтобы этого избежать, необходимо выполнять очистку для каждой таблицы в каждой базе данных как минимум единожды на два миллиардов транзакций.

Периодическое выполнение очистки решает эту проблему, потому что процедура `VACUUM` помечает строки как замороженные, указывая, что они были вставлены транзакцией, зафиксированной достаточно давно, так что эффект добавляющей транзакции с точки зрения MVCC определённо будет виден во всех текущих и будущих транзакциях. Обычные значения XID сравниваются по модулю 232. Это означает, что для каждого обычного XID существуют два миллиарда значений XID, которые «старше» него, и два миллиарда значений, которые «младше» него; другими словами, пространство значений XID циклично и не имеет конечной точки. Следовательно, как только создаётся версия строки с обычным XID, для следующих двух миллиардов транзакций эта версия строки оказывается «в прошлом», неважно о каком значении обычного XID идет речь. Если после двух миллиардов транзакций эта версия строки всё ещё существует, она внезапно окажется в будущем. Для того чтобы это предотвратить, в какой-то момент значение XID для старых версий строк должно быть заменено на `FrozenTransactionId` (заморожено) до того, как будет достигнута граница в два миллиарда транзакций. После получения этого особенного XID для всех обычных транзакций эти версии строк будут относиться «к прошлому», независимо от зацикливания, и, таким образом, эти версии строк будут действительны до момента их удаления, когда бы это ни произошло.

Параметр `vacuum_freeze_min_age` определяет, насколько старым должен стать XID, чтобы строки с таким XID были заморожены. Увеличение его значения помогает избежать ненужной работы, если строки, которые могли бы быть заморожены в ближайшее время, будут изменены ещё раз, а уменьшение приводит к

увеличению количества транзакций, которые могут выполниться, прежде чем потребуется очередная очистка таблицы.

VACUUM определяет, какие страницы таблицы нужно сканировать, анализируя карту видимости. Обычно при этой операции пропускаются страницы, в которых нет мёртвых версий строк, даже если в них могут быть версии строк со старыми XID. Таким образом, обычная команда VACUUM не будет всегда замораживать все версии строк, имеющиеся в таблице. Периодически VACUUM будет также производить *агрессивную очистку*, пропуская только те страницы, которые не содержат ни мёртвых строк, ни незамороженных значений XID или MXID. Когда VACUUM будет делать это, зависит от параметра vacuum_freeze_table_age: полностью видимые, но не полностью замороженные страницы будут сканироваться, если число транзакций, прошедших со времени последнего такого сканирования, оказывается больше чем vacuum_freeze_table_age минус vacuum_freeze_min_age. Если vacuum_freeze_table_age равно 0, VACUUM будет применять эту более агрессивную стратегию при каждом сканировании.

Максимальное время, в течение которого таблица может обходиться без очистки, составляет два миллиарда транзакций минус значение vacuum_freeze_min_age с момента последней агрессивной очистки. Если бы таблица не подвергалась очистке дольше, была бы возможна потеря данных. Чтобы гарантировать, что это не произойдёт, для любой таблицы, которая может содержать значения XID старше, чем возраст, указанный в конфигурационном параметре autovacuum_freeze_max_age, вызывается автоочистка. Это случится, даже если автоочистка отключена.

Это означает, что если очистка таблицы не вызывается другим способом, то автоочистка для неё будет вызываться приблизительно через каждые autovacuum_freeze_max_age минус vacuum_freeze_min_age транзакций. Для таблиц, очищаемых регулярно для высвобождения пространства, это неактуально. В то же время статичные таблицы (включая таблицы, в которых данные

вставляются, но не изменяются и не удаляются) не нуждаются в очистке для высвобождения пространства, поэтому для очень больших статичных таблиц имеет смысл увеличить интервал между вынужденными запусками автоочистки. Очевидно, это можно сделать, либо увеличив autovacuum_freeze_max_age, либо уменьшив vacuum_freeze_min_age.

Фактический максимум для vacuum_freeze_table_age составляет 0.95 * autovacuum_freeze_max_age; большее значение будет ограничено этим пределом. Значение, превышающее autovacuum_freeze_max_age, не имело бы смысла, поскольку по достижении этого значения в любом случае вызывалась бы автоочистка для предотвращения зацикливания, а коэффициент 0.95 оставляет немного времени для того, чтобы запустить команду VACUUM вручную до того, как это произойдёт. Как правило, установленное значение vacuum_freeze_table_age должно быть несколько меньше autovacuum_freeze_max_age, чтобы оставленный промежуток был достаточен для выполнения в этом окне VACUUM по расписанию или автоочистки, управляемой обычной активностью операций удаления и изменения. Если это значение будет слишком близким к максимуму, автоочистка для предотвращения зацикливания будет выполняться, даже если таблица только что была очищена для высвобождения пространства, в то же время при небольшом значении будет чаще осуществляться полное сканирование таблицы.

Единственный минус увеличения autovacuum_freeze_max_age (и vacuum_freeze_table_age с ним) заключается в том, что подкаталог pg_clog в кластере базы данных будет занимать больше места, поскольку в нём нужно будет хранить статус фиксации всех транзакций вплоть до горизонта autovacuum_freeze_max_age. Для статуса фиксации используется по два бита на транзакцию, поэтому если в autovacuum_freeze_max_age установлено максимально допустимое значение в два миллиарда, то размер pg_clog может составить примерно половину гигабайта. Если по сравнению с объёмом вашей базы данных этот объём незначителен, тогда рекомендуется установить для

autovacuum_freeze_max_age максимально допустимое значение. В противном случае, установите значение этого параметра в зависимости от объёма, который вы готовы выделить для pg_clog. Значению по умолчанию, 200 миллионам транзакций, соответствует приблизительно 50 Мб в pg_clog.

Уменьшение значения vacuum_freeze_min_age, с другой стороны, чревато тем, что команда VACUUM может выполнять бесполезную работу: замораживание версии строки - пустая трата времени, если эта строка будет вскоре изменена (и в результате получит новый XID). Поэтому значение этого параметра должно быть достаточно большим для того, чтобы строки не замораживались, пока их последующее изменение не станет маловероятным.

Для отслеживания возраста самых старых значений XID в базе данных команда VACUUM сохраняет статистику по XID в системных таблицах pg_class и pg_database. В частности, столбец relfrozenxid в записи для определённой таблицы в pg_class содержит граничное значение XID, с которым в последний раз выполнялась операция VACUUM для всей этой таблицы. Все строки, добавленные транзакциями с более ранними XID, гарантированно будут заморожены. Аналогично столбец datfrozenxid в строке таблицы pg_database представляет нижнюю границу обычных значений XID, встречающихся в этой базе данных - он просто хранит минимальное из всех значений relfrozenxid для таблиц в этой базе данных. Эту информацию удобно получать с помощью таких запросов:

```
SELECT c.oid::regclass as table_name,
       greatest(age(c.relfrozenxid),age(t.relfrozenxid))
as age
FROM pg_class c
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r', 'm');
SELECT datname, age(datfrozenxid) FROM pg_database;
```

Столбец age показывает количество транзакций от граничного значения XID до XID текущей транзакции.

Обычно VACUUM сканирует только те страницы, которые изменялись после последней очистки, однако relfrozenxid может увеличиться только при полном сканировании таблицы. Сканирование всей таблицы происходит, когда возраст relfrozenxid достигает vacuum_freeze_table_age, когда VACUUM вызывается с указанием FREEZE, или, когда оказывается, что очистку для удаления мёртвых версий строк нужно провести во всех страницах. После того как VACUUM завершает сканирование всей таблицы, значение age(relfrozenxid) должно стать немного больше, чем значение vacuum_freeze_min_age (больше чем на число транзакций, начатых с момента запуска VACUUM). Если по достижении autovacuum_freeze_max_age для таблицы ни разу будет выполнена операция VACUUM с полным сканированием, в скором времени для неё будет принудительно запущена автоочистка.

Если по какой-либо причине автоочистка не может вычистить старые значения XID из таблицы, система начинает выдавать предупреждающие сообщения, подобные приведённому ниже, когда самое старое значение XID в базе данных оказывается в десяти миллионах транзакций от точки зацикливания:

```
ПРЕДУПРЕЖДЕНИЕ:  база данных "mydb" должна быть очищена
(предельное число транзакций: 177009986)
ПОДСКАЗКА:  Во избежание отключения базы данных
выполните очистку (VACUUM) всей базы "mydb".
```

Проблему можно решить, как предлагает подсказка, запустив VACUUM вручную; однако учтите, что выполнять VACUUM должен суперпользователь, в противном случае эта процедура не сможет обработать системные каталоги и, следовательно, не сможет увеличить значение datfrozenxid для базы данных. Если эти предупреждения игнорировать, система отключится и не будет начинать никаких транзакций, как только до точки зацикливания останется менее 1 миллиона транзакций:

```
ОШИБКА:  база данных не принимает команды во избежание
потери данных из-за зацикливания в БД "mydb"
```

```
ПОДСКАЗКА:  Остановите управляющий процесс (postmaster)
и выполните очистку (VACUUM) базы данных в
однопользовательском режиме.
```

Резерв в 1 миллион транзакций позволяет администратору провести восстановление без потери данных, выполнив необходимые команды VACUUM вручную. Однако, поскольку после безопасной остановки система не будет исполнять команды, администратору останется только перезапустить сервер в однопользовательском режиме, чтобы запустить VACUUM.

8.1.5.1. Мультитранзакции и зацикливание

Идентификаторы мультитранзакций используются для поддержки блокировки строк несколькими транзакциями одновременно. Поскольку в заголовке строки есть только ограниченное пространство для хранения информации о блокировках, в нём указывается «идентификатор множественной транзакции», или идентификатор мультитранзакции для краткости, когда строку блокируют одновременно несколько транзакций. Информация о том, какие именно идентификаторы транзакций относятся к определённой мультитранзакции, хранится отдельно в подкаталоге pg_multixact, а в поле xmax в заголовке строки сохраняется только идентификатор мультитранзакции. Как и идентификаторы транзакций, идентификаторы мультитранзакций исполнены в виде 32-разрядного счётчика и хранятся аналогично, что требует аккуратного управления их возрастом, очисткой хранилища и предотвращением зацикливаний. Существует отдельная область, в которой содержится список членов каждой мультитранзакции, где счётчики также 32-битные и требуют должного контроля.

Во время сканирования таблицы командой VACUUM, частичного или полного, любой идентификатор мультитранзакции, старее чем vacuum_multixact_freeze_min_age, будет заменён другим значением, которое может быть нулевым, идентификатором одиночной транзакции или новым идентификатором мультитранзакции. Для каждой таблицы в pg_class.relminmxid хранится самый старый возможный идентификатор

мультитранзакции, все ещё задействованный в какой-либо строке этой таблицы. Если это значение оказывается старее vacuum_multixact_freeze_table_age, выполняется принудительное сканирование всей таблицы. Узнать возраст pg_class.relminmxid можно с помощью функции mxid_age().

Благодаря операциям VACUUM, сканирующим таблицу целиком, вне зависимости от их причины, это значение для таблицы будет увеличиваться. В конце концов, по мере сканирования всех таблиц во всех базах данных и увеличения их старейших значений мультитранзакций, информация о старых мультитранзакциях может быть удалена с диска.

В качестве меры защиты, полное сканирование таблицы с целью очистки будет происходить для любой таблицы, возраст мультитранзакций которой больше, чем autovacuum_multixact_freeze_max_age. Операция полной очистки таблицы также будет выполняться постепенно со всеми таблицами, начиная с имеющих старейшие мультитранзакции, если объём занятой области членов мультитранзакций превышает 50% от объёма адресуемого пространства. Эти два варианта сканирования осуществляются, даже если процесс автоочистки отключён.

8.1.6. Демон автоочистки

В Синергия-БД 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» на самом деле 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.

Если в течение короткого промежутка времени потребность в очистке возникает для нескольких больших таблиц, все рабочие процессы автоочистки могут продолжительное время заниматься очисткой только этих таблиц. В результате другие таблицы и базы данных будут ожидать очистки, пока не появится свободный рабочий процесс. Число рабочих процессов для одной базы не ограничивается, при этом каждый процесс старается не повторять работу, только что выполненную другими. Заметьте, что в ограничениях max_connections или superuser_reserved_connections число выполняющихся рабочих процессов не учитывается.

Для таблиц с relfrozenxid, устаревшим более чем на autovacuum_freeze_max_age транзакций, очистка выполняется всегда (это также применимо к таблицам, для которых максимальный порог заморозки был изменён через параметры хранения; см. ниже). В противном случае, очистка таблицы производится, если количество кортежей, устаревших с момента последнего выполнения VACUUM, превышает «пороговое значение очистки». Пороговое значение очистки определяется как:

```
порог очистки = базовый порог очистки + коэффициент доли
для очистки * количество кортежей
```

где базовый порог очистки - значение autovacuum_vacuum_threshold,

коэффициент доли - autovacuum_vacuum_scale_factor, а количество кортежей - pg_class.reltuples. Количество устаревших кортежей получается от сборщика статистики; оно представляет собой приблизительное число, обновляемое после каждой операции UPDATE и DELETE. Точность не гарантируется, потому что при большой нагрузке часть информации может быть утеряна. Если значение relfrozenxid для таблицы старее vacuum_freeze_table_age транзакций, производится сканирование всей таблицы с целью заморозить старые версии строк и увеличить значение relfrozenxid, в противном случае сканируются только страницы, изменённые после последней очистки.

Для выполнения сбора статистики используется аналогичное условие: пороговое значение, определяемое как:

```
порог анализа = базовый порог анализа + коэффициент доли
для анализа * количество кортежей
```

сравнивается с общим количеством кортежей добавленных, изменённых или удалённых после последнего выполнения ANALYZE.

Автоочистка не обрабатывает временные таблицы. Поэтому очистку и сбор статистики в них нужно производить с помощью SQL-команд в обычном сеансе.

Используемые по умолчанию пороговые значения и коэффициенты берутся из postgresql.conf, однако их (и многие другие параметры, управляющие автоочисткой) можно переопределить для каждой таблицы. Если какие-либо значения определены через параметры хранения таблицы, при обработке этой таблицы действуют они, а в противном случае - глобальные параметры. За более подробной информацией о глобальных параметрах обратитесь к п. 3.10.

Когда выполняются несколько рабочих процессов, параметры задержки автоочистки по стоимости (см. п. 3.4.4) «распределяются» между всеми этими процессами, так что общее воздействие на систему остаётся неизменным, независимо от их числа. Однако этот алгоритм распределения нагрузки не учитывает процессы, обрабатывающие таблицы с индивидуальными значениями параметров хранения autovacuum_vacuum_cost_delay и

autovacuum_vacuum_cost_limit.

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.

Возможность потери пространства в индексах на основе не B-деревьев глубоко не исследовалась. Поэтому имеет смысл периодически отслеживать физический размер индекса, когда применяется индекс такого типа.

Кроме того, с B-деревьями доступ по недавно построенному индексу осуществляется немного быстрее, нежели доступ по индексу, который неоднократно изменялся, поскольку в недавно построенном индексе страницы, близкие логически, обычно расположены так же близко и физически. Это соображение неприменимо к индексам, которые основаны не на B-деревьях. Поэтому периодически проводить переиндексацию стоит хотя бы для того, чтобы увеличить скорость доступа.

Команду REINDEX можно безопасно и просто применять во всех случаях. Но так как она требует исключительной блокировки таблицы, часто предпочтительнее перестраивать индекс в несколько этапов, включающих создание и замену индекса. Типы индексов, которые поддерживает CREATE INDEX с указанием CONCURRENTLY, можно построить именно так. Если это удаётся и получен рабочий индекс, изначальный индекс можно заменить им, выполнив ALTER INDEX и DROP INDEX. Когда индекс используется для обеспечения уникальности или других

ограничений, может потребоваться команда `ALTER TABLE`, чтобы поменять существующее ограничение на то, что обеспечивает новый индекс. Обстоятельно продумайте эту многоходовую процедуру, прежде чем выполнять её, так как не все индексы можно перестроить таким образом, и предусмотрите обработку ошибок.

ПРИЛОЖЕНИЕ 1

КОДЫ ОШИБОК СУБД «СИНЕРГИЯ-БД»

Всем сообщениям, которые выдаёт СУБД «Синергия-БД», назначены пятисимвольные коды ошибок, соответствующие кодам `SQLSTATE`, описанным в стандарте SQL. Приложения, которые должны знать, какое условие ошибки имело место, обычно проверяют код ошибки и только потом обращаются к текстовому сообщению об ошибке. Заметьте, что отдельные, но не все коды ошибок, которые выдаёт Синергия-БД, определены стандартом SQL; некоторые дополнительные коды ошибок для условий, не описанных стандартом, были добавлены независимо или позаимствованы из других баз данных.

Согласно стандарту, первые два символа кода ошибки обозначают класс ошибок, а последние три символа обозначают определённое условие в этом классе. Таким образом, приложение, не знающее значение определённого кода ошибки, всё же может понять, что делать, по классу ошибки.

В таблице 1.1 перечислены все коды ошибок, определённые в Синергия-БД. Некоторые коды в настоящее время не используются, хотя они определены в стандарте SQL. Также показаны классы ошибок. Для каждого класса ошибок имеется «стандартный» код ошибки с последними тремя символами `000`. Этот код выдаётся только для таких условий ошибок, 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. Заметьте, что PL/pgSQL, в отличие от ошибок, не распознаёт предупреждения; то есть классы 00, 01 и 02.

Для некоторых типов ошибок сервер сообщает имя объекта базы данных (таблица, столбец таблицы, тип данных или ограничение), связанного с ошибкой; например, имя уникального ограничения, вызвавшего ошибку `unique_violation`. Такие имена передаются в отдельных полях сообщения об ошибке, чтобы приложениям не пришлось извлекать его из возможно

локализованного текста ошибки для человека.

Таблица 1.1 - Коды ошибок СУБД «Синергия-БД»

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
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

Продолжение таблицы 1.1

Код ошибки	Имя условия
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
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

Продолжение таблицы 1.1

Код ошибки	Имя условия
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
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

Продолжение таблицы 1.1

Код ошибки	Имя условия
2F002	modifying_sql_data_not_permitted
2F003	prohibited_sql_statement_attempted
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
42P18	indeterminate_datatype
42P21	collation_mismatch

Продолжение таблицы 1.1

Код ошибки	Имя условия
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
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

Продолжение таблицы 1.1

Код ошибки	Имя условия
55P02	cant_change_runtime_param
55P03	lock_not_available
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 Синергия-БД)	
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
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

Окончание таблицы 1.1

Код ошибки	Имя условия
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

Лист регистрации изменений									
Номера листов (страниц).					*Всего листов (страниц) в докум.*	*№ документа*	*Входящий № сопроводительного докум. и дата*	*Подп.*	*Дата*
Изм.	*измененных*	*замененных*	*новых*	*аннулированных*					

Федеральное государственное унитарное предприятие
Российский федеральный ядерный центр
Всероссийский научно-исследовательский институт экспериментальной физики

УТВЕРЖДЕН
07623615.00082-02 32 01-ЛУ

КОМПЛЕКС ПРОГРАММ В ЗАЩИЩЕННОМ ИСПОЛНЕНИИ
«СИСТЕМА ПОЛНОГО ЖИЗНЕННОГО ЦИКЛА ИЗДЕЛИЙ
«ЦИФРОВОЕ ПРЕДПРИЯТИЕ»

Программный модуль
«Система управления базами данных «Синергия-БД»

Руководство системного программиста

Часть 2

07623615.00082-02 32 01-2

Листов 90

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

2021

СОДЕРЖАНИЕ

1. РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ

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

Существует три фундаментально разных подхода к резервному копированию данных в Синергия-БД:

1. Выгрузка в SQL.

2. Копирование на уровне файлов.

3. Непрерывное архивирование.

Каждый из них имеет свои сильные и слабые стороны; все они обсуждаются в следующих разделах.

1.1. Выгрузка в SQL

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

```
pg_dump имя_базы > выходной_файл
```

Как видите, `pg_dump` записывает результаты своей работы в устройство стандартного вывода. Далее будет рассмотрено, чем это может быть полезно. В то время как вышеупомянутая команда создаёт текстовый файл, `pg_dump` может создать файлы и в других форматах, которые допускают параллельную обработку и более гибкое управление восстановлением объектов.

Программа `pg_dump` является для Синергия-БД обычным клиентским приложением. Это означает, что вы можете выполнять процедуру резервного копирования с любого удалённого компьютера, если имеете доступ к нужной базе данных. Но помните, что `pg_dump` не использует для своей работы какие-то специальные привилегии. В частности, ей обычно требуется доступ на чтение всех таблиц, которые вы хотите выгрузить, так что для копирования всей базы данных

практически всегда её нужно запускать с правами суперпользователя СУБД. Если у вас нет достаточных прав для резервного копирования всей базы данных, вы, тем не менее, можете сделать резервную копию той части базы, доступ к которой у вас есть, используя такие параметры, как `-n` схема или `-t` таблица.

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

Как и любое другое клиентское приложение Синергия-БД, `pg_dump` по умолчанию будет подключаться к базе данных с именем пользователя, совпадающим с именем текущего пользователя операционной системы. Чтобы переопределить имя, либо добавьте параметр `-U`, либо установите переменную окружения `PGUSER`. Помните, что `pg_dump` подключается к серверу через обычные механизмы проверки подлинности клиента (которые описываются в п. 4. Часть 1. Руководство системного программиста).

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

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

например, большинство форм команды ALTER TABLE.

1.1.1. Восстановление дампа

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

```
psql имя_базы < входной_файл
```

где входной_файл - это файл, содержащий вывод команды pg_dump. База данных, заданная параметром имя_базы, не будет создана данной командой, так что вы должны создать её сами из базы template0 перед запуском psql (например, с помощью команды createdb -T template0 имя_базы). Программа psql принимает параметры, указывающие сервер, к которому осуществляется подключение, и имя пользователя, подобно pg_dump. За дополнительными сведениями обратитесь к справке по pg_restore. Копии не текстовые восстанавливаются утилитой pg_restore.

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

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

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

В любом случае, вы получите только частично восстановленную базу данных. В качестве альтернативы можно указать, что весь дамп должен быть восстановлен в одной транзакции, так что восстановление либо полностью выполнится, либо полностью отменится. Включить данный режим можно, передав psql аргумент -1 или --single-transaction. Выбирая этот режим, учтите, что даже незначительная ошибка может привести к откату восстановления, которое могло

продолжаться несколько часов. Однако, это всё же может быть предпочтительней, чем вручную вычищать сложную базу данных после частично восстановленного дампа.

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
```

Дампы, которые выдаёт pg_dump, содержат определения относительно template0. Это означает, что любые языки, процедуры и т. п., добавленные в базу через template1, pg_dump также выгрузит в дамп. Как следствие, если при восстановлении вы используете модифицированный template1, вы должны создать пустую базу данных из template0, как показано в примере выше.

После восстановления резервной копии имеет смысл запустить ANALYZE для каждой базы данных, чтобы оптимизатор запросов получил полезную статистику; за подробностями обратитесь к п. 8.1.3 (Часть 1. Руководство системного программиста) и п. 8.1.6 (Часть 1. Руководство системного программиста).

1.1.2. 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. Простое использование этой команды:

```
pg_dumpall > выходной_файл
```

Полученную копию можно восстановить с помощью psql:

```
psql -f входной_файл postgres
```

В принципе, здесь в качестве начальной базы данных можно указать имя любой существующей базы, но, если вы загружаете копию в пустой кластер, обычно

нужно использовать postgres. Восстанавливать дамп, который выдала pg_dumpall, всегда необходимо с правами суперпользователя, так как они требуются для восстановления информации о ролях и табличных пространствах. Если вы используете табличные пространства, убедитесь, что пути к табличным пространствам в дампе соответствуют новой среде.

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

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

1.2. Резервное копирование на уровне файлов

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

```
tar -cf backup.tar /var/lib/sdb/sdb-11/data
```

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

- чтобы полученная резервная копия была годной, сервер баз данных должен быть остановлен. Такие полумеры, как запрещение всех подключений к серверу, работать не будут (отчасти потому что tar и подобные средства не получают мгновенный снимок состояния файловой системы, но ещё и потому, что в сервере есть внутренние буферы). Необходимо отметить, что сервер нужно будет остановить и перед восстановлением данных;
- если вы ознакомились с внутренней организацией базы данных в файловой системе, у вас может возникнуть соблазн скопировать или восстановить только

отдельные таблицы или базы данных в соответствующих файлах или каталогах. Это не будет работать, потому что информацию, содержащуюся в этих файлах, нельзя использовать без файлов журналов транзакций, `pg_clog/*`, которые содержат состояние всех транзакций. Без этих данных файлы таблиц непригодны к использованию. Разумеется, также невозможно восстановить только одну таблицу и соответствующие данные `pg_clog`, потому что в результате нерабочими станут все другие таблицы в кластере баз данных. Таким образом, копирование на уровне файловой системы будет работать, только если выполняется полное копирование и восстановление всего кластера баз данных.

Ещё один подход к резервному копированию файловой системы заключается в создании «целостного снимка» каталога с данными, если это поддерживает файловая система. Типичная процедура включает создание «замороженного снимка» тома, содержащего базу данных, затем копирование всего каталога с данными (а не его избранных частей, см. выше) из этого снимка на устройство резервного копирования, и наконец освобождение замороженного снимка. При этом сервер базы данных может не прекращать свою работу. Однако резервная копия, созданная таким способом, содержит файлы базы данных в таком состоянии, как если бы сервер баз данных не был остановлен штатным образом; таким образом, когда вы запустите сервер баз данных с сохранёнными данными, он будет считать, что до этого процесс сервера был прерван аварийно, и будет накатывать журнал WAL. Это не проблема, просто имейте это в виду и обязательно включите файлы WAL в резервную копию. Чтобы сократить время восстановления, можно выполнить команду `CHECKPOINT` перед созданием снимка.

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

документацию по вашей файловой системе, прежде чем довериться технологии согласованных снимков.

Если одновременные снимки невозможны, остаётся вариант с остановкой сервера баз данных на время, достаточное для получения всех замороженных снимков. Другое возможное решение - получить базовую копию путём непрерывного архивирования (см. п. 1.3.2), такие резервные копии не могут пострадать от изменений файловой системы в процессе резервного копирования. Для этого требуется включить непрерывное архивирование только на время резервного копирования; для восстановления применяется процедура восстановления из непрерывного архива.

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

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

1.3. Непрерывное архивирование и восстановление на момент времени

Всё время в процессе работы Синергия-БД 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. Прежде всего, журнал существует для безопасного восстановления после краха сервера: если происходит крах, целостность СУБД может быть восстановлена в

результате «воспроизведения» записей, зафиксированных после последней контрольной точки. Однако наличие журнала делает возможным использование третьей стратегии копирования баз данных: можно сочетать резервное копирование на уровне файловой системы с копированием файлов WAL. Если потребуется восстановить данные, мы можем восстановить копию файлов, а затем воспроизвести журнал из скопированных файлов WAL, и таким образом привести систему в нужное состояние. Такой подход более сложен для администрирования, чем любой из описанных выше, но он имеет значительные преимущества:

- в качестве начальной точки для восстановления необязательно иметь полностью согласованную копию на уровне файлов. Внутренняя несогласованность копии будет исправлена при воспроизведении журнала (практически то же самое происходит при восстановлении после краха). Таким образом, согласованный снимок файловой системы не требуется, вполне можно использовать `tar` или похожие средства архивации;
- поскольку при воспроизведении можно обрабатывать неограниченную последовательность файлов WAL, непрерывную резервную копию можно получить, просто продолжая архивировать файлы WAL. Это особенно ценно для больших баз данных, полные резервные копии которых делать как минимум неудобно;
- воспроизводить все записи WAL до самого конца нет необходимости. Воспроизведение можно остановить в любой точке и получить целостный снимок базы данных на этот момент времени. Таким образом, данная технология поддерживает восстановление на момент времени: можно восстановить состояние базы данных на любое время с момента создания резервной копии;
- если непрерывно передавать последовательность файлов WAL другому серверу, получившему данные из базовой копии того же кластера, получается система тёплого резерва: в любой момент мы можем запустить второй сервер, и он будет иметь практически текущую копию баз данных.

Как и обычное резервное копирование файловой системы, этот метод

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

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

1.3.1. Восстановление непрерывной архивной копии

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

- остановите сервер баз данных, если он запущен;
- если у вас есть место для этого, скопируйте весь текущий каталог кластера баз данных и все табличные пространства во временный каталог на случай, если они вам понадобятся. Учтите, что эта мера предосторожности требует, чтобы свободного места на диске было достаточно для размещения двух копий существующих данных. Если места недостаточно, необходимо сохранить как минимум содержимое подкаталога `pg_wal` каталога кластера, так как он может содержать журналы, не попавшие в архив перед остановкой системы;
- удалите все существующие файлы и подкаталоги из каталога кластера и из корневых каталогов используемых табличных пространств;
- восстановите файлы базы данных из архивной копии файлов. Важно, чтобы у восстановленных файлов были правильные разрешения и правильный владелец (пользователь, запускающий сервер, а не root!). Если вы используете

табличные пространства, убедитесь также, что символьные ссылки в `pg_tblspc/` восстановились корректно;

- удалите все файлы из `pg_wal/`; они восстановились из резервной копии файлов и поэтому, скорее всего, будут старее текущих. Если вы вовсе не архивировали `pg_wal/`, создайте этот каталог с правильными правами доступа, но, если это была символьная ссылка, восстановите её;
- если на вротором шаге вы сохранили незаархивированные файлы с сегментами WAL, скопируйте их в `pg_wal/`. Лучше всего именно копировать, а не перемещать их, чтобы у вас остались неизменённые файлы на случай, если возникнет проблема и всё придётся начинать сначала;
- создайте командный файл восстановления `recovery.conf` в каталоге кластера баз данных (см. п. 3). Вы можете также временно изменить `pg_hba.conf`, чтобы обычные пользователи не могли подключаться, пока вы не будете уверены, что восстановление завершилось успешно;
- запустите сервер. Сервер запустится в режиме восстановления и начнёт считывать необходимые ему архивные файлы WAL. Если восстановление будет прервано из-за внешней ошибки, сервер можно просто перезапустить, и он продолжит восстановление. По завершении процесса восстановления сервер переименует файл `recovery.conf` в `recovery.done` (чтобы предотвратить повторный запуск режима восстановления), а затем перейдёт к обычной работе с базой данных;
- просмотрите содержимое базы данных, чтобы убедиться, что вы вернули её к желаемому состоянию. Если это не так, вернитесь к первому шагу. Если всё хорошо, разрешите пользователям подключаться к серверу, восстановив обычный файл `pg_hba.conf`.

Ключевой момент этой процедуры заключается в создании файла конфигурации восстановления, описывающего, как будет выполняться восстановление и до какой точки. В качестве прототипа вы можете использовать файл `recovery.conf.sample` (он обычно помещается в каталог `share/` после установки). Единственное, что совершенно необходимо указать в

recovery.conf - это команду restore_command, которая говорит Синергия-БД, как получать из архива файл-сегменты WAL. Как и archive_command, это командная строка для оболочки. Она может содержать символы %f, которые заменятся именем требующегося файла журнала, и %p, которые заменятся целевым путём для копирования этого файла. Путь задаётся относительно текущего рабочего каталога, т. е. каталога кластера данных. Если вам нужно включить в команду сам символ %, напишите %%. Простейшая команда, которая может быть полезна, такая:

```
restore_command = 'cp /mnt/server/archivedir/%f %p'
```

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

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

Не все запрашиваемые файлы будут сегментами WAL; следует также ожидать запросов файлов с суффиксом .backup или .history. Также учтите, что базовое имя пути %p будет отличаться от %f; не думайте, что они взаимозаменяемы.

Сегменты WAL, которые не найдутся в архиве, система будет искать в pg_wal/; благодаря этому можно использовать последние незаархивированные сегменты. Однако файлы в pg_wal/ будут менее предпочтительными, если такие сегменты окажутся в архиве.

Обычно при восстановлении обрабатываются все доступные сегменты WAL и, таким образом, база данных восстанавливается до последнего момента времени (или максимально близкого к нему, в зависимости от наличия сегментов WAL). Таким

образом, восстановление обычно завершается с сообщением «файл не найден»; точный текст сообщения об ошибке зависит от того, что делает `restore_command`. Вы также можете увидеть сообщение об ошибке в начале восстановления для файла с именем типа `00000001.history`. Это также нормально и обычно не говорит о какой-либо проблеме при восстановлении в простых ситуациях; подробнее об этом рассказывается в п. 1.3.5.

Если вы хотите восстановить базу на какой-то момент времени (скажем, до момента, когда неопытный администратор базы данных удалил основную таблицу транзакций), просто укажите требуемую точку остановки в `recovery.conf`. Вы можете задать точку останова, так называемую «цель восстановления», по дате/времени, именованной точке восстановления или определённому идентификатору транзакции. На момент написания этой документации полезными могут быть только указания даты/времени или имени точки восстановления, пока нет никаких средств, позволяющих точно определить, какой идентификатор транзакции нужно выбрать.

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

1.3.2. Линии времени

Возможность восстановить базу данных на некий предыдущий момент времени создаёт некоторые сложности. Например, предположим, что в начальной

истории базы данных вы удалили важную таблицу в 17:15 во вторник, но осознали эту ошибку только в среду в полдень. Вы можете спокойно взять резервную копию, восстановить данные на 17:14 во вторник и запустить сервер. В этой истории мира базы данных вы никогда не удаляли вышеупомянутую таблицу. Но предположим, что позже вы заметили, что это была не такая уж хорошая идея и захотели вернуться к утру среды в первоначальной истории базы данных. Вы не сможете сделать это, если в процессе работы базы данных она успеет перезаписать какие-либо файлы-сегменты WAL, приводящие к моменту времени, к которому вы хотите вернуться теперь. Таким образом, для получения желаемого результата необходимо как-то отличать последовательности записей WAL, добавленные после восстановления на какой-то момент времени от тех, что существовали в начальной истории базы данных.

Для решения этой проблемы в Синергия-БД есть такое понятие, как линия времени. Всякий раз, когда завершается восстановление из архива, создаётся новая линия времени, позволяющая идентифицировать последовательность записей WAL, добавленных после этого восстановления. Номер линии времени включается в имя файлов-сегментов WAL, так что файлы новой линии времени не перезаписывают файлы WAL, сгенерированные предыдущими линиями времени. Фактически это позволяет архивировать много различных линий времени. Хотя это может показаться бесполезной возможностью, на самом деле она часто бывает спасительной. Представьте, что вы не определились, какую точку времени выбрать для восстановления, и таким образом должны проводить восстановление методом проб и ошибок, пока не найдёте лучший момент для ответвления от старой истории. Без линий времени этот процесс быстро стал бы очень запутанным. А благодаря линиям времени, вы можете вернуться к любому предыдущему состоянию, включая состояния в ветках линий времени, покинутых ранее.

Каждый раз, когда образуется новая линия времени, Синергия-БД создаёт файл «истории линии времени», показывающий, от какой линии времени ответвилась данная и когда. Эти файлы истории нужны, чтобы система могла выбрать правильные файлы-сегменты WAL при восстановлении из архива, содержащего

несколько линий времени. Таким образом, они помещаются в область архивов WAL так же, как и файлы сегментов WAL. Файлы истории представляют собой небольшие текстовые файлы, так что они не занимают много места и их вполне можно сохранять неограниченно долго (в отличие от файлов сегментов, имеющих большой размер). Если хотите, вы можете добавлять в файл истории комментарии, свои собственные заметки о том, как и почему была создана эта конкретная линия времени. Такие комментарии будут особенно ценны, если в результате экспериментов у вас образуется хитросплетение разных линий времени.

По умолчанию при восстановлении восстанавливается та же линия времени, которая была текущей при создании базовой резервной копии. Если вы хотите восстановить состояние на какой-либо дочерней линии времени, (то есть, хотите вернуться к некоторому состоянию, которое тоже было получено в результате попытки восстановления), вам необходимо указать идентификатор целевой линии времени в `recovery.conf`. Восстановить состояние в линии времени, ответвившейся раньше, чем была сделана базовая резервная копия, нельзя.

2. КОНФИГУРАЦИЯ ВОССТАНОВЛЕНИЯ

В разделе описаны параметры, задаваемые в файле `recovery.conf`. Они применяются лишь во время восстановления и должны заново устанавливаться перед каждым последующим восстановлением. После начала процесса они не могут быть изменены.

Параметры в файле `recovery.conf` указываются в формате `name = 'value'`. Каждый параметр должен располагаться на отдельной строке. Для строчного комментария используется хеш (`#`). Чтобы включить апостроф в значение параметра, продублируйте его (`''`).

Пример файла `share/recovery.conf.sample`, поставляемый в рамках дистрибутива, размещён в каталоге `share/`.

2.1. Параметры восстановления из архива

`restore_command (string)`

Это команда оболочки ОС, которая выполняется для извлечения архивного сегмента 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 при копировании 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 необходимо хранить для возможности восстановления, и эту информацию можно использовать для усечения архива в целях его минимизации. `%r` обычно используется при организации тёплого резерва (см. п. 2.2). Для того чтобы указать символ `%`, продублируйте его (`%%`).

Обратите внимание, что команда должна возвращать ноль на выходе лишь в случае успешного выполнения. У команды будет запрошен список имён файлов, которые отсутствуют в архиве; в этом случае она должна возвращать ненулевой статус, например:

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
```

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

archive_cleanup_command (string)

Этот необязательный параметр указывает команду оболочки ОС, которая будет вызываться при каждой точке перезапуска. Назначение команды archive_cleanup_command в том, чтобы предоставить механизм очистки от старых архивных файлов WAL, которые более не нужны на резервном сервере. Любое вхождение %r заменяется на имя файла, содержащего последнюю действительную точку восстановления. Это самый ранний файл, который необходимо хранить для возможности восстановления, а более старые файлы, чем %r, могут быть безболезненно удалены. Эта информация может быть использована для усечения архива с целью его минимизации при сохранении возможности последующего восстановления из заданной точки. Модуль pg_archivecleanup часто используется в качестве значения параметра archive_cleanup_command в конфигурациях с одним резервным сервером, например:

```
archive_cleanup_command = 'pg_archivecleanup
/mnt/server/archivedir %r'
```

Стоит обратить внимание, что в конфигурациях с множеством резервных серверов, использующих общий архивный каталог для восстановления, необходимо контролировать удаление файлов WAL, так как для некоторых они ещё могут требоваться. Поэтому archive_cleanup_command обычно используется при организации тёплого резерва (см. п. 2.2). Для того чтобы указать символ % в команде, он пишется дважды %%.

Если команда вернёт ненулевой статус завершения, то в журнал будет записано предупреждающее сообщение. В ситуации, когда команда прерывается сигналом или в случае ошибки оболочки ОС (например, команда не найдена), будет вызвана фатальная ошибка.

recovery_end_command (string)

Этот необязательный параметр указывает команду оболочки, которая будет выполнена единожды в конце процесса восстановления. Назначение параметра `recovery_end_command` в предоставлении механизма очистки, следующего за репликацией или восстановлением. Любое вхождение `%r` заменяется именем файла, содержащим последнюю действительную точку восстановления, например, как в `archive_cleanup_command`.

Если команда вернёт ненулевой статус завершения, то в журнал будет записано предупреждающее сообщение, но при этом запуск сервера будет продолжен. За исключением случаев, когда команда прервана сигналом или при возникновении ошибки оболочки ОС (например, команда не найдена). В таких случаях база данных не будет запускаться.

2.2. Параметры управления восстановлением

По умолчанию процесс восстановления производится вплоть до окончания журналов WAL. Нижеуказанные параметры могут использоваться, чтобы остановить процесс восстановления в более ранней точке:

- `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.

Технически, это строковый параметр, но значение `'immediate'` единственно допустимое в данный момент;

- `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. Точность этой точки останова также зависит

от recovery_target_inclusive;

- recovery_target_xid (string)

Параметр указывает идентификатор транзакции, вплоть до которой необходимо произвести процедуру восстановления. Имейте в виду, что несмотря на то, что при старте идентификаторы транзакций назначаются последовательно, завершаться они могут в ином порядке. Восстанавливаемые транзакции это те, что были зафиксированы до указанной (и, возможно, включая её). Точность точки останова также зависит от recovery_target_inclusive;

- recovery_target_lsn (pg_lsn)

Данный параметр указывает LSN позиции в журнале предзаписи, до которой должно выполняться восстановление. Точная позиция остановки зависит также от параметра recovery_target_inclusive. Данный параметр принимает значение системного типа данных pg_lsn.

Лишь один из параметров recovery_target, recovery_target_name, recovery_target_time, или recovery_target_xid может быть использован; если в конфигурационном файле их несколько, то будет использоваться последний.

Следующие параметры уточняют целевую точку восстановления и оказывают влияние на процесс при её достижении:

- recovery_target_inclusive (boolean)

Указывает на необходимость остановки сразу после (true), либо до (false) достижения целевой точки. Применяется одновременно с recovery_target_time, либо recovery_target_xid. Этот параметр управляет тем, нужно ли восстанавливать транзакции, у которых время фиксации либо идентификатор в точности совпадает со значением соответствующего параметра. Значение по умолчанию – true;

- recovery_target_timeline (string)

Указывает линию времени для восстановления. По умолчанию производится

восстановление той же линии времени, которая была текущей в момент создания базовой резервной копии. Со значением `latest` восстанавливаться будет последняя линия времени, найденная в архиве, что полезно для резервного сервера. Иное значение параметра может потребоваться в более сложной ситуации повторного восстановления, когда необходимо вернуться к состоянию, которое само было достигнуто после восстановления на момент времени(см. п. 1.3.5);

- `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 базе данных 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 же окажется, что мы ещё не достигли желаемой точки восстановления, 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 (и на самом деле ему придётся воспроизводить записи WAL после последней контрольной точки при следующем перезапуске).

Заметьте, что так как `recovery.conf` не переименовывается, когда в `recovery_target_action` выбран вариант `shutdown`, при последующем запуске будет происходить немедленная остановка, пока вы не измените конфигурацию или не удалите файл `recovery.conf` вручную.

Этот параметр не действует, если цель восстановления не установлена. Если

не включён режим hot_standby, значение pause действует так же, как и shutdown.

2.3. Параметры резервного сервера

standby_mode (boolean)

Указывает, является ли сервер Синергия-БД резервным или нет. Если параметр установлен в on, то сервер не прекратит восстановление по окончании последнего архивного файла WAL, а продолжит попытки извлечения новых сегментов WAL посредством команды restore_command и/или через подключение к ведущему, как указано в параметре primary_conninfo.

primary_conninfo (string)

Указывает строку подключения резервного сервера к ведущему. Вместо опущенных параметров подключения используются соответствующие переменные окружения. Если же какие-то переменные не установлены, то используются значения по умолчанию.

Строка может содержать имя (или адрес) основного сервера и порт подключения. Также можно указать необходимого пользователя на ведущем сервере с требуемыми привилегиями (см. п. 2.2.5.1). Если настроена аутентификация по паролю, то его также необходимо указать. Пароль можно указать как в строке подключения primary_conninfo, так и в файле ~/.pgpass резервного сервера (в качестве имени базы данных используйте replication). Не указывайте имя базы данных среди параметров подключения primary_conninfo.

Этот параметр не действует, если standby_mode имеет значение off.

primary_slot_name (string)

Дополнительно указывает заранее созданный слот при потоковой репликации для контроля очистки ресурсов подключённого узла (см. п. 2.2.6). Этот параметр не действует, если не указана строка подключения primary_conninfo.

trigger_file (string)

Указывает триггерный файл, наличие которого завершает работу в режиме резервирования. Даже если это значение не установлено, существует возможность

назначить резервный сервер основным с помощью команды `pg_ctl promote`. Этот параметр не действует, если параметр `standby_mode` имеет значение `off`.

`recovery_min_apply_delay (integer)`

По умолчанию резервный сервер восстанавливает записи WAL ведущего настолько быстро, насколько это возможно. Иногда полезно иметь возможность задать задержку при копировании данных, например, для устранения ошибок, связанных с потерей данных. Параметр позволяет задать эту задержку, указав период времени в миллисекундах (по умолчанию) или иных единицах измерения. Например, если установить значение `5min`, резервный сервер будет воспроизводить фиксацию транзакции не раньше, чем через 5 минут (судя по его системным часам) после времени фиксации, сообщённого главным.

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

Задержка применяется лишь для записей WAL, относящимся к зафиксированным транзакциям. 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 база данных достигает согласованного состояния, и исключается, когда резервный сервер переключается в режим основного. С момента переключения резервный сервер завершает восстановление незамедлительно.

Данный параметр предназначен для применения в конфигурациях с потоковой

репликацией; однако, если он задан, он будет учитываться в любых случаях. На синхронную репликацию этот параметр не влияет, пока нет никакого параметра для запроса синхронного применения фиксирования транзакций. Задержка, устанавливаемая этим параметром, распространяется и на сообщения `hot_standby_feedback`, что может привести к раздуванию базы на главном сервере; сочетание этих параметров требует осторожности.

3. СЕРВЕРНЫЕ ПРОГРАММЫ СУБД «СИНЕРГИЯ-БД»

3.1. Программа initdb

Initdb - создать кластер баз данных Синергия-БД.

3.1.1. Синтаксис

```
initdb [ параметр ...] [ --pgdata | -D ]каталог
```

3.1.2. Описание

Команда `initdb` создаёт новый кластер баз данных Синергия-БД. Кластер - это коллекция баз данных под управлением единого экземпляра сервера.

Инициализация кластера базы данных заключается в создании каталогов для хранения данных, формировании общих системных таблиц (относящихся ко всему кластеру, а не к какой-либо базе) и создании баз данных `template1` и `postgres`. Впоследствии все новые базы создаются на основе шаблона `template1` (все дополнения, установленные в `template1` автоматически копируются в каждую новую базу данных). База `postgres` используется пользователями, утилитами и сторонними приложениями по умолчанию.

При попытке создать каталог для хранения данных `initdb` может столкнуться с нехваткой прав доступа, если этот каталог принадлежит суперпользователю `root`. В таком случае необходимо назначить пользователя базы данных владельцем этого каталога при помощи `chown`. Затем выполнить `su` для смены пользователя и дальнейшего выполнения `initdb`.

Команда `initdb` должна выполняться от имени пользователя, под которым будет запускаться сервер, так как ему необходим полный доступ к файлам и каталогам, создаваемым `initdb`. Сервер не может запускаться от имени суперпользователя, поэтому выполнение команды `initdb` от его лица будет отклонено.

`initdb` инициализирует локали и кодировки баз данных кластера, которые будут использоваться по умолчанию. Кодировка, порядок сортировки

(LC_COLLATE), классы наборов символов (LC_CTYPE, например, заглавные, строчные буквы, цифры) могут устанавливаться раздельно при создании новой базы данных. initdb определяет параметры локали для шаблона template1, которые будут применяться по умолчанию для новых баз.

Чтобы изменить порядок сортировки по умолчанию или классы наборов символов, используются параметры --lc-collate и --lc-ctype. Порядок сортировки, отличающийся от C или POSIX, оказывает влияние на производительность. Поэтому необходимо тщательно выбирать необходимую и достаточную локаль при выполнении initdb.

Другие категории локали можно изменить и после старта сервера. Также можно использовать параметр --locale, чтобы задать локаль для всех категорий одновременно, включая порядок сортировки и классы наборов символов. Значения локалей сервера (lc_*) можно вывести командой SHOW ALL. Узнать об этом больше можно в п. 7.1 (Часть 1. Руководство системного программиста).

Для изменения кодировки по умолчанию используется параметр --encoding. Узнать об этом больше можно в п. 7.3 (Часть 1. Руководство системного программиста).

3.1.3. Параметры

-A authmethod

--auth=authmethod

Параметр определяет метод аутентификации для локальных пользователей, указанный в файле pg_hba.conf (строки host и local). Не используйте trust, если не можете доверять всем локальным пользователям в вашей системе. Режим trust используется по умолчанию для облегчения процесса установки.

--auth-host=authmethod

Параметр указывает метод аутентификации для локальных пользователей, подключающихся по TCP/IP, используемый в pg_hba.conf (строки host).

--auth-local=authmethod

Параметр указывает метод аутентификации локальных пользователей

подключающихся по доменным Unix-сокетам, используемый в pg_hba.conf (строки local).

-D каталог

--pgdata=каталог

Параметр указывает каталог хранения данных кластера. Это единственный обязательный параметр для команды initdb. При этом его можно указать в переменной окружения PGDATA, что будет удобным при дальнейшем использовании (postgres обращается к этой же переменной).

-E кодировка

--encoding=кодировка

Устанавливает кодировку шаблона и новых баз данных по умолчанию, если не указать иное при их создании. По умолчанию устанавливается исходя из указанной локали, и далее, если не удалось определить, выбирается SQL_ASCII. Кодировки, поддерживаемые сервером Синергия-БД, описаны в п. 7.3.1 (Часть 1. Руководство системного программиста).

-k

--data-checksums

Применять контрольные суммы на страницах данных для выявления сбоев при вводе/выводе, которые иначе останутся незамеченными. Расчёт контрольных сумм может значительно повлияет на производительность. Этот режим можно включить только при инициализации и нельзя изменить позже. Когда контрольные суммы включены, они рассчитываются для всех объектов и во всех базах данных.

--locale=локаль

Устанавливает локаль кластера по умолчанию. Если флаг не указан, локаль устанавливается согласно окружению, в котором исполняется команда initdb. Поддерживаемые локали описаны в п. 7.1 (Часть 1. Руководство системного программиста).

--lc-collate=локаль

--lc-ctype=локаль

--lc-messages=локаль

--lc-monetary=локаль

--lc-numeric=локаль

--lc-time=локаль

Аналогично --locale устанавливает необходимую локаль, но в заданной категории.

--no-locale

Аналогично флагу --locale=C.

-N

--nosync

По умолчанию initdb ждёт, пока все файлы не будут надёжно записаны на диск. С данным параметром initdb завершается немедленно, то есть выполняется быстрее, но в случае неожиданного сбоя операционной системы каталог данных может оказаться испорченным. Вообще этот параметр предназначен прежде всего для тестирования, для производственной среды он не подходит.

--pwfile=имя_файла

Принуждает initdb читать пароль суперпользователя базы данных из файла, первая строка которого используется в качестве пароля.

-S

--sync-only

Безопасно записывает все файлы базы на диск и останавливается. Другие операции initdb при этом не выполняются.

-T CFG

--text-search-config=CFG

Устанавливает конфигурацию текстового поиска по умолчанию.

-U имя_пользователя

--username=имя_пользователя

Устанавливает имя суперпользователя базы данных. По умолчанию используется имя пользователя ОС, запустившего initdb. По факту, само по себе имя суперпользователя базы данных не важно, но этот параметр позволяет оставить привычное postgres, если имя пользователя ОС другое.

-W

--pwprompt

Указывает initdb запросить пароль, который будет назначен суперпользователю базы данных. Это не важно, если не планируется использовать аутентификацию по паролю. В ином случае этот режим аутентификации оказывается неприменимым, пока пароль не задан.

-X каталог

--xlogdir=каталог

Флаг указывает каталог для хранения журналов транзакций.

Другие реже используемые параметры описаны здесь:

-d

--debug

Выводит отладочные сообщения загрузчика и ряд других сообщений, не очень интересных широкой публике. Загрузчик - это приложение initdb, используемое для создания каталога таблиц. С этим параметром выдаётся очень много крайне скучных сообщений.

-L каталог

Указывает initdb, где необходимо искать входные файлы для развёртывания кластера. Обычно это не требуется. Приложение само запросит эти данные, если будет необходимо.

-n

--noclean

По умолчанию, при выявлении ошибки на этапе развёртывания кластера, initdb удаляет все файлы, которые к тому моменту были созданы. Параметр предотвращает очистку файлов для целей отладки.

Прочие параметры:

-V

--version

Выводит версию initdb и останавливается.

-?

--help

Показывает помощь по аргументам команды initdb и останавливается.

3.1.4. Переменные окружения

PGDATA

Указывает каталог хранения данных кластера, можно изменить параметром -D.

TZ

Указывает часовой пояс кластера по умолчанию. Значение - это полное имя часового пояса.

Эта утилита, как и большинство других утилит Синергия-БД, также использует переменные среды, поддерживаемые libpq.

3.1.5. Замечания

initdb можно выполнить командой pg_ctl initdb.

3.2. Программа pg_archivecleanup

Pg_archivecleanup - вычистить файлы архивов WAL Синергия-БД.

3.2.1. Синтаксис

```
pg_archivecleanup [ параметр ...] расположение_архива
старейший_сохраняемый_файл
```

3.2.2. Описание

Утилита pg_archivecleanup предназначена для использования в качестве archive_cleanup_command для удаления старых файлов WAL на резервном сервере (см. п. 2.2). pg_archivecleanup можно использовать и как отдельную программу для выполнения тех же действий.

Чтобы использовать pg_archivecleanup на резервном сервере, добавьте эту строку в файл конфигурации recovery.conf:

```
archive_cleanup_command = 'pg_archivecleanup
```

```
расположение_архива %r'
```

Здесь расположение_архива определяет каталог, из которого должны удаляться файлы сегментов WAL.

Вызываемая в качестве `archive_cleanup_command`, эта программа просматривает `расположение_архива` и удаляет все файлы WAL, логически предшествующие значению аргумента `%r`. Целью этой операции является сокращение числа сохраняемых файлов без потери возможности восстановления при перезапуске. Такой вариант использования уместен, когда `расположение_архива` указывает на область рабочих файлов конкретного резервного сервера, но не когда `расположение_архива` - каталог с архивом WAL для долговременного хранения, или когда несколько резервных серверов восстанавливают записи WAL из одного расположения.

При отдельном использовании этой программы из каталога `расположение_архива` будут удалены все файлы WAL, логически предшествующие файлу `старейший_сохраняемый_файл`. В этом режиме, если вы укажете имя файла с расширением `.partial` или `.backup`, `старейший_сохраняемый_файл` будет определяться по имени без расширения. Благодаря такой интерпретации расширения `.backup` будут корректно удалены все файлы WAL, заархивированные до определённой базовой копии. Например, следующая команда удалит все файлы старее файла WAL с именем `000000010000003700000010`:

```
pg_archivecleanup -d archive
000000010000003700000010.00000020.backup
pg_archivecleanup:  keep WAL file
"archive/000000010000003700000010" and later
pg_archivecleanup:  removing file
"archive/00000001000000370000000F"
pg_archivecleanup:  removing file
"archive/00000001000000370000000E"
```

`pg_archivecleanup` рассчитывает на то, что расположение_архива

доступно для чтения и записи пользователю, владеющему серверным процессом.

3.2.3. Параметры

`pg_archivecleanup` принимает следующие аргументы командной строки:

`-d`

Выводить подробные отладочные сообщения в `stderr`.

`-n`

Вывести имена файлов, которые должны быть удалены, в `stdout` (не выполняя удаление).

`-V`

`--version`

Вывести версию `pg_archivecleanup` и завершиться.

`-x расширение`

При отдельном использовании этой программы укажите расширение, которое будет убрано из имён файлов до принятия решения об удалении определённых файлов. Это обычно полезно для очистки файлов архивов, которые сжимаются для хранения и получают расширение, задаваемое программой сжатия, например: `-x .gz`.

`-?`

`--help`

Вывести справку об аргументах командной строки `pg_archivecleanup` и завершиться.

3.2.4. Замечания

Программа `pg_archivecleanup` рассчитана на работу как команда очистки архива.

Программа `pg_archivecleanup` написана на C; её исходный код легко поддаётся модификации (он содержит секции, предназначенные для изменения при надобности)

3.2.5. Примеры

В системах Linux или Unix можно использовать команду:

```
archive_cleanup_command = 'pg_archivecleanup -d
/mnt/standby/archive %r 2>>cleanup.log'
```

Предполагается, что каталог архива физически располагается на резервном сервере, так что команда `archive_command` обращается к нему по NFS, но для резервного сервера эти файлы локальные. Эта команда будет:

- выводить отладочную информацию в `cleanup.log`;
- удалять ставшие ненужными файлы из каталога архива.

3.3. Программа pg_controldata

Pg_controldata - вывести управляющую информацию кластера баз данных Синергия-БД.

3.3.1. Синтаксис

```
pg_controldata [ параметр ] [[-D] datadir]
```

3.3.2. Описание

`pg_controldata` показывает свойства, установленные командой `initdb`, например, версию каталога. Она также выводит сведения о работе журнала упреждающей записи и контрольных точках. Эта информация относится ко всему кластеру, а не к отдельной базе данных.

Утилита запускается от лица пользователя, создавшего кластер, так как требует права на чтение в каталоге хранения данных. Можно указать путь к каталогу из командной строки, либо использовать значение переменной окружения `PGDATA`. Также поддерживаются флаги `-V` и `--version`, которые выводят версию `pg_controldata` и прерывают выполнение. Флаги `-?` и `--help` отображают помощь по поддерживаемым командой аргументам.

3.3.3. Переменные окружения

`PGDATA`

Каталог размещения данных кластера по умолчанию

3.4. Программа pg_ctl

Pg_ctl - инициализировать, запустить, остановить или управлять сервером Синергия-БД.

3.4.1. Синтаксис

```
pg_ctl init[db] [ -s ] [-D datadir] [-o initdb-options]
pg_ctl start [ -w ] [-t секунды] [ -s ] [-D datadir] [-l
имя_файла] [-o параметры] [-p path] [ -c ]
pg_ctl stop [ -W ] [-t секунды] [ -s ] [-D datadir] [-m
s[mart] | f[ast] | i[mmediate] ]
pg_ctl restart [ -w ] [-t секунды] [ -s ] [-D datadir]
[ -c ] [-m s[mart] | f[ast] | i[mmediate] ] [-o
параметры]
pg_ctl reload [ -s ] [-D datadir]
pg_ctl status [-D datadir]
pg_ctl promote [ -s ] [-D datadir]
pg_ctl kill signal_name process_id
pg_ctl register [-N servicename] [-U имя_пользователя]
[-P пароль] [-D datadir] [-S a[uto] | d[emand] ] [ -w ]
[-t секунды] [ -s ] [-o параметры]
pg_ctl unregister [-N servicename]
```

3.4.2. Описание

pg_ctl - это утилита для начальной инициализации, запуска, остановки, повторного запуска и управления кластером баз данных Синергия-БД. Сервер можно стартовать в ручном режиме, но pg_ctl реализует задачи направления вывода в журнал и отсоединения от терминала и группы процессов, а также предоставляет удобный интерфейс остановки кластера.

Для инициализации нового кластера Синергия-БД используются режимы

init или initdb. Кластер - это коллекция баз данных под управлением единого сервера. По факту вызывается команда initdb. За подробностями обратитесь к п. 7.1.1.

Сервер запускается в режиме start. Процесс работает в фоновом режиме, а стандартный ввод связывается с /dev/null. По умолчанию в Unix-подобных системах вывод и ошибки сервера пишутся в устройство стандартного вывода (не ошибок) pg_ctl. Вывод pg_ctl следует перенаправить в файл или процесс, например, приложение ротации журналов rotatelogs; в ином случае, postgres будет писать вывод в управляющий терминал (в фоновом режиме) и останется в группе процессов оболочки.

В режиме stop сервер, работающий в указанном каталоге данных, останавливается. Параметр -m позволяет выбрать три различных режима остановки. Режим "Smart" ожидает отключения всех активных клиентов и завершения всех текущих процессов резервного копирования. Если сервер работает в режиме горячего резерва, восстановление и потоковая репликация будут прерваны, как только отключатся все клиенты. Режим "Fast" (выбираемый по умолчанию) не ожидает отключения клиентов и завершает все текущие процессы резервного копирования. Все активные транзакции откатываются, а клиенты принудительно отключаются, после чего сервер останавливается. Режим "Immediate" незамедлительно прерывает все серверные процессы, не выполняя процедуру штатной остановки. В результате при следующем запуске будет запущено восстановление после сбоя.

В режиме restart по сути выполняется остановка и последующий запуск сервера. Это позволяет изменить параметры командной строки postgres. Режим restart может не отработать, если при запуске сервера в командной строке задавались относительные пути.

Чтобы перечитать конфигурацию (postgresql.conf, pg_hba.conf и т. д.), используется reload, при этом процесс postgres получает системный сигнал SIGHUP. Это позволяет применить изменения без полного рестарта сервера.

Чтобы проверить статус кластера, используется `status`. Если кластер запущен, то будет выведен PID процесса, а также команда с использованными при запуске аргументами. Если кластер остановлен, то процесс вернёт статус завершения `3`. Если не указан каталог хранения данных, то процесс вернёт статус завершения `4`.

Чтобы перевести резервный сервер в режим основного, используется `promote`. При этом сервер прекращает работу в режиме восстановления и начинает работать в режиме чтения-записи.

Чтобы послать сигнал процессу, используется `kill`. Чтобы посмотреть список доступных сигналов, обратитесь к справке `--help`.

3.4.3. Параметры

`-c`

`--core-file`

Способствует сбросу дампа памяти процесса при крахе сервера на платформах, где это возможно, поднимая мягкие ограничения, задаваемые для файлов дампа. Это полезно при отладке и диагностике проблем, так как позволяет получить трассировку стека отказавшего процесса сервера.

`-D datadir`

`--pgdata datadir`

Указывает размещение конфигурационных файлов кластера. Если не указано, используется значение переменной окружения `PGDATA`.

`-l имя_файла`

`--log имя_файла`

Направляет вывод сообщений сервера в файл `имя_файла`. Файл создаётся, если он ещё не существует. При этом устанавливается `umask 077`, что предотвращает доступ других пользователей к этому файлу.

`-m режим`

`--mode режим`

Задаёт режим остановки кластера. Значением режим может быть `smart`, `fast`

или immediate, либо первая буква этих вариантов. По умолчанию выбирается режим fast.

-o параметры

Указывает флаги, которые будут переданы непосредственно программе postgres; несколько параметров складываются вместе.

Эти параметры обычно следует обрамлять одинарными или двойными кавычками, чтобы они передавались вместе как одна группа.

-o initdb-options

Указывает флаги, которые будут переданы в initdb.

Эти параметры обычно следует обрамлять одинарными или двойными кавычками, чтобы они передавались вместе как одна группа.

-p path

Указывает размещение исполняемого файла postgres. По умолчанию задействуется исполняемый файл postgres из того же каталога, из которого запускался pg_ctl, а если это невозможно, из жёстко заданного каталога инсталляции. Применять этот параметр может понадобиться, только если вы делаете что-то необычное или получаете сообщения, что найти исполняемый файл postgres не удаётся.

В режиме init этот параметр аналогичным образом задаёт размещение исполняемого файла initdb.

-s

--silent

Выводить лишь ошибки, без сообщений информационного характера.

-t

--timeout

Максимальное время (в секундах) ожидания запуска или остановки сервера. По умолчанию принимается значение переменной среды PGCTLTIMEOUT или, если оно не задано, 60 секунд.

-V

--version

Выводит версию pg_ctl и прерывает выполнение.

-w

Ждать завершения запуска или остановки. Это вариант по умолчанию при остановке, но не при запуске. Ожидая запуска, pg_ctl постоянно пытается подключиться к серверу. Ожидая остановки, pg_ctl ждёт, пока сервер не удалит свой файл PID. Этот параметр позволяет ввести парольную фразу SSL при запуске. pg_ctl возвращает код завершения, сообщающий об успехе запуска или остановки.

-W

Не ждать завершения запуска или остановки. Это вариант по умолчанию для режимов запуска и перезапуска.

-?

--help

Вывести справку по команде pg_ctl и прервать выполнение.

3.4.4. Переменные окружения

PGCTLTIMEOUT

Значение по умолчанию для максимального времени ожидания запуска или остановки сервера (в секундах). По умолчанию это время составляет 60 секунд.

PGDATA

Размещение каталога хранения данных по умолчанию.

pg_ctl, как и большинство других утилит Синергия-БД, также использует переменные окружения, поддерживаемые libpq.

3.4.5. Файлы

postmaster.pid

Наличие файла в каталоге хранения данных помогает pg_ctl определить, работает ли сервер в настоящий момент.

postmaster.opts

Если файл существует в каталоге хранения данных, то pg_ctl (при restart) передаст его содержимое в качестве аргументов postgres, если не

указаны иные значения в `-o`. Содержимое файла также отображается при вызове в режиме `status`.

3.4.6. Примеры

3.4.6.1. Запуск сервера

Для запуска сервера:

```
$ pg_ctl start
```

Для запуска сервера с ожиданием готовности к приему подключений:

```
$ pg_ctl -w start
```

Для старта сервера на `5433` порту без использования `fsync`:

```
$ pg_ctl -o "-F -p 5433" start
```

3.4.6.2. Остановка сервера

Для остановки сервера:

```
$ pg_ctl stop
```

Параметр `-m` указывает режим остановки:

```
$ pg_ctl stop -m fast
```

3.4.6.3. Повторный запуск сервера

Повторный запуск сервера производится аналогично остановке с дальнейшим его запуском, за исключением того, что `pg_ctl` использует аргументы, которые были переданы при предыдущем запуске кластера. В простейшем случае повторный запуск выглядит так:

```
$ pg_ctl restart
```

Для повторного запуска сервера с ожиданием полной остановки и последующего запуска:

```
$ pg_ctl -w restart
```

Для повторного запуска на `5433` порту с выключенным `fsync` после старта:

```
$ pg_ctl -o "-F -p 5433" restart
```

3.4.6.4. Вывод статуса работы сервера

Ниже представлен простейший вывод статуса pg_ctl:

```
$ pg_ctl status
pg_ctl: server is running (PID: 13718)
/usr/local/pgsql/bin/postgres "-D"
"/usr/local/pgsql/data" "-p" "5433" "-B" "128"
```

Эта командная строка будет использоваться в режиме повторного запуска.

3.5. Программа pg_resetxlog

Pg_resetwal - сбросить журнал упреждающей записи и другую управляющую информацию кластера Синергия-БД.

3.5.1. Синтаксис

```
pg_resetwal [-f] [-n] [параметр...] {[-D]
каталог_данных}
```

3.5.2. Описание

pg_resetwal очищает журнал предзаписи (WAL) и может сбросить некоторую другую управляющую информацию, хранящуюся в файле pg_control. Данная функция может быть востребована при повреждении этих файлов. Использовать её нужно только как крайнюю меру, когда запуск сервера оказывается невозможен из-за этого повреждения.

После выполнения этой команды запуск сервера, скорее всего, будет возможен, однако стоит учитывать, что база данных может содержать несогласованные данные из-за транзакций, зафиксированных частично. Вы должны немедленно выгрузить данные, выполнить initdb, а затем восстановить данные. После этого проверьте целостность базы и внесите необходимые коррективы.

Эту утилиту может запускать только пользователь, установивший сервер, так как ей нужны права записи/чтения в каталоге хранения данных кластера. В целях безопасности каталог необходимо указывать в командной строке. pg_resetwal не

поддерживает переменную окружения PGDATA.

Если pg_resetwal выводит сообщение о невозможности определить данные из pg_control, то команду можно запустить принудительно, указав -f. В этом случае будут использованы наиболее вероятные значения. Для большинства полей это нормально, но для некоторых может потребоваться явное указание: следующее значение OID, следующее значение ID транзакции и времени, ID мультитранзакции и смещение, начальный адрес WAL. Эти значения можно указать с помощью далее описанных параметров. Если их невозможно определить, то флаг f позволяет это обойти, однако, достоверность данных восстановленной базы останется под сомнением: незамедлительная выгрузка с последующим восстановлением данных крайне необходимы. Не выполняйте никаких операций модификации до создания дампа данных, так как это может привести к ещё более печальным последствиям.

3.5.3. Параметры

-f

Принудительно выполнять pg_resetwal, даже если не удаётся получить приемлемые данные из pg_control, как описано выше.

-n

С флагом -n (нет операции) команда pg_resetwal отображает извлечённые из pg_control данные, а также значения, которые можно изменить. Режим полезен для отладки и тестирования предстоящей операции без реального применения изменений.

-V

--version

Показать версию, а затем завершиться.

-?

--help

Показать справку, а затем завершиться.

Следующие параметры необходимы, только когда pg_resetwal не может определить подходящие значения, прочитав pg_control. Безопасные значения

можно определить, как описано ниже. Для значений, принимающих числовые аргументы, можно задать шестнадцатеричные значения, добавив префикс `0x`.

`-c xid,xid`

Вручную задать идентификаторы старейшей и новейшей транзакций, для которых можно получить время фиксации. Безопасное значение идентификатора старейшей транзакции, для которой можно получить время фиксации (первый компонент), можно определить, найдя наименьшее в числовом виде имя файла в каталоге `pg_commit_ts` внутри каталога данных. Безопасное значение идентификатора новейшей транзакции, для которой можно получить время фиксации (второй компонент), можно определить, найдя, напротив, наибольшее в числовом виде имя файла в том же каталоге. Числа в именах этих файлов представлены в шестнадцатеричном формате.

`-e эпоха_xid`

Вручную задать эпоху в ID следующей транзакции. Эпоха идентификаторов транзакции не хранится в базе данных нигде, кроме поля, устанавливаемого командой `pg_resetwal`, поэтому если рассматривать собственно базу, допустимым будет любое значение. Это значение, возможно, понадобится скорректировать для обеспечения правильной работы системы репликации, например, `Slony-I` и `Skytools`. В этом случае подходящее значение следует получить из состояния нижележащей реплицированной базы данных.

`-l walfile`

Вручную задать начальный адрес WAL. Начальный адрес журнала WAL должен превышать наибольшее значение сегмента в именифайла, расположенного в каталоге `pg_wal`. Эти имена тоже представлены в шестнадцатеричном формате и состоят из трёх частей. Первая из них - «ID линии времени» и её обычно не следует менять. Например, если `000000010000003200000004A` - наибольшее значение в `pg_wal`, нужно указать `-l 000000010000003200000004B` или большее число.

`-m mxid,mxid`

Вручную задать ID следующей и старейшей мультитранзакции. Безопасное значение следующего идентификатора мультитранзакции (первый компонент)

можно вычислить, найдя наибольшее числовое значение среди имён файлов, расположенных в каталоге pg_multixact/offsets. К найденному значению необходимо прибавить один, затем умножить на 65536 (0x10000). Для вычисления безопасного значения ID старейшей мультитранзакции (второй компонент -m) необходимо найти наименьшее числовое значение среди тех же файлов, и умножить его на 65536. Имена файлов представляются в шестнадцатеричном формате, так что значения проще указывать в нём же, добавив в конце четыре нуля.

-o oid

Вручную задать следующий OID. Не существует относительно простого способа вычисления, следующего за наибольшим из существующих значением OID, однако это некритично.

-O mxoff

Вручную задать смещение следующей мультитранзакции. Безопасное значение можно определить, найдя наибольшее числовое значение в имени файла в каталоге pg_multixact/members, прибавив один и умножив результат на 52352 (0xCC80). Имена файлов представлены в шестнадцатеричном формате. Простого рецепта с прибавлением нулей в конце, как для других параметров, в данном случае нет.

-x xid

Вручную задать ID следующей транзакции. Безопасное значение можно определить, найдя наибольшее числовое значение в имени файла в подкаталоге pg_xact каталога данных, прибавив один и умножив результат на 1048576 (0x100000). Заметьте, что имена файлов представлены в шестнадцатеричном формате. Значение этого параметра обычно также проще задавать в шестнадцатеричном виде. Например, если 0011 - наибольшее значение в pg_xact, подходящим значением будет -x 0x1200000 (нужный множитель дают пять последних нулей).

pg_resetwal работает только с серверами той же основной версии.

3.6. Программа pg_rewind

Pg_rewind - синхронизировать каталог данных Синергия-БД с другим каталогом, ответвлённым от первого.

3.6.1. Синтаксис

```
pg_rewind [ параметр ...] { -D | --target-
pgdata }каталог { --source-pgdata=каталог | --source-
server=строка подключения }
```

3.6.2. Описание

Утилита pg_rewind представляет собой средство синхронизации кластера Синергия-БД с другой копией того же кластера после расхождения линий времени этих кластеров. Обычный сценарий её использования - вернуть в работу старый главный сервер после переключения на резервный, в качестве резервного для сервера, ставшего главным.

Её результат равнозначен замене целевого каталога данных исходным. Она копирует все файлы, включая файлы конфигурации. Преимущество pg_rewind по сравнению с созданием новой базовой копии или такими средствами, как rsync, состоит в том, что pg_rewind не требует читать все неизменённые файлы в кластере. Благодаря этому, она действует гораздо быстрее, когда база данных большая, а объём различий между кластерами невелик.

Утилита pg_rewind изучает историю линий времени исходного и целевого кластеров с целью найти точку, в которой они разошлись, и ожидает найти журналы WAL в каталоге pg_wal целевого кластера вплоть до точки расхождения. В типичном сценарии отработки отказа, когда целевой кластер отключается вскоре после расхождения, это не проблема, но, если целевой кластер проработал долгое время после расхождения, старые файлы WAL могут быть уже удалены. В этом случае их можно вручную скопировать из архива WAL в каталог pg_wal. Автоматическая загрузка недостающих файлов из архива WAL в настоящее время не поддерживается.

Когда целевой сервер запускается в первый раз после выполнения pg_rewind, он переходит в режим восстановления и воспроизводит все изменения из WAL с исходного сервера после точки расхождения. Если какие-то сегменты WAL оказались недоступны на исходном сервере, когда выполнялась pg_rewind, и поэтому не могли быть скопированы в ходе работы pg_rewind, их необходимо предоставить, когда сервер будет запускаться. Это можно сделать, создав в целевом каталоге данных файл recovery.conf с подходящей командой restore_command.

Утилита pg_rewind требует, чтобы на целевом сервере был либо включён режим wal_log_hints в postgresql.conf, либо включены контрольные суммы, когда кластер был инициализирован командой initdb. Оба эти режима по умолчанию отключены. Также должен быть включён режим full_page_writes. По умолчанию он включён.

3.6.3. Параметры

Аргументы командной строки для pg_rewind.

-D каталог

--target-pgdata=каталог

Этот параметр задаёт целевой каталог данных, который будет синхронизирован с источником. Целевой сервер должен быть отключён штатным образом до запуска pg_rewind.

--source-pgdata=каталог

Задаёт путь к каталогу данных исходного сервера, с которым будет синхронизироваться целевой. Когда используется ключ --source-pgdata, исходный сервер должен быть остановлен штатным образом.

--source-server=строка подключения

Задаёт строку подключения libpq для подключения к исходному серверу Синергия-БД, с которым будет синхронизирован целевой. Подключение должно устанавливаться как обычное (не реплицирующее) с правами суперпользователя. Сервер должен быть запущен и работать в обычном режиме, не в режиме

восстановления.

`-n`

`--dry-run`

Делать всё, кроме внесения изменений в целевой каталог.

`-P`

`--progress`

Включает вывод сообщений о прогрессе. При этом в процессе копирования данных из исходного кластера будет выдаваться приблизительный процент выполнения.

`--debug`

Выводить подробные отладочные сообщения, полезные в основном для разработчиков, отлаживающих `pg_rewind`.

`-V`

`--version`

Показать версию, а затем завершиться.

`-?`

`--help`

Показать справку, а затем завершиться.

3.6.4. Переменные окружения

Когда используется `--source-server`, `pg_rewind` также использует переменные среды, поддерживаемые `libpq`.

3.6.5. Замечания

Когда исходным кластером для `pg_rewind` является работающий сервер, вместо суперпользователя может применяться роль, имеющая в этом кластере достаточные права для выполнения функций, которые использует `pg_rewind`. Такую роль (`rewind_user`) можно создать так:

```
CREATE USER rewind_user LOGIN;
GRANT EXECUTE ON function pg_catalog.pg_ls_dir(text,
```

```
boolean, boolean) TO rewind_user;
GRANT EXECUTE ON function pg_catalog.pg_stat_file(text,
boolean) TO rewind_user;
GRANT EXECUTE ON function
pg_catalog.pg_read_binary_file(text) TO rewind_user;
GRANT EXECUTE ON function
pg_catalog.pg_read_binary_file(text, bigint, bigint,
boolean) TO rewind_user;
```

Когда исходным кластером для `pg_rewind` является работающий сервер сразу после повышения, в нём необходимо выполнить команду `CHECKPOINT`, чтобы его управляющий файл содержал актуальную информацию о линии времени. Эта информацию нужна `pg_rewind` для проверки, можно ли синхронизировать целевой кластер с выбранным исходным.

Основная идея заключается в том, чтобы скопировать из нового кластера в старый всё, за исключением блоков, которые, насколько нам известно, одинаковые:

1. Просканировать журнал WAL в старом кластере, начиная с последней контрольной точки, и до точки, в которой история линии времени нового кластера разошлась со старым кластером. Для каждой записи WAL отметить, какие блоки данных были затронуты. В результате будет получен список всех блоков данных, которые были изменены в старом кластере после отделения нового кластера.
2. Скопировать все эти изменённые блоки из нового кластера в старый.
3. Скопировать все остальные файлы, в частности `clog` и файлы конфигурации из нового кластера в старый, пропуская файлы отношений.
4. Применить WAL из нового кластера, начиная с контрольной точки, созданной при отработке отказа. Строго говоря, утилита `pg_rewind` не применяет WAL, она просто создаёт файл метки резервной копии, найдя который при следующем запуске, Синергия-БД начнёт воспроизведение с этой контрольной точки и применит все требуемые записи WAL.

3.7. Программа pg-setup

Pg-setup - создать кластер баз данных СУБД «Синергия-БД» и управлять соответствующей службой.

3.7.1. Синтаксис

```
pg-setupinitdb [параметры_initdb]
pg-setup find-free-port
pg-setup set-server-port порт
pg-setup set параметр значение
pg-setup service параметр_service
```

3.7.2. Описание

Программа `pg-setup` представляет собой скрипт оболочки, включённый в дистрибутив СУБД «Синергия-БД» и предназначенный для автоматизации установки кластера баз данных в системах Linux. Этот скрипт входит в состав пакета `postgresql-sdb-11-server`. После установки СУБД «Синергия-БД» вы можете найти `pg-setup` в каталоге `каталог-инсталляции/bin`, где каталог-инсталляции обозначает путь `/opt/sdb/sdb-11`.

Программу `pg-setup` нужно запускать от имени `root`, но она выполняет административные операции с базой данных под именем пользователя `postgres`. Запуская её с разными параметрами, вы можете:

- инициализировать кластер баз данных;
- проверить доступные порты и изменить порт, который будет использоваться сервером СУБД «Синергия-БД»;
- изменить параметры конфигурации сервера;
- включить/отключить автоматический запуск службы СУБД «Синергия-БД»;
- запустить, остановить или перезапустить службу СУБД «Синергия-БД».

3.7.3. Параметры

Аргументы командной строки для `pg-setup`.

`initdb [параметры_initdb]`

Инициализировать кластер баз данных от имени пользователя postgres.

По умолчанию кластер баз данных инициализируется в каталоге /var/lib/sdb/sdb-11/data; в этом кластере будут включены контрольные суммы, параметр auth-local будет иметь значение peer, а auth-host - значение md5. Параметры локализации наследуются из переменной окружения LANG, определённой в текущем сеансе. Все переменные окружения LC_* при этом игнорируются. Вы можете также передать параметры initdb для дополнительной настройки инсталляции.

Если база данных по умолчанию создаётся с помощью pg-setup, путь к каталогу данных сохраняется в файле /etc/default/postgrespro-SDB-11, чтобы все последующие команды pg-setup, а также все команды, управляющие службой СУБД «Синергия-БД», воздействовали именно на эту базу данных. Управлять несколькими базами данных с помощью pg-setup нельзя.

find-free-port

Найти свободный порт в системе. Это может быть полезно, если вы планируете устанавливать несколько экземпляров серверов, либо стандартный порт 5432 уже используется другой программой.

set-server-port порт

Задать номер порта, на котором сервер будет принимать соединения. Воспользуйтесь этой командой, чтобы избежать конфликтов, если вы устанавливаете в одной системе более одного сервера.

По умолчанию: 5432

set имя значение

Установить для заданного параметра конфигурации указанное значение в файле postgresql.conf. Если этот параметр ранее был определён командой ALTER SYSTEM, его предыдущее значение удаляется из файла postgresql.auto.conf.

service параметр_service

Управление службой СУБД «Синергия-БД» с помощью следующих команд:

- enable - включить автоматический запуск службы при перезагрузке системы;

- disable - отключить автоматический запуск службы при перезагрузке системы;

- start - запустить службу;

- stop - остановить службу;

- condrestart - перезапустить службу, если она работает в момент вызова pg-setup;

- status - возвращает статус службы СУБД «Синергия-БД».

3.7.4. Замечания

Если вы устанавливаете СУБД «Синергия-БД» из пакета postgresql-sdb-11, pg-setup вызывается автоматически с параметрами по умолчанию. В результате создаётся кластер баз данных и база по умолчанию в каталоге /var/lib/sdb/sdb-11/data, настраивается автозапуск службы СУБД «Синергия-БД» и служба запускается.

Если вы устанавливаете сервер СУБД «Синергия-БД» непосредственно из пакета postgresql-sdb-11-server, вы должны запустить этот скрипт вручную для инициализации кластера баз данных или управления службой СУБД «Синергия-БД».

3.8. Программа pg_test_fsync

Pg_test_fsync - подобрать наилучший вариант wal_sync_method для Синергия-БД.

3.8.1. Синтаксис

```
pg_test_fsync [параметр...]
```

3.8.2. Описание

Программа pg_test_fsync предназначена для того, чтобы дать вам

представление о том, какой из вариантов wal_sync_method оптимален для вашей конкретной системы, а также выдать вспомогательные диагностические сведения в случае проблем со вводом/выводом. Однако отличия, показанные программой pg_test_fsync, могут не оказывать большого влияния на реальную производительность баз данных, в частности потому, что для многих серверов баз данных производительность упирается не в запись журналов предзаписи. pg_test_fsync выводит среднее время операции синхронизации с ФС для каждого метода wal_sync_method, что также может быть полезно при поиске оптимального значения commit_delay.

3.8.3. Параметры

Параметры командной строки для pg_test_fsync.

-f

--filename

Задаёт имя файла для записи данных тестов. Этот файл должен находиться в той же файловой системе, где размещается или будет размещаться каталог pg_wal. В каталоге pg_wal содержатся файлы WAL. По умолчанию выбирается файл pg_test_fsync.out в текущем каталоге.

-s

--secs-per-test

Задаёт продолжительность каждого теста (в секундах). Чем больше длится тест, тем точнее результат, но тем дольше работает программа. Со значением по умолчанию (5 секунд) программа должна завершиться примерно за 2 минуты.

-V

--version

Вывести версию pg_test_fsync и завершиться.

-?

--help

Вывести справку об аргументах командной строки pg_test_fsync и завершиться.

3.9. Программа pg_test_timing

Pg_test_timing - определить издержки замера времени.

3.9.1. Синтаксис

```
pg_test_timing [параметр...]
```

3.9.2. Описание

Программа pg_test_timing позволяет оценить издержки замера времени в вашей системе и убедиться в том, что системное время никогда не идёт назад. Системы, в которых замер времени является длительной операцией, дают менее точные результаты EXPLAIN ANALYZE.

3.9.3. Параметры

Аргументы командной строки для pg_test_timing.

-d длительность

--duration=длительность

Задаёт продолжительность теста (в секундах). Чем больше эта продолжительность, тем выше точность и больше вероятность обнаружить аномалию с обратным ходом системных часов. По умолчанию время тестирования - 3 секунды.

-V

--version

Вывести версию pg_test_timing и завершиться.

-?

--help

Вывести справку об аргументах командной строки pg_test_timing и завершиться.

3.9.4. Использование

3.9.4.1. Интерпретация результатов

В благоприятном случае практически все (>90%) отдельные вызовы замеров времени должны выполняться быстрее одной микросекунды. Средние издержки замера на цикл должны быть ещё меньше, в пределах 100 наносекунд. Эта проба, взятая в системе Intel i7-860 через источник времени TSC, показывает отличную производительность:

```
Testing timing overhead for 3 seconds.
Per loop time including overhead: 35.96 ns
Histogram of timing durations:
  < us   % of total      count
     1     96.40465   80435604
     2      3.59518    2999652
     4      0.00015        126
     8      0.00002         13
    16      0.00000          2
```

Заметьте, что время вызова в цикле и время в гистограмме выражается в разных единицах. Время в цикле может определяться с точностью до наносекунд (`ns`), а длительность отдельного вызова замера времени - только с точностью до микросекунд (`us`).

3.9.4.2. Измерение издержек исполнителя на замер времени

Когда исполнитель запроса выполняет запрос под контролем `EXPLAIN ANALYZE`, замеряется не только общее время, но и время отдельных операций. Каковы издержки этих операций в вашей системе, можно узнать, подсчитав строки тестовой таблицы в программе `psql`:

```
CREATE TABLE t AS SELECT * FROM
generate_series(1,100000);
\timing
```

```
SELECT COUNT(*) FROM t;
EXPLAIN ANALYZE SELECT COUNT(*) FROM t;
```

В системе i7-860 этот запрос выполняется 9.8 мс, а версия с EXPLAIN ANALYZE - 16.6 мс, при этом обрабатывается около 100 000 строк. Это различие в 6.8 мс означает, что издержки замера времени для одной строки составляют около 68 нс, примерно вдвое больше, чем предсказала pg_test_timing. Даже с такими относительно небольшими издержками операция COUNT с полным подсчётом времени выполняется почти на 70% дольше. На более сложных запросах издержки замера времени могут быть не так важны.

3.9.4.3. Смена источника времени

В некоторых современных системах Linux можно в любой момент сменить источник времени, который используется для замера времени. Второй пример иллюстрирует возможное замедление от переключения на более медленный источник времени acpi_pm time в той же системе, в которой были получены показанные выше хорошие результаты:

```
# cat
/sys/devices/system/clocksource/clocksource0/available_c
locksource
tsc hpet acpi_pm
# echo acpi_pm >
/sys/devices/system/clocksource/clocksource0/current_clo
cksource
# pg_test_timing
Per loop time including overhead: 722.92 ns
Histogram of timing durations:
  < us   % of total      count
     1     27.84870    1155682
     2     72.05956    2990371
     4      0.07810       3241
```

```
     8      0.01357         563
    16      0.00007           3
```

В этой конфигурации тот же `EXPLAIN ANALYZE` выполняется 115.9 мс. Таким образом издержки составили 1061 нс, что соответствует непосредственному результату этой утилиты с небольшим коэффициентом. Такие большие издержки означают, что сам запрос выполняется лишь небольшой процент всего времени, а основное время уходит на замеры времени. В такой конфигурации временные показатели `EXPLAIN ANALYZE` для запросов со множеством замеряемых операций значительно увеличатся за счёт издержек замера времени.

Другие системы могут допускать смену источника времени только при загрузке. В старых системах Linux это можно было сделать только с помощью параметра ядра «clock». И даже в некоторых самых последних системах можно увидеть только один источник времени - `jiffies`. Это старая программная реализация часов в Linux, которая может давать хорошее разрешение, когда поддерживается достаточно хорошим оборудованием, как в этом примере:

```
$ cat
/sys/devices/system/clocksource/clocksource0/available_c
locksource
jiffies
$ dmesg | grep time.c
time.c: Using 3.579545 MHz WALL PM GTOD PIT/TSC timer.
time.c: Detected 2400.153 MHz processor.
$ pg_test_timing
Testing timing overhead for 3 seconds.
Per timing duration including loop overhead: 97.75 ns
Histogram of timing durations:
  < us   % of total      count
     1     90.23734   27694571
     2      9.75277    2993204
     4      0.00981       3010
```

```
 8       0.00007          22
16       0.00000           1
32       0.00000           1
```

3.9.4.4. Аппаратные часы и точность замера времени

Измерение времени обычно осуществляется на компьютерах по аппаратным часам, точность которых может быть разного уровня. С некоторым оборудованием операционные системы могут передавать время системных часов непосредственно программам. Также системное время может поступать с чипа, который просто генерирует прерывания по времени, с заведомо известным периодом. В любом случае ядра операционных систем предоставляют источник времени, который скрывает эти детали. Но точность этого источника и возможная скорость получения результатов от него зависит от нижележащего оборудования.

Неточность в замерах времени может приводить к нестабильности системы. Поэтому стоит очень тщательно протестировать выбранный источник времени. Иногда по умолчанию в ОС выбирается источник не более точный, а более надёжный. И если вы используете виртуальную машину, поинтересуйтесь, какие источники времени рекомендуется использовать с ней. Имитация таймеров на виртуальном оборудовании связана с дополнительными сложностями, и производители средств виртуализации часто рекомендуют определённые параметры для операционных систем.

Источник времени TSC (Time Stamp Counter, Счётчик отметки времени) наиболее точный из всех для процессоров текущего поколения. Его рекомендуется использовать для получения системного времени, когда он поддерживается операционной системой и показания TSC надёжны. Возможны ситуации, когда TSC не является точным источником времени, и таким образом, оказывается ненадёжным. Например, в старых системах показания TSC могут зависеть от температуры процессора, что не годится для точного замера времени. При попытке использовать TSC в некоторых старых многоядерных процессорах можно получить разное время на различных ядрах. В результате может оказаться, что время идёт

назад (эту аномалию выявляет данная программа). И даже на самых современных системах не всегда можно получить точное время через TSC в режимах очень агрессивного энергосбережения.

Новые операционные системы могут проверять наличие известных проблем TSC и переключаться на более медленный, но более стабильный источник времени, если они проявляются. Если ваша система поддерживает источник TSC, но не выбирает его по умолчанию, возможно, он отключён обоснованно. С другой стороны, некоторые операционные системы могут не выявлять все возможные проблемы, либо разрешают использовать TSC даже в ситуациях, когда он определённо неточен.

HPET (High Precision Event Timer, Таймер событий высокой точности) рекомендуется использовать в системах, где он имеется, а TSC неточен. Сам этот чип можно запрограммировать для получения точности до 100 наносекунд, но системное время с такой точностью вы не получите.

ACPI (Advanced Configuration and Power Interface, Расширенный интерфейс конфигурации и питания) обеспечивает таймер PM (Управления питанием), который в Linux называется `acpi_pm`. Время, поступающее из `acpi_pm`, в лучшем случае будет иметь разрешение 300 наносекунд.

На старом оборудовании PC применялись таймеры 8254 PIT (Programmable Interval Timer, Программируемый интервальный таймер), RTC (Real-Time Clock, Часы реального времени), таймер APIC (Advanced Programmable Interrupt Controller, Расширенный программируемый контроллер прерываний) и Cyclone. Все эти таймеры обеспечивали точность до миллисекунд.

3.10. Программа pg_upgrade

Pg_upgrade - обновить экземпляр сервера Синергия-БД.

3.10.1. Синтаксис

```
pg_upgrade-bстарый_каталог_bin-Вновый_каталог_bin-
dстарый_каталог_конфигурации-Dновый_каталог_конфигурации
```

[параметр...]

3.10.2. Описание

Программа pg_upgrade (ранее называвшаяся pg_migrator) позволяет обновить данные в каталоге базы данных Синергия-БД до последней основной версии Синергия-БД без операции выгрузки/перезагрузки данных, обычно необходимой при обновлениях основной версии. Эти действия не требуются при установке корректирующей версии.

С выходом новых основных версий в Синергия-БД регулярно добавляются новые возможности, которые часто меняют структуру системных таблицы, но внутренний формат хранения меняется редко. Учитывая этот факт, pg_upgrade позволяет выполнить быстрое обновление, создавая системные таблицы заново, но сохраняя старые файлы данных. Если при обновлении основной версии формат хранения данных изменится так, что данные в старом формате окажутся нечитаемыми, pg_upgrade не сможет произвести такое обновление.

Программа pg_upgrade делает всё возможное, чтобы убедиться в том, что старый и новый кластеры двоично-совместимы, в частности проверяя параметры времени компиляции и разрядность (32/64 бита) исполняемых файлов. Важно, чтобы и все внешние модули тоже были двоично-совместимыми, хотя это pg_upgrade проверить не может.

3.10.3. Параметры

Аргументы командной строки для pg_upgrade.

-b каталог_bin

--old-bindir= каталог_bin

Каталог с исполняемыми файлами старой версии Синергия-БД; переменная окружения PGBINOLD.

-B каталог_bin

--new-bindir= каталог_bin

Каталог с исполняемыми файлами новой версии Синергия-БД; переменная

окружения PGBINNEW.

-c

--check

Только проверить кластеры, не изменять никакие данные.

-d каталог_конфигурации

--old-datadir= каталог_конфигурации

Каталог конфигурации старого кластера; переменная окружения PGDATAOLD.

-D каталог_конфигурации

--new-datadir= каталог_конфигурации

Каталог конфигурации нового кластера; переменная окружения PGDATANEW.

-j

--jobs

Число одновременно задействуемых процессов или потоков.

-k

--link

Использовать жёсткие ссылки вместо копирования файлов в новый кластер.

-o параметры

--old-options параметры

Параметры, передаваемые непосредственно старой программе postgres; несколько параметров складываются вместе.

-O параметры

--new-options параметры

Параметры, передаваемые непосредственно новой программе postgres; несколько параметров складываются вместе.

-p порт

--old-port= порт

Номер порта старого кластера; переменная окружения PGPORTOLD.

-P порт

--new-port= порт

Номер порта нового кластера; переменная окружения PGPORTNEW.

-r

--retain

Сохранить SQL и журналы сообщений даже при успешном завершении.

-U имя_пользователя

--username= имя_пользователя

Имя пользователя, установившего кластер; переменная окружения PGUSER.

-v

--verbose

Включить подробные внутренние сообщения.

-V

--version

Показать версию, а затем завершиться.

-?

--help

Показать справку, а затем завершиться.

3.10.4. Использование

Далее описан план обновления с использованием pg_upgrade:

1) переместить старый кластер (необязательно)

Если ваш каталог инсталляции привязан к версии, например, /opt/sdb/11, перемещать его не требуется. Все графические инсталляторы выбирают при установке каталоги, привязанные к версии.

Если ваш каталог инсталляции не привязан к версии, например, /usr/local/pgsql, необходимо переместить каталог текущей инсталляции Синергия-БД, чтобы он не конфликтовал с новой инсталляцией Синергия-БД. Когда текущий сервер Синергия-БД отключён, каталог этой инсталляции Синергия-БД можно безопасно переместить; если старый каталог /usr/local/pgsql, его можно переименовать, выполнив:

mv /usr/local/pgsql /usr/local/pgsql.old

2) собрать новую версию при установке из исходного кода

Соберите из исходного кода новую версию Синергия-БД с флагами configure, совместимыми с флагами старого кластера. Программа pg_upgrade проверит результаты pg_controldata, чтобы убедиться, что все параметры совместимы, прежде чем начинать обновление;

3) установить новые исполняемые файлы Синергия-БД

Установите новые исполняемые файлы сервера и вспомогательные файлы. Программа pg_upgrade включена в инсталляцию по умолчанию.

При установке из исходного кода, если вы хотите разместить сервер в нестандартном каталоге, воспользуйтесь переменной prefix:

```
make prefix=/usr/local/pgsql.new install
```

4) инициализировать новый кластер Синергия-БД

Инициализируйте новый кластер, используя initdb. При этом так же необходимо указать флаги initdb, совместимые с флагами в старом кластере. Многие готовые инсталляторы выполняют это действие автоматически. Запускать новый кластер не требуется;

5) установить дополнительные разделяемые объектные файлы

Установите в новый кластер все нестандартные разделяемые объектные файлы (или DLL), которые использовались в старом кластере, например, pgcrypto.so, где бы они ни находились - в contrib или в другом месте. Устанавливать определения схемы (например, CREATE EXTENSION pgcrypto) не требуется, так как они будут перенесены из старого кластера. Кроме того, в новый кластер нужно скопировать и все нестандартные файлы поддержки полнотекстового поиска (словари, тезаурусы, списки синонимов и стоп-слов);

6) настроить аутентификацию

Программа pg_upgrade будет подключаться к новому и старому серверу несколько раз, так что имеет смысл установить режим аутентификации peer в pg_hba.conf или использовать файл ~/.pgpass;

7) остановить оба сервера

Убедитесь в том, что оба сервера баз данных остановлены. Для этого в Unix можно выполнить:

```
pg_ctl -D /opt/PostgreSQL/9.6 stop
pg_ctl -D /opt/PostgreSQL/11 stop
```

Ведомые серверы с потоковой репликацией и трансляцией журнала могут продолжать работать до последнего шага;

8) подготовиться к обновлению ведомых серверов

Если вы производите обновление ведомых серверов (как описано в шаге 10), удостоверьтесь, что эти серверы находятся в актуальном состоянии, запустив `pg_controldata` в старых ведущем и ведомых кластерах. Убедитесь в том, что «Положение последней контрольной точки» во всех кластерах одинаковое. Несовпадение будет иметь место, если старые ведомые серверы будут отключены раньше, чем старый ведущий, или если старые сервера продолжают работать. Также смените `wal_level` на `replica` в файле `postgresql.conf` нового ведущего кластера;

9) запустить `pg_upgrade`

Всегда запускайте программу `pg_upgrade` от нового сервера, а не от старого. `pg_upgrade` требует указания каталогов данных старого и нового кластера, а также каталогов исполняемых файлов (`bin`). Вы можете также определить имя пользователя и номера портов, и нужно ли копировать файлы данных (по умолчанию) или создавать ссылки на них.

Если выбрать вариант со ссылкой на данные, обновление выполнится гораздо быстрее (так как файлы не копируются) и потребует меньше места на диске, но вы лишитесь возможности обращаться к вашему старому кластеру, запустив новый после обновления. Этот вариант также требует, чтобы каталоги данных старого и нового кластера располагались в одной файловой системе. Табличные пространства и `pg_wal` могут находиться в других файловых системах. Полный список параметров вы можете получить, выполнив `pg_upgrade --help`.

Параметр `--jobs` позволяет задействовать для копирования/связывания

файлов и для выгрузки/перезагрузки схем баз данных несколько процессорных ядер. В качестве начального значения параметра стоит выбрать максимум из числа процессорных ядер и числа табличных пространств. Этот параметр может радикально сократить время обновления сервера со множеством баз данных, работающего в многопроцессорной системе.

При запуске `pg_upgrade` проверит два кластера на совместимость и, если всё в порядке, выполнит обновление. Также возможно запустить `pg_upgrade --check`, чтобы ограничиться только проверками (при этом старый сервер может продолжать работать). Команда `pg_upgrade --check` также сообщит, какие коррективы вам нужно будет внести вручную после обновления. Если вы планируете использовать режим ссылок на данные, укажите вместе с `--check` параметр `--link`, чтобы были проведены специальные проверки для этого режима. Программе `pg_upgrade` требуются права на запись в текущий каталог.

Очевидно, никто не должен обращаться к кластерам в процессе обновления. Программа `pg_upgrade` по умолчанию запускает серверы с портом `50432`, чтобы не допустить нежелательных клиентских подключений. В процессе обновления оба кластера могут использовать один номер порта, так как они не будут работать одновременно. Однако для проверки старого работающего сервера новый порт должен отличаться от старого.

Если при восстановлении схемы базы данных происходит ошибка, `pg_upgrade` завершает свою работу и вы должны вернуться к старому кластеру, как описывается ниже в Шаг 16. Чтобы попробовать `pg_upgrade` ещё раз, вы должны внести коррективы в старом кластере, чтобы `pg_upgrade` могла успешно восстановить схему. Если проблема возникла в модуле `contrib`, может потребоваться удалить этот модуль `contrib` в старом кластере, а затем установить его в новом после обновления (предполагается, что этот модуль не хранит пользовательские данные);

10) обновить ведомые серверы с потоковой репликацией и трансляцией журнала

Если вы используете режим ссылок и у вас реализована потоковая репликация или трансляция журнала для ведомых серверов, вы можете быстро обновить эти серверы следующим образом. Вам не нужно будет запускать на них `pg_upgrade`, вместо этого вы выполните rsync на ведущем. Не запускайте никакие серверы на этом этапе.

Если вы *не* используете режим ссылок, либо у вас нет `rsync` или вы не хотите его использовать, либо если вам нужен более простой вариант, пропустите инструкции в этом разделе и просто пересоздайте ведомые серверы сразу после завершения `pg_upgrade` и запуска нового ведущего сервера:

- установите новые исполняемые файлы Синергия-БД на ведомых серверах.

Убедитесь в том, что на всех ведомых серверах установлены новые исполняемые и вспомогательные файлы;

- убедитесь в том, что новые каталоги данных на ведомых серверах не существуют.

Новые каталоги данных ведомых серверов должны *отсутствовать* либо быть пустыми. Если запускалась программа `initdb`, удалите новые каталоги данных на ведомых;

- установить дополнительные разделяемые объектные файлы.

Установите на новых ведомых серверах те же дополнительные разделяемые объектные файлы, что вы установили в новом ведущем кластере;

- остановите ведомые серверы.

Если ведомые серверы продолжают работу, остановите их, следуя приведённым выше инструкциям;

- сохраните файлы конфигурации.

Сохраните все нужные вам файлы конфигурации из старых каталогов конфигурации ведомых серверов, в частности `postgresql.conf` и `recovery.conf`, так как они будут перезаписаны или удалены на следующем этапе;

- Запустите rsync.

Когда используется режим ссылок, ведомые серверы можно быстро обновить, применив rsync. Для этого в каталоге, внутри которого находятся каталоги старого и нового кластера, для каждого ведомого сервера выполните на *ведущем*:

```
rsync --archive --delete --hard-links --size-only --
no-inc-recursive old_cluster new_cluster remote_dir
```

Здесь каталоги old_cluster и new_cluster задаются относительно текущего каталога на ведущем, а remote_dir находится *над* каталогами старого и нового кластера на ведомом. Структура подкаталогов в заданных каталогах на ведущем и ведомых серверах должна быть одинаковой. Обратитесь к странице руководства rsync, где подробно описано, как указать удалённый каталог, например, так:

```
rsync --archive --delete --hard-links --size-only --
no-inc-recursive /opt/PostgreSQL/9.5 \
      /opt/PostgreSQL/9.6
standby.example.com:/opt/PostgreSQL
```

Проверить, что будет делать команда, можно, воспользовавшись параметром rsync --dry-run. Выполнить rsync на ведущем необходимо как минимум с одним ведомым, но затем, пока обновлённый ведомый остаётся остановленным, можно запускать rsync на нём для обновления других ведомых.

В ходе этой операции записываются ссылки, созданные режимом ссылок pg_upgrade, связывающие файлы нового и старого кластера на ведущем сервере. Затем в старом кластере ведомого находятся соответствующие файлы и в новом кластере ведомого создаются ссылки на них. Файлы, не связанные ссылками на ведущем, копируются с него на ведомый. Обычно их объём невелик. Это позволяет произвести обновление ведомого быстро. К сожалению, при этом rsync будет напрасно копировать файлы, связанные с временными и нежурналируемыми таблицами, так как они обычно не будут

существовать на ведомых серверах.

Если у вас есть табличные пространства, вам потребуется выполнить подобную команду rsync для каталогов всех табличных пространств, например:

```
rsync --archive --delete --hard-links --size-only --
no-inc-recursive /vol1/pg_tblsp/PG_9.5_201510051 \
      /vol1/pg_tblsp/PG_9.6_201608131
standby.example.com:/vol1/pg_tblsp
```

Если вы вынесли pg_wal за пределы каталогов данных, нужно будет запустить rsync и для этих каталогов;

- настройте ведомые серверы с потоковой репликацией и трансляцией журнала.

Настройте серверы для трансляции журнала. Запускать pg_start_backup() и pg_stop_backup() или делать копию файловой системы не нужно, так как ведомые серверы остаются синхронизированными с ведущим;

11) восстановить pg_hba.conf

Если вы изменяли pg_hba.conf, восстановите его исходное состояние. Также может потребоваться скорректировать другие файлы конфигурации в новом кластере, чтобы они соответствовали старому, например, postgresql.conf;

12) запустить новый сервер

Теперь можно безопасно запустить новый сервер, а затем ведомые серверы, синхронизированные с ним с помощью rsync;

13) действия после обновления

Если после обновления требуются какие-то дополнительные действия, программа pg_upgrade выдаст предупреждения об этом по завершении работы. Она также сгенерирует файлы скриптов, которые должны запускаться администратором. Эти скрипты будут подключаться к каждой базе данных, требующей дополнительных операций. Каждый такой скрипт следует

выполнять командой:

```
psql --username=postgres --file=script.sql postgres
```

Эти скрипты могут выполняться в любом порядке, а после выполнения их можно удалить.

3.10.5. Внимание

Обычно к таблицам, задействованным в перестраивающих базу скриптах, опасно обращаться, пока эти скрипты не сделают свою работу; при этом можно получить некорректный результат или плохую производительность. К таблицам, не задействованным в таких скриптах, можно обращаться немедленно.

3.10.5.1. Статистика

Так как статистика оптимизатора не передаётся в процессе работы `pg_upgrade`, вы получите указание запустить соответствующую команду для воссоздания этой информации после обновления. Возможно, для этого вам понадобится установить параметры подключения к новому кластеру.

3.10.5.2. Удалить старый кластер

Если вы удовлетворены результатами обновления, вы можете удалить каталоги данных старого кластера, запустив скрипт, упомянутый в выводе `pg_upgrade` после обновления. Автоматическое удаление невозможно, если в старом каталоге данных находятся дополнительные табличные пространства. Также вы можете удалить каталоги старой инсталляции (например, `bin`, `share`).

3.10.5.3. Возврат к старому кластеру

Если, выполнив команду `pg_upgrade`, вы захотите вернуться к старому кластеру, возможны следующие варианты:

1) если использовался ключ `--check`, в старом кластере ничего не меняется; его можно просто перезапустить;

2) если не использовался ключ `--link`, в старом кластере ничего не меняется; его можно просто перезапустить;

3) если использовался ключ `--link`, у старого и нового кластера могут оказаться общие файлы данных:

- если работа `pg_upgrade` была прервана до начала расстановки ссылок, в старом кластере ничего не меняется; его можно просто перезапустить;
- если вы не запускали новый кластер, старый кластер не претерпел никаких изменений, за исключением того, что при создании ссылки на данные к имени `$PGDATA/global/pg_control` было добавлено окончание `.old`. Чтобы продолжить использование старого кластера, достаточно убрать окончание `.old` из имени файла `$PGDATA/global/pg_control`; после этого старый кластер можно будет перезапустить;
- если вы запускали новый кластер, он внёс изменения в общие файлы, и использовать старый кластер небезопасно. В этом случае старый кластер нужно будет восстановить из резервной копии.

3.10.6. Замечания

`pg_upgrade` не поддерживает обновление баз данных, в которых есть таблицы со столбцами, имеющими следующие системные типы данных `reg*`, ссылающиеся на OID: `regproc`, `regprocedure`, `regoper`, `regoperator`, `regconfig` и `regdictionary`. Обновление `regtype` возможно.

Программа `pg_upgrade` сообщит обо всех актуальных для вашей инсталляции ошибках и потребностях перестроения или переиндексации базы; при этом будут созданы завершающие обновление скрипты, перестраивающие таблицы или индексы. Если вы попытаетесь автоматизировать обновление множества серверов, вы обнаружите, что для кластеров с одинаковыми схемами баз данных потребуются одинаковые действия после обновления; это объясняется тем, что эти действия диктуются схемой базы данных, а не данными пользователей.

Для проверки развёртывания новой версии создайте копию только схемы старого кластера, наполните этот кластер фиктивными данными, и попробуйте обновить его.

Если вы хотите использовать режим ссылок на данные, но не хотите каких-

либо изменений в старом кластере при запуске нового, сделайте копию старого кластера и обновите его в этом режиме. Чтобы получить рабочую копию старого кластера, воспользуйтесь командой `rsync` и создайте предварительную копию кластера при работающем сервере, а затем отключите старый сервер и ещё раз запустите `rsync --checksum`, чтобы привести эту копию в согласованное состояние. Ключ `--checksum` необходим, потому что rsync различает время с точностью только до секунд. При этом вы можете исключить некоторые файлы, например, `postmaster.pid`. Если ваша файловая система поддерживает снимки файловой системы или копирование при записи, вы можете воспользоваться этим для создания копии старого кластера и табличных пространств; при этом важно, чтобы такие снимки и копии файлов создавались одномоментно или, когда сервер баз данных отключён.

3.11. Программа pg_verify_checksums

Pg_verify_checksums - проверить контрольные суммы данных в кластере Синергия-БД.

3.11.1. Синтаксис

```
pg_verify_checksums [параметр...] [[ -D | --
pgdata ]каталог_данных]
```

3.11.2. Описание

`pg_verify_checksums` проверяет контрольные суммы данных в кластере Синергия-БД. Для `pg_verify_checksums` требуется, чтобы сервер был остановлен в штатном режиме. Код состояния будет нулевым в случае ошибок контрольных сумм и ненулевым в противном случае.

3.11.3. Параметры

`-D каталог`

`--pgdata=каталог`

Указывает каталог, в котором располагается кластер баз данных.

`-v`

`--verbose`

Выводить подробные сообщения, в частности список всех проверенных файлов.

`-r файловый_узел_отношения`

Проверять контрольные суммы только в отношении с указанным файловым узлом.

`-V`

`--version`

Сообщить версию `pg_verify_checksums` и завершиться.

`-?`

`--help`

Показать справку по аргументам командной строки `pg_verify_checksums` и завершиться.

3.11.4. Переменные окружения

`PGDATA`

Указывает каталог, в котором располагается кластер баз данных; можно переопределить параметром `-D`.

3.12. Программа pg_waldump

Pg_waldump - вывести журнал предзаписи кластера Синергия-БД в понятном человеку виде.

3.12.1. Синтаксис

```
pg_waldump [параметр...] [начальный_сегмент
[конечный_сегмент] ]
```

3.12.2. Описание

Программа `pg_waldump` показывает содержимое журнала предзаписи (WAL) и прежде всего полезна для целей отладки и обучения.

Эту утилиту может запускать только пользователь, установивший сервер, так как ей требуется доступ на чтение к каталогу данных.

3.12.3. Параметры

Следующие аргументы командной строки задают расположение данных и формат вывода.

`начальный_сегмент`

Начать чтение с указанного файла сегмента журнала. Это неявно определяет каталог, в котором будут находиться файлы, и целевую линию времени.

`конечный_сегмент`

Остановиться после чтения указанного файла сегмента журнала.

`-b`

`--bkp-details`

Выводить подробные сведения о блоках-копиях страниц.

`-e конец`

`--end=конец`

Прекратить чтение в заданной позиции в WAL, а не читать поток до конца.

`-f`

`--follow`

Достигнув конца корректного WAL, проверять раз в секунду поступление новых записей WAL.

`-n предел`

`--limit=предел`

Вывести заданное число записей и остановиться.

`-p путь`

`--path=путь`

Задаёт каталог, содержащий файлы сегментов журнала, либо каталог с подкаталогом `pg_wal`, содержащим такие файлы. По умолчанию в поисках этих файлов просматривается текущий каталог, подкаталог `pg_wal` текущего каталога и подкаталог `pg_wal` каталога `PGDATA`.

`-r менеджер_ресурсов`

`--rmgr=менеджер_ресурсов`

Выводить только записи, созданные указанным менеджером ресурсов. Когда в качестве имени менеджера передаётся `list`, программа выводит только список возможных имён менеджеров ресурсов и завершается.

`-s начало`

`--start=начало`

Позиция в WAL, с которой нужно начать чтение. По умолчанию чтение начинается с первой корректной записи журнала в самом первом из найденных файлов.

`-t линия_времени`

`--timeline=линия_времени`

Линия времени, из которой будут читаться записи журнала. По умолчанию используется значение, заданное параметром начальный_сегмент, если он присутствует, а иначе - `1`.

`-V`

`--version`

Вывести версию `pg_waldump` и завершиться.

`-x ид_транзакции`

`--xid=ид_транзакции`

Вывести только записи, относящиеся к указанной транзакции.

`-z`

`--stats[=record]`

Вывести общую статистику (число и размер записей и образов полных страниц) вместо отдельных записей. Возможен вариант получения статистики по записям, а не по менеджерам ресурсов.

`-?`

`--help`

Вывести справку об аргументах командной строки `pg_waldump` и завершиться.

3.12.4. Замечания

Когда сервер работает, результаты могут быть некорректными.

Выводятся записи только указанной линии времени (или линии времени по умолчанию, если она не задана явно). Записи в других линиях времени игнорируются.

`pg_waldump` не будет читать файлы WAL с расширением `.partial`. Если требуется прочитать такие файлы, расширение `.partial` нужно убрать из их имён.

3.13. Программа postgres

Postgres - Сервер баз данных Синергия-БД.

3.13.1. Синтаксис

```
postgres [параметр...]
```

3.13.2. Описание

`postgres` это сервер баз данных Синергия-БД. Для получения доступа к базе данных клиент устанавливает соединение (локально или по сети) с сервером `postgres`. После установки соединения сервер `postgres` поднимает выделенный процесс для его обслуживания.

Один экземпляр `postgres` всегда управляет данными ровно одного кластера баз данных. Кластер - это коллекция баз данных, хранящихся в файловой системе в определённом размещении («области данных»). На одном физическом сервере можно запустить несколько экземпляров `postgres` одновременно, при условии, что они используют различные области данных и порты. При запуске `postgres` необходимо указать размещение данных, которое задаётся в параметре `-D` или переменной окружения `PGDATA`, значение по умолчанию отсутствует. Обычно `-D` или `PGDATA` указывает на каталог, созданный во время развёртывания кластера с помощью `initdb`.

По умолчанию `postgres` запускается не в фоновом режиме, а вывод журнала осуществляет в стандартный поток ошибок. На практике `postgres` должен

запускаться в фоновом режиме, возможно, при старте системы.

Команду `postgres` также возможно использовать в однопользовательском режиме. В основном этот режим используется на этапе начальной загрузки при выполнении `initdb`. Иногда также применяется в целях отладки или после аварийного сбоя. Стоит обратить внимание, что однопользовательский режим не совсем корректно использовать для отладки, так как условия далеки от реальных боевых. При запуске из командной строки в однопользовательском режиме можно вводить запросы и видеть результаты их выполнения на экране в форме пригодной для разработки, нежели конечных пользователей. Также в этом режиме ID сессионного пользователя выставляется в единицу, а пользователь неявно наделяется правами суперпользователя. Существование пользователя необязательно, поэтому режим можно использовать в некоторых ситуациях при повреждениях системных каталогов.

3.13.3. Параметры

Программа `postgres` принимает следующие аргументы командной строки. От необходимости вводить большинство этих параметров можно избавиться, записав их в файл конфигурации. Некоторые (безопасные) параметры можно также задать со стороны подключающегося клиента (в зависимости от приложения), чтобы они применялись только к одному сеансу. Например, если установлена переменная окружения `PGOPTIONS`, клиенты на базе `libpq` передадут эту строку серверу, который воспримет её как параметры, передаваемые в командной строке `postgres`.

3.13.3.1. Параметры общего назначения

`-B количество-буферов`

Устанавливает количество разделяемых между процессами буферов. Значение по умолчанию выбирается автоматически при развёртывании кластера с помощью `initdb`. Установка флага аналогична конфигурации параметра `shared_buffers`.

`-c имя=значение`

Устанавливает заданный параметр времени исполнения. Большинство других

параметров командной строки на самом деле представляют собой краткие формы такого присвоения значений параметрам. Для установления нескольких параметров -c можно указывать многократно.

`-C имя`

Отображает значение заданного параметра времени исполнения и прерывает дальнейшее выполнение (подробнее см. выше). Можно применять на работающем сервере, при этом будут возвращены значения `postgresql.conf` с учётом проведённых в рамках вызова изменений. Значения, переданные при старте кластера, не отображаются.

Параметр предназначен для приложений, взаимодействующих с сервером, например, `pg_ctl`, и запрашивающих параметры конфигурации. Пользовательские приложения должны использовать команду `SHOW` или представление `pg_settings`.

`-d уровень-отладки`

Устанавливает уровень отладки (от `1` до `5`). Чем выше значение, тем подробнее осуществляется вывод в журнал сервера. Также возможно передать `-d 0` для отдельной сессии, что предотвратит в её рамках влияние выставленного для `postgres` значения.

`-D datadir`

Указывает размещение конфигурационных файлов базы в пределах файловой системы.

`-e`

Устанавливает формат вводимых дат по умолчанию в «`European`» с последовательностью значений `DMY`. Также влияет на вывод дня, идущего перед значением месяца.

`-F`

Отключает вызовы `fsync` для увеличения производительности, но с увеличением рисков потери данных в случае краха системы. Этот параметр работает аналогично параметру конфигурации `fsync`. Внимательно прочтите документацию перед использованием данного параметра!

`-h компьютер`

Указывает IP-адрес или имя компьютера, на котором сервер `postgres` принимает клиентские подключения по TCP/IP. Значением может быть список адресов, разделённых запятыми, либо символ `*`, обозначающий все доступные интерфейсы. Если значение опущено, то подключения принимаются только через Unix-сокеты. По умолчанию принимаются подключения только к `localhost`. Флаг работает аналогично конфигурационному параметру `listen_addresses`.

`-i`

Позволяет клиентам подключаться по TCP/IP. Без этого параметра допускаются лишь локальные подключения. Действие этого параметра аналогично действию параметра конфигурации `listen_addresses` со значением `*` в `postgresql.conf` или ключа `-h`.

Параметр устарел, так как не даёт полной функциональности `listen_addresses`. Лучше устанавливать значение `listen_addresses` напрямую.

`-k каталог`

Указывает каталог Unix-сокета, через который `postgres` будет принимать подключения. Значением параметра может быть список каталогов через запятую. Если это значение пустое, использование Unix-сокетов запрещается, разрешаются только подключения по TCP/IP. По умолчанию выбирается каталог `/tmp`, но его можно сменить на этапе компиляции. Этот параметр действует аналогично параметру конфигурации `unix_socket_directories`.

`-l`

Включает поддержку безопасных соединений.

`-N максимальное количество соединений`

Устанавливает максимально возможное количество одновременных клиентских соединений. Значение по умолчанию устанавливается автоматически на этапе развёртывания с помощью `initdb`. Флаг работает аналогично конфигурационному параметру `max_connections`.

-o дополнительные-параметры

Аргументы командной строки, поступившие через дополнительные-параметры, передаются всем обслуживающим процессам, запускаемым этим процессом postgres.

Пробелы в строке дополнительные-параметры воспринимаются как разделяющие аргументы, если перед ними нет обратной косой черты (\); чтобы представить обратную косую черту буквально, продублируйте её (\\). Также можно задать несколько аргументов, используя -o несколько раз.

Использование этого параметра считается устаревшим, так как на данный момент все параметры postgres можно задать в командной строке.

-p порт

Указывает порт TCP/IP или расширение файла локального Unix-сокета, через который postgres принимает подключения клиентских приложений. По умолчанию принимает значение переменной окружения PGPORT, или, если значение PGPORT не установлено, то используется значение, установленное на этапе компиляции (обычно это 5432). Если значение порта меняется, то на стороне клиентов это необходимо учитывать, установив, либо PGPORT, либо флаг командной строки.

-s

Отображает информацию о времени и другую статистику после каждой выполненной команды, что полезно для оценки производительности во время настройки количества буферов.

-S рабочая-память

Указывает объём рабочей памяти для внутренних сортировок и хешированных строк во избежание использования временных файлов. Обратитесь к описанию параметра work_mem.

-V

--version

Отображает версию postgres и прерывает дальнейшее выполнение.

--имя=значение

Устанавливает заданный параметр времени исполнения. Является короткой формой ключа `-c`.

`--describe-config`

Выводит значения конфигурационных переменных сервера, их описаний и значений по умолчанию в формате команды `COPY` со знаком табуляции в качестве разделителя. В основном это предназначено для средств администрирования.

`-?`

`--help`

Выводит помощь по аргументам команды `postgres`.

3.13.3.2. Параметры для внутреннего использования

Далее описанные параметры, в основном, применяются в целях отладки, а в некоторых случаях при восстановлении сильно повреждённых баз данных. Их описание приведено для системных разработчиков Синергия-БД, поэтому они могут быть изменены без уведомления.

`-f { s | i | o | b | t | n | m | h }`

Запрещает использование специфических методов сканирования и объединения: `s` и `i` выключают последовательное сканирование и по индексу соответственно, а `o`, `b` и `t` выключает сканирование только по индексу, сканирование по битовым векторам, и сканирование по ID кортежей соответственно, в то время как `n`, `m` и `h` выключает вложенные циклы, слияния и хеширование соответственно.

Ни последовательное сканирование, ни вложенные циклы невозможно выключить полностью. Флаги `-fs` и `-fn` просто указывают планировщику избегать выполнения этих операций при наличии других альтернатив.

`-n`

Параметр предназначен для отладки сервера в случае аномального завершения процесса. Обычная практика в таком случае - завершение порождённых процессов с дальнейшей инициализацией разделяемой памяти и семафоров. Это связано с тем, что потерянный процесс мог повредить область разделяемой памяти. Параметр указывает `postgres` не производить повторной инициализации общих структур

данных, что позволяет произвести дальнейшую отладку текущего состояния памяти и семафоров.

`-O`

Разрешает модифицировать структуру системных таблиц. Используется командой `initdb`.

`-P`

Игнорировать системные индексы при чтении, но продолжать обновлять их при изменениях системных таблиц. Это используется при их восстановлении после повреждения.

`-t pa[rser] | pl[anner] | e[xecutor]`

Выводит статистику по времени исполнения каждого запроса в контексте каждого системного модуля. Использование флага совместно с `-s` невозможно.

`-T`

Параметр предназначен для отладки сервера в случае аномального завершения процесса. Обычная практика в таком случае - завершение порождённых процессов с дальнейшей инициализацией разделяемой памяти и семафоров. Это связано с тем, что потерянный процесс мог повредить область разделяемой памяти. Параметр указывает postgres на необходимость остановки порождённых процессов сигналом `SIGSTOP`, но не завершит их, что позволяет разработчикам сделать снимки памяти процессов.

`-v протокол`

Указывает для данного сеанса версию протокола взаимодействия сервера с клиентом. Флаг используется лишь для внутренних целей.

`-W секунды`

При старте сервера производится задержка на указанное количество секунд, после чего производится процедура аутентификации, что позволяет подключить отладчик к процессу.

3.13.3.3. Параметры для однопользовательского режима

Следующие параметры применимы только для однопользовательского режима.

`--single`

Устанавливает однопользовательский режим. Параметр должен идти первым в командной строке.

`база_данных`

Указывает имя базы данных, к которой производится подключение. Параметр должен идти последним в командной строке. Если не указан, то используется имя текущего системного пользователя.

`-E`

Выводить все команды в устройство стандартного вывода прежде чем выполнять их.

`-j`

Считать признаком окончания ввода команды точку с запятой с двумя переводами строки, а не один перевод строки.

`-r имя_файла`

Отправляет вывод журнала сервера в файл `filename`. Этот параметр применяется лишь при запуске из командной строки.

3.13.4. Переменные окружения

`PGCLIENTENCODING`

Кодировка, используемая клиентом по умолчанию. Может переопределяться на стороне клиента, а также устанавливаться в конфигурационном файле сервера.

`PGDATA`

Каталог размещения данных кластера по умолчанию

`PGDATESTYLE`

Значение по умолчанию для параметра времени исполнения DateStyle. Применение этой переменной является устаревшим.

`PGPORT`

Порт по умолчанию, лучше устанавливать в конфигурационном файле.

3.13.5. Диагностика

Ошибки с упоминанием о `semget` или `shmget` говорят о возможной

необходимости проведения более оптимального конфигурирования ядра. Отложить переконфигурирование можно, уменьшив `shared_buffers` для снижения общего потребления разделяемой памяти Синергия-БД и/или уменьшив `max_connections` для снижения затрат на использование семафоров.

Необходимо внимательно проверять сообщения об ошибке с упоминанием о другом запущенном экземпляре, например, с использованием команды

```
$ps ax | grep postgres
```

или

```
$ps -ef | grep postgres
```

в зависимости от ОС. Если есть полная уверенность, что противоречий нет, необходимо самостоятельно удалить упомянутый в сообщении запирающий файл и повторить попытку.

Упоминание о невозможности привязки к порту в сообщениях об ошибках может указывать на то, что он уже занят другим процессом помимо Синергия-БД. Также сообщение может возникнуть при мгновенном рестарте postgres на том же скпорту. В этом случае нужно немного подождать, пока ОС не закроет порт, и повторить попытку. Ещё возможна ситуация, в которой используется резервный системный порт. Например, многие Unix-подобные ОС резервируют «доверительные» порты от 1024 и ниже, и лишь суперпользователь имеет к ним доступ.

3.13.6. Замечания

Для комфортного запуска и остановки сервера можно использовать утилиту `pg_ctl`.

Если возможно, *не используйте* сигнал `SIGKILL` для головного процесса `postgres`. В этом случае `postgres` не освободит системные ресурсы, например, разделяемую память и семафоры. Это может привести к проблемам при повторном запуске `postgres`.

Для корректного завершения `postgres` используются сигналы `SIGTERM`, `SIGINT` или `SIGQUIT`. При первом будут ожидаться все дочерние процессы до их

завершения, второй приведёт к принудительному закрытию соединений, а третий - к незамедлительному выходу без корректного завершения, приводящему к необходимости выполнения процедуры восстановления на следующем старте.

Получая сигнал SIGHUP, сервер перечитывает свои файлы конфигурации. Также возможно отправить SIGHUP отдельному процессу, но это чаще всего бессмысленно.

Для отмены исполняющегося запроса, отправьте SIGINT обслуживающему его процессу. Для чистого завершения серверного процесса отправьте ему SIGTERM. Также см. pg_cancel_backend и pg_terminate_backend, которые являются аналогами в форме SQL-инструкций.

Сервер postgres обрабатывает SIGQUIT для завершения дочерних процессов в грязную, и сигнал *не должен* отправляться пользователем. Также не стоит посылать SIGKILL серверному процессу - головной postgres процесс расценит это как аварию и принудительно завершит остальные порождённые, как это было бы сделано при процедуре восстановления после сбоя.

3.13.7. Ошибки

3.13.7.1. Однопользовательский режим

Для запуска сервера в однопользовательском режиме используется, например, команда

```
postgres --single -D /var/lib/sdb/sdb-11/data другие
параметры my_database
```

Необходимо указать корректный путь к каталогу хранения данных в параметре -D, или установить переменную окружения PGDATA. Также замените имя базы данных на необходимое.

Обычно перевод строки в однопользовательском режиме сервер воспринимает как завершение ввода команды; точка с запятой не имеет для него такого значения, как для psql. Поэтому, чтобы ввести команду, занимающую несколько строк, необходимо добавлять в конце каждой строки, кроме последней, обратную косую

черту. Обратная косая черта и следующий за ней символ перевода строки автоматически убираются из вводимой команды. Заметьте, что это происходит даже внутри строковой константы или комментария.

Но если применить аргумент командной строки `-j`, одиночный символ перевода строки не будет завершать ввод команды; это будет делать последовательность «точка с запятой, перевод строки, перевод строки». То есть для завершения команды нужно ввести точку с запятой, и сразу за ней пустую строку. Просто точка с запятой, дополненная переводом строки, в этом режиме не имеет специального значения. Внутри строковых констант и комментариев такая завершающая последовательность воспринимается в том же ключе.

Вне зависимости от режима ввода, символ точки с запятой, введённый не прямо перед или в составе последовательности завершения команды, воспринимается как разделитель команд. После ввода завершающей последовательности введённые с разделителями несколько операторов будут выполняться в одной транзакции.

Для завершения сеанса введите символ конца файла (EOF, обычно это сочетание **Control**+**D**). Если вы вводили текст после окончания ввода последней команды, символ EOF будет воспринят как символ завершения команды, и для выхода потребуется ещё один EOF.

Заметьте, что однопользовательский режим не предоставляет особых возможностей для редактирования команд (например, нет истории команд). Также в однопользовательском режиме не производятся никакие фоновые действия, например, не выполняются автоматические контрольные точки или репликация.

3.13.8. Примеры

Для запуска postgres в фоновом режиме с параметрами по умолчанию:

```
$nohup postgres >logfile 2>&1 </dev/null &
```

Для запуска postgres с определённым портом, например, 1234:

```
$postgres -p 1234
```

Для соединения с помощью `psql` укажите этот порт в параметре `-p`:

```
$psql -p 1234
```

или в переменной окружения PGPORT:

```
$export PGPORT=1234
```

```
$psql
```

Именованный параметр времени исполнения можно указать одним из приведённых способом:

```
$postgres -c work_mem=1234
```

```
$postgres --work-mem=1234
```

Любой из методов переопределяет значение work_mem конфигурации postgresql.conf. Символ подчёркивания в именах можно указать и в виде тире. Задавать параметры обычно (не считая кратковременных экспериментов) лучше в postgresql.conf, а не в аргументах командной строки.

3.14. Программа postmaster

Postmaster - Сервер баз данных Синергия-БД

3.14.1. Синтаксис

```
postmaster [параметр...]
```

3.14.2. Описание

postmaster это устаревшее название postgres.

4. УСТАНОВКА МОДУЛЯ МОНИТОРИНГА И УПРАВЛЕНИЯ СУБД

На сервер, с которого мы будем снимать метрики необходимо:

4.1. Установить sdb-agent

Установку модуля мониторинга необходимо производить на сервере, который подключается к системе мониторинга. На сервере должна быть развернута СУБД «Синергия-БД» версии, не ниже 13. Модуль должен быть установлен на ОС специального назначения «Astra Linux Special Edition» РУСБ.10015-01 версии 1.7.

Для установки необходимо сделать следующее:

- подключить репозиторий;
- импортировать открытый ключ для репозитория: apt-key add /media/cdrom/keys/GPG-KEY-POSTGRES;
- добавить репозиторий apt-cdrom add –m;
- обновить список пакетов apt-get update;
- установить sdb-agent из репозитария sudo apt-get install sdb-agent.

Для получения информации об ожиданиях и нагрузке на дисковую подсистему необходимо дополнительно установить расширения wait-sampling , stat-kcache, выполнив команды:

- sudo apt-get install sdb-wait-sampling-13;
- sudo apt-get install sdb-stat-kcache-13.

4.2. Настройка sdb-agent

Для настройки sdb-agent необходимо настроить расширения, необходимые для мониторинга СУБД:

sudo -u postgres psql -U postgres -c 'alter system set shared_preload_libraries = pg_stat_statements,pg_stat_kcache,pg_wait_sampling;'

sudo service postgresql-sdb-13 restart

sudo -u postgres psql -U postgres -d postgres -c 'create extension pg_stat_statements'

sudo -u postgres psql -U postgres -d postgres -c 'create extension pg_stat_kcache'

sudo -u postgres psql -U postgres -d postgres -c 'create extension pg_wait_sampling'

Создать базу данных (или использовать уже существующую БД, например postgres) и заполнить ее служебными функциями. Для этого надо выполнить команду:

sudo -u postgres sdb-agent bootstrap -U postgres -M postgres

Отредактировать файл /etc/sdbagent/agent.conf:

- в секции [Zabbix], параметр client поставить равным имени, по которому данный сервер будет доступен в Zabbix-server, например server_test;
- в секции [Zabbix], параметр address поставить равным имени IP адресу Zabbix-server, например 192.168.22.17;
- в секции [postgres] параметр user поставить равным имени пользователя, под которым sdb-agent подключается к СУБД. В данном примере postgres;
- в секции [postgres] параметр database поставить равным имени базы, к которой sdb-agent подключается для сбора метрик. В данном примере postgres;
- в секции [postgres] параметр password поставить равным паролю, под которым sdb-agent подключается к СУБД. В данном примере postgres;
- в секции [postgres] параметр host поставить равным имени хоста, где установлена СУБД.

Пример файла настроек представлен ниже.

```
user@astra-smolensk:~$ cat /etc/sdb-agent/agent.conf
[Zabbix]
; Zabbix server address
address = 192.168.22.17
; configured 'Host name' of client in Zabbix
client = server_test
[postgres]
enabled = True
host = localhost
user = postgres
database = postgres
password = postgres
port = 5432
query_timeout = 10
[system]
enabled = True
[log]
file = /var/log/sdb-agent/agent.log
level = INFO
```

4.3. Управление sdb-agent

Остановка sdb-agent производится следующей командой:

```
sudo service sdb-agent stop
```

Логи sdb-agent пишутся в файл, указанный в файле /etc/sdb-agent/agent.conf, в секции [log], параметр file. В данном примере /var/log/sdb-agent/agent.log.

4.4. Установка sdb-agent

Для работоспособности планировщика задач в графическом интерфейсе необходимо дополнительно установить расширение synergydb_cron_scheduler, выполнив команду:

sudo apt-get install synergydb_cron_scheduler-1.5;

Подготавливаем сервер мониторинга:

- Разворачиваем виртуальную машину, которая будет снимать метрики и отображать в графическом интерфейсе.
- В настройках устанавливаем тип подключение «Сетевой мост».
- Доступ к серверу осуществляется по логину: root и паролю: P@ssword1.
- Проверяем настройки сетевых устройств командой ip a.
- Настройка сетевых устройств хранятся в /etc/network/interfaces.

Пример настройки устройства

auto eth2

iface eth2 inet dhcp

Для применения настроек необходимо перезапустить сервер.

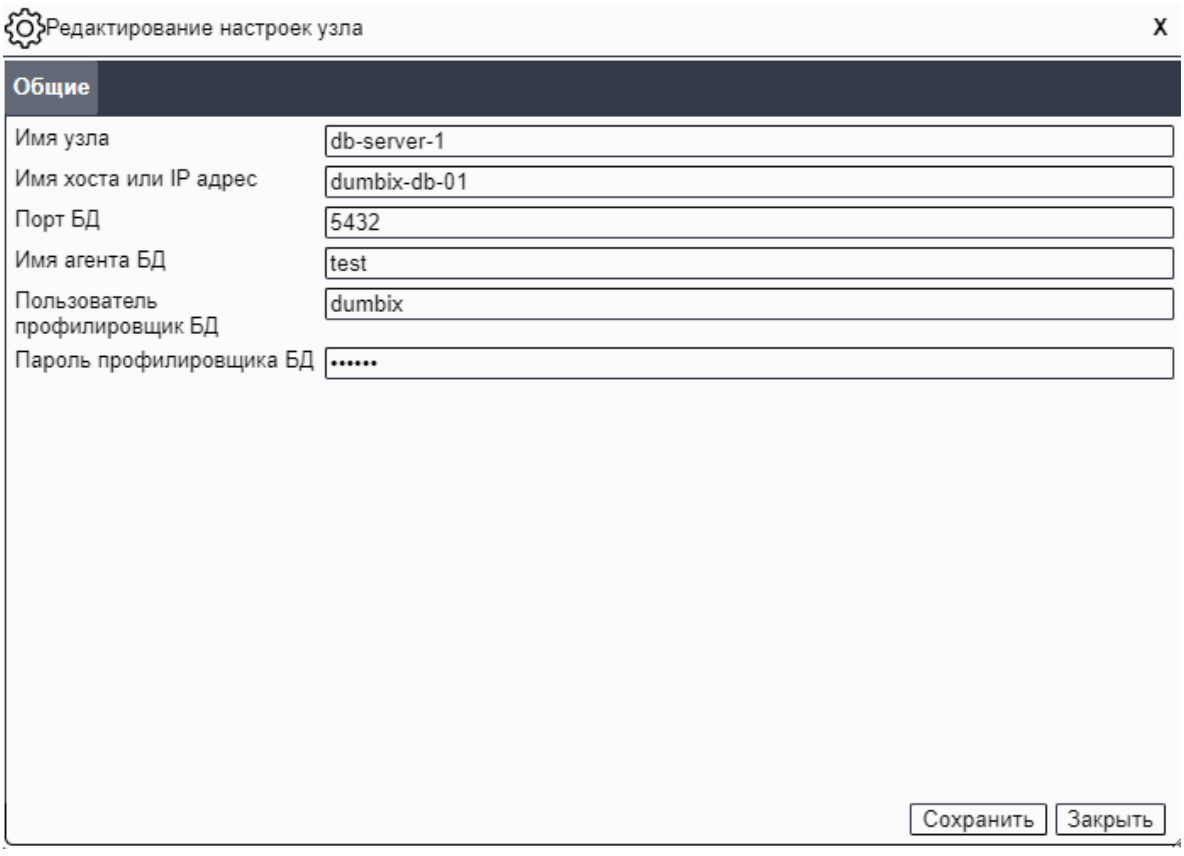
Лог программы dumbix находятся /opt/dumdux/php/var/log.

Открываем браузер – вводим адрес, указанный в настройках сетевых устройств, которые выводятся командой ip a.

Вход в Apache сервер осуществляется по логину – dumbix и паролю – dumbix.

В открывшейся странице вводим сетевой адрес, указанный в строке браузера.

Настроем узел СУБ, нажав добавить узел.



Выбираем сохраненный узел, с которого собираемся снимать метрики.

На вкладках «Управление базами данных», «Управление резервным копированием» и «Планировщик» вводим логин и пароль пользователя БД сервера, с которого снимаются метрики.

Лист регистрации изменений									
Номера листов (страниц).					*Всего листов (страниц) в докум.*	*№ документа*	*Входящий № сопроводительного докум. и дата*	*Подп.*	*Дата*
Изм.	*измененных*	*замененных*	*новых*	*аннулированных*					